LARGE-SCALE LINEAR PROGRAMMING

Proceedings of a IIASA Workshop,
2–6 June 1980

Volume 1

George B. Dantzig, M.A.H. Dempster,
and Markku Kallio
Editors

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
Laxenburg, Austria
1981

International Standard Book Number 3-7045-0006-2

Volumes in the *IIASA Collaborative Proceedings Series* contain papers offered at IIASA professional meetings, and are designed to be issued promptly, with a minimum of editing and review.

The views or opinions expressed in this volume do not necessarily represent those of the Institute or the National Member Organizations that support it.

FOREWORD

The International Institute for Applied Systems Analysis is a nongovernmental, multidisciplinary, international research institution whose goal is to bring together scientists from around the world to work on problems of common interest.

IIASA pursues this goal, not only by pursuing a research program at the Institute in collaboration with many other institutions, but also by holding a wide variety of scientific and technical meetings. Often the interest in these meetings extends beyond the concerns of the participants, and proceedings are issued. Carefully edited and reviewed proceedings occasionally appear in the *International Series on Applied Systems Analysis* (published by John Wiley and Sons Limited, Chichester, England); edited proceedings appear in the *IIASA Proceedings Series* (published by Pergamon Press Limited, Oxford, England).

When relatively quick publication is desired, unedited and only lightly reviewed proceedings reproduced from manuscripts provided by the authors of the papers appear in this new *IIASA Collaborative Proceedings Series.* Volumes in this series are available from the Institute at moderate cost.

PREFACE

During the week of June 2–6, 1980, the System and Decision Sciences Area of the International Institute for Applied Systems Analysis organized a workshop on large-scale linear programming in collaboration with the Systems Optimization Laboratory (SOL) of Stanford University, and cosponsored by the Mathematical Programming Society (MPS). The participants in the meeting were invited from amongst those who actively contribute to research in large-scale linear programming methodology (including development of algorithms and software). Although primarily methodologically oriented scientists attended the workshop, its theme was the improvement of the long range applicability of linear programming (LP) techniques. Besides the exchange of ideas and experience – and suggestions for future research directions and international cooperation – fostered by the meeting, it was a general feeling of the participants that a proceedings would reflect the current state of large-scale linear programming in both East and West.

To this end, it was considered important to produce the proceedings volumes in a lecture note format as quickly as possible, so as to secure a complete record of the papers presented at the workshop – including those destined for publication elsewhere – together with several papers solicited by the editors in order to extend coverage. In some cases, papers presented at IIASA have been revised by their authors in the two months following the meeting; in others, no revisions have been made. Although a standard title page format has been used, the papers have been largely reproduced from camera-ready copy supplied by their authors. Most have not been refereed, edited or proofread for typographical errors. Papers are grouped together in chapters by topic and are listed in alphabetical order by author in each chapter.

The first volume of these *Proceedings* contains five chapters. The first is an historical review by George B. Dantzig of his own and related research in time-staged linear programming problems. Chapter 2 contains five papers which address various techniques for exploiting sparsity and degeneracy in the now standard LU decomposition of the basis used with the simplex algorithm for standard (unstructured) problems. The six papers of Chapter 3 concern aspects of variants of the simplex method which take into account through basis factorization the specific block-angular structure of constraint matrices generated by dynamic and/or stochastic linear programs. By means of these techniques it is hoped to extend the size of solvable LP's beyond the range of current commercial codes for specific problems in the fields of energy, resource and macro/economic modeling (including economic planning models). In Chapter 4, five papers address extensions of the original Dantzig–Wolfe procedure for utilizing the structure of planning problems by decomposing the original LP into LP subproblems coordinated by a relatively simple LP master problem of a certain type. Two of these papers concern the recent idea of applying this approach recursively to the subproblems themselves. Chapter 5 contains four papers which constitute a mini-symposium on the now famous Shor-Khachian ellipsoidal method applied to both real and integer linear programs. This completes the description of the contents of Volume 1.

The first chapter of Volume 2 contains three papers on non-simplex methods for linear programming. This chapter concludes reports in the mainstream of current research on solution algorithms in large-scale linear programming. The remaining chapters of Volume 2 concern more peripheral — but no less important — topics of present interest in the field. Techniques for exploiting network structure in LP problems are the topic of the three papers of Chapter 7. In the next chapter, the emphasis turns to the practically crucial and inter-related issues of automatic LP model generation and structure identification. The seven papers of this chapter discuss software both for model and matrix generation and for model reduction through detection of imbedded special constraint structure. The final chapter, 9, contains a number of applications of large-scale LP techniques to practical problems in industrial and agricultural production and economic planning. Some of these involve multi-criteria optimization, and two of the eight papers deal explicitly with implementations of new approaches to the multi-criteria problem. A bibliography of large-scale linear programming research completes Volume 2.

The editors wish to take this opportunity on behalf of the participants to thank IIASA, SOL and MPS for their cooperation and to thank IIASA as well as various Academies of Sciences and governmental agencies of several countries for making the resources available to hold the Large-scale Linear Programming Workshop and to publish these *Proceedings.* In particular, we are grateful to the Communications Department at IIASA for their cheerful cooperation in expediting publication of this record of an important and memorable international meeting.

George B. Dantzig
M.A.H. Dempster
Markku Kallio

Stanford, California
August 1980

CONTENTS

VOLUME 1

VOLUME 2

TIME-STAGED METHODS IN LINEAR PROGRAMMING: COMMENTS AND EARLY HISTORY

TIME-STAGED METHODS IN LINEAR PROGRAMMING: COMMENTS AND EARLY HISTORY

George B. Dantzig

Department of Operations Research
Stanford University

The Workshop on Large-scale Linear Programming reflects the active research taking place in many parts of the world along a very broad front, namely on:

- the theory of solution,
- software development,
- experiments on representative problems,
- application to real problems,
- matrix input generators,
- matrix analyzers,
- output report generators,
- alternative methods of formulation.

This paper is a historical review of the author's interest in one important facet of this field – the solution of time-staged programs. Indeed it was dynamic LP that initiated the linear programming field back in 1947. Over the years, many good ideas have been proposed, some that still merit serious consideration. This Workshop may provide the answer to the question whether or not we have begun at last to achieve the efficiency of solution necessary for successful application.

This paper is a more polished version of the talk which I delivered opening the International Institute for Applied Systems Analysis Workshop on Large-Scale Linear Programming at Laxenburg Austria, June 2-6, 1980. Except for a short review of large-scale methods also presented, but omitted here, my perspective is historical.

TIME-STAGED STAIRCASE SYSTEMS

The first formal papers about the new field of linear programming (that started in 1947) appeared in Econometrica July - October 1949. At the very beginning, the emphasis was on solving time-staged (dynamic) linear programs. That this is so, is clear from the following quote from [1]:

> This paper is concerned with improved techniques of program planning, particularly as they apply to the scheduling of activities over time within an organization or economy in which the activities must share in the use of limited amounts of various commodities. The contemplated use of electronic computers for rapidly computing programs and the assumptions underlying the mathematical model are discussed. The paper is concluded by an illustrative example, [Berlin Airlift, A Time-Staged Dynamic Linear Program].
>
> The Mathematical Model discussed here is a generalization of the Leontief Inter-Industry Model. It is closely related to the one found in von Neumann's paper "A Model of General Economic Equilibrium". Its chief points of difference lie in its emphasis on dynamic, rather than equilibrium or steady states. Its purpose is close control of an organization--

> hence it must be quite detailed; it is designed to handle highly dynamic problems--hence greater emphasis on time lags and capital equipment; it takes into consideration the many different ways of doing things--hence it explicitly introduces alternative activities; and it recognizes that any particular choice of a dynamic program depends on the "objectives" of the "economy",--hence the selection and types of activities are made to depend on the maximization of an objective function.

In the companion paper [2], the time staged staircase model is displayed and its relationship to Leontief Input-Output model and continuous-time models is discussed:

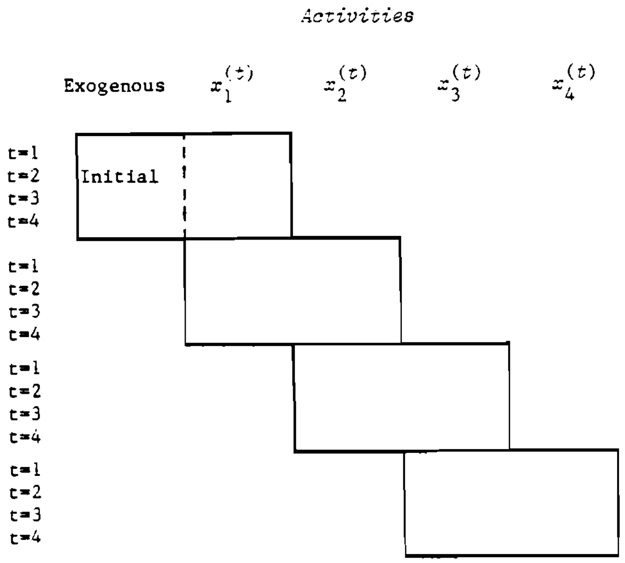
$$
\begin{array}{llllll}
\alpha^{(1)}x^{(1)} & \cdots & & & & = a^{(1)} \\
-\bar{\alpha}^{(1)}x^{(1)} & +\alpha^{(2)}x^{(2)} & \cdots & & & = a^{(2)} \\
\cdots & -\bar{\alpha}^{(2)}x^{(2)} & +\alpha^{(3)}x^{(3)} & \cdots & & = a^{(3)} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
 & & & -\bar{\alpha}^{(T-1)}x^{(T-1)} & +\alpha^{(T)}x^{(T)} & = a^{(T)} \\
\gamma^{(1)}x^{(1)} & + \cdots & & & +\gamma^{(T)}x^{(T)} & = \max,
\end{array}
$$

> where the $x^{(t)}$ are vectors of nonnegative elements.
>
> When the matrices $\alpha^{(t)}$ and $\bar{\alpha}^{(t)}$ $(t=1,2,\ldots,T)$ are square and nonsingular, a direct solution is possible that may lead, however, to negative and nonnegative activity levels (in which case no feasible solution exists).
>
> It should be noted that the general mathematical problem reduces in the linear programming case to consideration of a system of equations of nonnegative variables whose matrix of coefficients is composed mostly of blocks of zeros except for submatrices along and just off the "diagonal". Thus any good computational technique for solving programs would probably take advantage of this fact.

Having formulated the time-staged model, it soon became clear that the techniques at hand at the time were inadequate. In a companion paper [3], first presented in 1949, appeared the following statement:

> Computing techniques are now available for solution of small linear programming problems. However, for accurate over-all Air Force planning, the size of the required model is such that conventional punched card computing equipment, or even the interim electronic computer being built for the Air Force by the National Bureau of Standards, is not sufficiently powerful to cope satisfactorily with the problem of choosing the optimum activities and activity levels over time.

In order to obtain a programming procedure which would be immediately useful with presently available computing equipment, we have been *forced* to use a determinate, and hence less general formulation of the programming problem that parallels closely the staff procedure.



We have called this a *triangular model* because in it the matrix of detached coefficients, when arragned as in the Table, and omitting the "initial" part, assumes a triangular form, with all coefficients above and to the right of the principal diagonal being zero. Thus the activities and items are so ordered that the levels of any one activity over time depend only on the levels of the activities which precede it in the hierarchy. This means that in the computation of the program we successively work down the hierarchy, at each step solving completely for the levels of each activity in each of the time periods before proceeding to the next activity (see figure above).

The triangular model technique is a powerful empirical method when there is a natural hierarchy of activities and output items. Certain energy models, for example, currently in vogue use such an approach.

BLOCK TRIANGULARITY

My paper [4], is my first on methods for solving large systems:

> With the growing awareness of the potentialities of the linear programming approach to both dynamic and static problems of industry, of the economy, and of the military, the main obstacle toward full application is the inability of current computational methods to cope with the magnitude of the technological matrices for even the simplest situations. However, in certain cases, such as the now classical Hitchcock-Koopmans transportation model, it has been possible to solve the linear inequality system in spite of size because of simple properties of the system. This suggests that considerable research be undertaken to exploit certain special matrix structures in order to facilitate ready solution of larger systems.
>
> Indeed, recent computational experience has made it clear that standard techniques such as the simplex algorithm, which have been used to solve successfully general systems involving one hundred equations (in any reasonable number of nonnegative unknowns), are too tedious and lengthy to be practical for extensions much beyond this figure. Our purpose here will be to develop short-cut computational methods for solving an important class of systems whose matrices may be generally described as "block triangular".
>
> By "block" triangular we mean that if one partitions the matrix of coefficients of the technology matrix into submatrices, the submatrices (or blocks) considered as elements form a *triangular system,*
>
> $$\begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \cdots & \cdots & \cdots & \\ A_{T1} & A_{T2} & \cdots\cdots & A_{TT} \end{bmatrix}$$
>
> For example, von Neumann, in considering a constantly expanding economy, developed a linear dynamic model whose matrix of coefficients may be written in the form,
>
> $$\begin{bmatrix} A & & & & \\ -B & A & & & \\ & -B & A & & \\ & & & & \\ & & & -B & A \end{bmatrix}$$
>
> where A is the submatrix of coefficients of activities initiated in period t, and B is the submatrix of *output* coefficients of these activities in the following period.

Now the main obstacle toward the full application of standard linear programming techniques to dynamic systems is the magnitude of the matrix for even the simplest situations. For example, a trivial 15-activity--7-item static model, when set up as a 12-period dynamic model, would become a 180-activity by 84-item system, which is considered a large problem for application of the standard simplex method. A fancy model involving, say, 200 activities and 100 items for a static case would become a 2000 x 1000 matrix if recast as a 10-period model. It is clear that dynamic models must be treated with special tools if any progress is to be made toward solutions of these systems.

From a computational point of view, there are a number of observed characteristics of the dynamic models which are often true for static models as well.
These are:

(1) The matrix (or its transpose) can be arranged in triangular form
(2) Most submatrices A_{ij} are either zero matrices or composed of elements, most of which are zero.
(3) A basis for the simplex method is often block triangular with its diagonal submatrices square and nonsingular (referred to as a "square block triangular" basis).
(4) For dynamic models similar type activities are likely to persist in the basis for several periods.

To illustrate, consider a dynamic version of the Leontief model in which (a) alternative activities are permitted (a simple case would be where steel can be obtained from direct production or storage); (b) inputs to an activity for production in the tth time period may occur in the same or earlier time periods. It can be shown in this model that (a) a basic solution will have exactly m activities in each time period (where m = number of time dependent equations), (b) each shift in basis will bring in a substitute activity in the same time period, and (c) optimization can be carried out as a sequence of one-period optimization problems; i.e., the optimum choice of activities (but not their amounts) can be determined for the first time period (independent of the later periods) this permits a determination for the second time period (independent of the later periods), et cetera.

When flow models are replaced with more complex models which include initial inventories, capacities, and the building of new capacities, the ideal structure of a basis (see third characteristic above) no longer holds. However, tests (carried on since 1950) on a number of cases indicate that bases, while often *not square block triangular in the sense above, could be made so by changing relatively few columns in the basis* (e.g., one or two activities in small models). This characteristic of *near-square block* triangularity of the basis, i.e., with nonsingular square submatrices down the diagonal, is, of course, computationally convenient and this paper will be concerned with ways to exploit it.

Towards the end of the above paper can be found the following:

> Finally, may I make a short plea that linear programmers pay greater attention to special methods for solving the larger matrices that are encountered in practice. The excellent work of Jacobs on the caterer problem and the work of Jacobs, Hoffman, Johnson on the production smoothing problem are examples of what may be done with certain dynamic models with a simple repetitive structure. Cooper and Charnes have employed in their work a number of short cuts that have permitted resolution of certain large scale systems. At RAND we have found efficient ways to hand compute generalized transportation problems, and Markowitz has proposed a general procedure in this area that is promising. Many models exhibit a block triangular structure and certain partitioning methods have been proposed which take advantage of this type of structure. There is need for those of you who are foresighted to do serious research in this area.
>
> At the present time (1955), it is possible to solve rapidly problems in the order of a hundred equations. The Orchard-Hays 701 Simplex Code has solved many problems of this size with as high as 1,500 unknowns and machine times of five to eight hours as a rule--all with excellent standards of accuracy. However, it is self-evident that no matter how much the general purpose codes are perfected they will be unable to cope with the next generation of problems which will be larger in size. It is also evident that the models currently being run could have been handled more effectively by the proposed special methods.
>
> There are certain characteristics common to many models which I believe should be emphasized:
> (1) Most factors in the coefficient matrix are zero.
> (2) In dynamic structures the coefficients are often the same from one time period to the next.
> (3) In dynamic solutions the activities employed often persist from one period to the next.
> (4) Transportation type submatrices are common.
> (5) Block triangular submatrices are common.
>
> Part of the research in this area should certainly be devoted to a better understanding of the potentialities of techniques other than the simplex method.

UNCERTAINTY

In a related paper [5], published in 1956, appears the following

> In the past few months there have been important developments that point to the *application of linear programming methods under uncertainty*. By way of background let us recall that there are in common use two essentially different types of scheduling applications--one designed for the short run and those

> for the long run. For the latter the effect of probabilistic or chance events is reduced to a minimum, by the usual technique of providing plenty of *fat* in the system. For example, *consumption rates, attrition rates, wear-out rates* are all planned on the high side. *Times to ship, time to travel, times to produce* are always made well above actual needs. Indeed, the entire system is put together with plenty of *slack* and *fat* with the hope that they will be the *shock absorbers* which will permit the general objectives and timing of the plan to be executed in spite of unforeseen events. In the general course of things, long-range plans are revised frequently because the stochastics elements of the problem have a nasty way of intruding. For this reason also the chief contribution, if any, of the long-range plan, is to effect an immediate decision--such as the appropiation of funds or the initiation of an important development contract.
>
> For short-run scheduling, many of the *slack* and *fat* techniques of its long-range brother are employed. The principle differences are attention to detail and the short time-horizon. As long as *capabilities* are well above *requirements* (or demands) or if the demands can be shifted in time, this approach presents no problems since it is feasible to implement the schedule in detail. However, where there are shortages, the projected plan based on such techniques may lead to actions far from optimal, whereas these new methods, where applicable, may result in considerable savings. I shall substantiate this later by reference to a problem of A. Ferguson on the routing of aircraft.
>
> With regard to the possibilities of solving large scale linear programming problems, one can sound both an optimistic and a pessimistic note. The pessimistic note concerns the ability of the problem formulator, either amateur or professional, to develop models that are large scale. The pessimistic note also concerns the inability of the problem sovler to compute models *by general techniques* when they are large scale. If this is so, is not the great promise that the linear programming approach will solve scheduling and long range planning problems with substantial savings to the organizations adopting these methods but an illusion and a snare? Are the big problems going to be solved as they have always been solved--by a detailed system of on-the-spot somewhat natural set of priorities that resolve every possible alternative as it arises?

The status of problems involving uncertainty as far as practical solutions are concerned, has not changed much since 1956. The following, sums up the 1965 situation:

> When one considers instead, a direct attack on uncertainty via mathematical programming, it inevitably leads to the consideration of *large-scale systems*. Problems with their structure, have proven difficult of solution so far. I believe that they will be the subject of intensive investigation in the future.

DECOMPOSITION PRINCIPLE

The Decomposition Principle [6] arose in 1958 in connection with a military tactical problem which was too large to handle by conventional linear programming problem. A good summary of the approach can be found in my 1965 survey article:

> Recently the author, jointly with Philip Wolfe, developed a new procedure that is particularly applicable to angular systems and multistage systems of the staircase type This is reported in preliminary form in RAND P-1544 (Nov.10, 1958) under the title, "A Decomposition Principle for Linear Programs". The system consists of certain goods shared in common among several parts and certain goods (including facilities, raw materials) peculiar to each part. In short the system is angular in structure.
>
> Although the entire procedure is one intended to be carried out internally in an electronic computer it may also be viewed as a *decentralized decision making process*. Each independent part initially offers a possible bill of goods (a vector of the *common* outputs and supporting inputs including outside costs) to a central coordinating agency. As a set these are mutually feasible with each other and the given common resources and demands from outside the system. The coordinator works out a system of "prices" for paying for each component of the vector plus a special subsidy for each part that just balances the cost.
>
> The management of each part then offers, based on these prices, a new feasible program for his part with lower cost *without regard to whether it is feasible for the system as a whole*. The coordinator, however, combines these new offers with the set of earlier offers so as to preserve mutual feasibility and consistency with exogeneous demand and supply and to minimize cost. Using the improved over-all solution he generates a revised set of prices, subsidies, and receives new offers. The essential idea is that old offers are never forgotten by the central agency (unless using "current" prices they are unprofitable); the former are mixed with the new offers to form new prices.

In the original paper [6] appears this abstract:

> A technique is presented for the decomposition of a linear program that permits the problem to be solved by alternate solutions of linear sub-programs representing its several parts and a coordinating program that is obtained from the parts by linear transformations. The coordinating program generates at each cycle new objective forms for each part, and each part generates in turn (from its optimal basic feasible solutions) new activities (columns) for the interconnecting program. Viewed as an instance of a 'generalized programming problem' whose columns are drawn freely from given convex sets, such a problem can be studied by an appropriate generalization of the duality theorem for linear

> programming, which permits a sharp distinction to be made between those constraints that pertain only to a part of the problem and those that connect its parts. This leads to a generalization of the Simplex Algorithm, for which the decomposition procedure becomes a special case.

The reported experience with solving structured linear programs by means of the decomposition principle varies from very good to poor, In general it appears that if the decomposition between master and sub is a "natural" one, it can perform very well. Like the simplex method, there is rapid improvement for the early iterations followed by a long tail except here the tail is much longer.

COMPACT BASIS INVERSES

From 1962 onwards there has been growing interest in schemes for compactly representing the inverse of the basis for the simplex method. This effort goes under various names: compact basis triangularization, LU basis factorization. One must worry not only about the compactness but also about the stability of the solution to small changes in the original data. My 1962 paper [7] was directed to finding a compact representation of a basis for staircase systems.

> Alex Orden was the first to point out that the inverse of the basis in the simplex method serves no function except as a means for obtaining the representation of the vector ertering the basis and for determining the new price vector. For this purpose one of the many forms of "substitute inverses" (such as the well known product form of the inverse) would do just as well and in fact may have certain advantages in computation.
>
> Harry Markowitz was interested in developing, for a sparse matrix, a substitute inverse with as few nonzero entries as possible. He suggested several ways to do this approximately. For example, the basis could be reduced to triangular form by successively selecting for pivot position that row and column whose product of nonzero entries (excluding the pivot) is minimum. He also pointed out that, for bases whose nonzeros appear in a band on a staircase about the diagnonal, proper selection of pivots could result in a compact substitute with no more nonzeros than the original basis.
>
> We shall adopt Markowitz's suggestion. However, instead of recording the successive transformations of one basis to the next in product form, we shall show that it is efficient to generate each substitute inverse in turn from its predecessor. The substitute inverse remains compact, of fixed size. Thus "reinversions" are unnecessary (except in so far as they are needed to restore loss of accuracy due to cumulative round-off error).
>
> The procedure which we shall give can be applied to a general m x m basis without special structure. As such, it is

> probably competitive with the standard product form, for it may have all of its advantages and none of its disadvantages. With certain matrix structures, moreover, it appears to be particularly attractive.
>
> We shall focus our remarks on *staircase structures*. The reader will find no difficulty in finding an equally efficient way to compact block-angular structures.

STATUS AS OF 1967

A summary of the status of solving large-acale problems can be found in my 1967 paper [8].

> From its very inception, it was envisioned that linear programming would be applied to very large, detailed models of economic and military systems. Kantorovitch's 1939 proposals, which were before the advent of the electronic computer, mentioned such possibilities. Linear programming evolved out of the U.S. Air Force interest in 1947 in finding optimal time-staged deployment plans in case of war; a problem whose mathematical structure is similar to that of finding an optimal growth pattern of a developing economy and similar to other control problems. Structurally the dynamic problems are characterized in discrete form by staircase matrices representing the inputs and outputs from one time period to the next. Treated as an ordinary linear program, the number of rows and columns grows in proportion to the number of time periods T and the computational effort grows by T^3 and possibly higher. This fact has limited the use of linear programming as a tool for planning over many time periods.
>
> At the present 1967 stage of the computer revolution, there is growing interest on the part of practical users of linear programming models to solve larger and larger systems. Such applications imply that eventually automated systems will obtain information from counters and sensing devices, process data into the proper form for optimization and finally implement the results by control devices. There has been steady progress in this mechanization of flow to and from the computer. Hitherto, this has been one of the obstacles encountered in setting-up and solving large-scale systems. The second obstacle has been the cost and the time required to successfully solve large problems.
>
> It is difficult to measure the potential of large-scale planning. Certain developing countries appear, according to optimal calculations on simplified models to be able to grow at the rate of 15% per year implying a doubling of their industrial base in five years. But administrators apparently ignore plans and make decisions based on political expediency which restrict growth to 2 or 3% or sometimes -2%. It is the belief of the author that the mechanization of data flow (at least in advanced countries) in the next decade will provide pathways for constructing

> large models and the effective implementation of the results of optimization. This application of mathematics to decision processes will eventually become as important as the classical applications to physics and will, in time, change the emphasis in pure mathematics.

In this paper the following unsolved problem was posed:

> It has been discovered recently that the size of the inverse representation of the basis in the simplex method could have an important effect on running time. Therefore, compact-inverse schemes along the lines first proposed by Harry Markovitz of RAND have become increasingly important. Recently, two groups working independently, developed this approach with astounding results. For example, the Standard Oil Company of California group reports running-time on some of their typical large problems cut to 1/4.
>
> How to find the most compact inverse representation of a sparse matrix is still an unsolved problem:
>
> CONJECTURE: *If a non-singular matrix has K non-zero elements, it is always possible to represent them as a product of elementary matrices such that the total number of non-zero entries (excluding their diagonal unit elements) is at most K.* [*Incidentally, the empirical schemes just mentioned often have no more than K+10%K non-zeros in the inverse representation.*]

STATUS TO THE PRESENT (1980)

From 1967 onwards there has been an increasing interest in techniques for solving large-scale linear programs. A number of conferences have been exclusively concerned with the topic. Most general operations research and management science meetings have at least one session devoted to it. A selected reference list which I use in my seminars (mostly published during the period 1970-78) contain 237 titles which I have arranged by sub area.

General Books (10 exclusively large scale, 2 sparse methods, 8 other)	20
Survey articles	12
GUB, G-GUB and the decomposition principle	15
Variants of above	19
Block Triangularity	3
Linear optimal control and dynamic systems	14
Nested decomposition	4
Column generation, convex and nonlinear programs	34
Sparse matrix techniques	10
Large networks and related problems	37
Applications	52
Software	17
Total	237

Some idea of the recent research of the Systems Optimization Laboratory of the Operations Research Department at Stanford can be gleaned from the titles that follow:

Andre Perold: "Fundamentals of a Continuous Time Simplex Method".

Andre Perold and George B. Dantzig: "A Basis Factorization Method for Block Triangular Linear Programs".

Bob Fourer: "Solving Staircase-structured Linear Programs by Adaptation of the Simplex Method".

Ron Davis: "New Jump Conditions for State Constrained Optimal Control Problems".

Philip Abrahamson and George B. Dantzig: "Imbedded Dual Decomposition Approach to Staircase Systems".

John Birge: "Solving Staircase Systems under Uncertainty".

This Workshop may well mark the point in time when efficient methods for solving large dynamic systems may be more than just a promise. Thirty three years from the time the hope was first expressed that such methods be found, they may soon become a reality!

REFERENCES

[1] Wood, Marshall K., and Dantzig, George B., Programming of Interdependent Activities, I, *Econometrica*, July-October 1949.

[2] Dantzig, George B., Programming of Interdependent Activities, II, *Econometrica*, July-October 1949.

[3] Wood, M.K., and Geister, M.A., Development of Dynamic Models for Program Planning, *Activity Analysis of Production and Allocation*, T. Koopmans, Ed., Wiley and Co. 1951.

[4] Dantzig, George B., Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming, *Econometrica*, 23, April 1955, pp 174-183.

[5] Dantzig, George B., Recent Advances in Linear Programming, *Management Science* Vol. 2, No. 2, January 1956, pp 131-144.

[6] Dantzig, George B., and Wolfe,Philip, Decomposition Principle for Linear Programs, *Operations Research* 8, Jan-Feb.1960, pp 101-111.
Another version of this paper appeared in *Econometrica* 29, Oct. 1960, pp 767-778.

[7] Dantzig, George B., Compact Basis Triangularization for the Simplex Method, *Recent Advances in Mathematical Programming*, Graves, R.L., and Wolfe, P., Eds., McGraw Hill, 1963, N.Y. pp 125-133.

[8] Dantzig, George B., Large-Scale Systems and the Computer Revolution, Proceedings of the Princeton Symposium on Mathematical Programming, H.W. Kuhn, Ed. Princeton University Press, Princeton N.J. Aug. 1967 pp 51-72.

THE SIMPLEX METHOD FOR NONSTRUCTURED LINEAR PROGRAMS

SOLVING LARGE SCALE LINEAR PROGRAMS WITHOUT STRUCTURE

P. Huard

Direction des Etudes et Recherches
Electricité de France

A variant of the simplex method is adapted for the solution of large-size linear programming problems with a very sparse constraint matrix. Instead of using the inverse of the basis, three sparse linear systems are directly solved at each step, using a suitable pivoting method. Two advantages of this variant compared to standard procedure are:

- Memory volume requirements are proportional to the number of constraints (and not to its square).
- Calculation may be faster; the appropriate numerical tests are described in the paper.

1. - INTRODUCTION

With regard to the resolution of large linear programs, the basis of a variant of the Simplex method, using only a small amount of memory, has already been briefly described [3].

The aim of the present paper is to give a detailed study of this method and of the numerical experiments that validate it.

In its classical form, the Simplex method uses a square matrix, the inverse of the basic matrix, whose value is updated at each iteration. The number of nonzero elements of this matrix increases rapidly as the iterations go along and it is necessary in practice, when using the explicit form of the inverse, to have on hand a number of memories equal to the square of its dimension, say m^2 for a linear program with m constraints. Thus it becomes difficult to handle problems having several hundred constraints, without using disks or tapes; then the overhead time may becomes prohibitive, because of their repetitive use and the large number of iterations.

Some special structures of the matrix of the linear program - like for example the block-angular one - allow for various interesting decompositions of the inverse of the basic matrix, which is similar to the solving of smaller linear programming problems. Then the amount of necessary memory varies only linearly with the size of the program, if the dimension of the blocks is a constant. Fortunately, such a block-angular structure is rather often encountered (dynamic problems, regionalization problems) and various decomposition methods have been proposed (see for example [5]).

However, many linear programs do not have any structures suitable for decomposition. This is the case for problems related to a graph - e.g. flow-problems - which contain the problem of electrical dispatching, as far as its structure is concerned.

Large linear programs, issued from "real life", have a very sparse matrix : only a few percent of the elements are nonzero. Of course, this sparsity appears in each basic matrix, but it disappears from the inverse matrix. The variant of the Simplex method, which follows, uses the basic matrix itself, instead of its inverse, and then eliminates the need for m^2 memory positions. However, in the calculations, products of a matrix by a vector are replaced by resolutions of linear systems of the same dimensionality. The complexity of these two operations would be of m^2 and m^3 order respectively, if the matrices were full, which would rule out the proposed variant. But, as will be seen below, two factors may make it competitive. One is the difference in sparsity between the basic matrix and its inverse. The other is the fact that generally, the basic matrix is almost triangular, or more precisely "triangular-band-wise". In other words : after having performed a suitable permutation of rows and columns, nonzero elements lie below an extra-diagonal line, located at a small distance p above the diagonal. Such a linear system is easily solved through a specialized pivoting method that we call below the method of parameters. The amount of calculations is proportional to $\rho\ p\ m^2$, where ρ is the proportion of nonzero elements, p the width of the band located above the diagonal, and m the dimension of the matrix (a large number, by hypothesis). In large problems, of real origin, that we have known of, p is often between $\rho\ m$ and $2\ \rho\ m$. If ρ' is the proportion of nonzero elements (density) of the inverse matrix (ρ' is normally much larger than ρ), the respective amounts of computation for one iteration of the Simplex method are roughly in the ratio $4(\rho/\rho')^2m$. For $\rho' = 60\ \rho$ and $m = 10^3$, this is practically 1. In actual fact, numerical comparisons of Section 7, involving linear programs of up to 900 constraints, exhibit a very good speed for the proposed variant. In Section 8 the detailed costs for one iteration of the Simplex method are given with a comparison between the two variants.

2. - THE REQUIRED CALCULATIONS DURING ONE ITERATION OF THE SIMPLEX METHOD

The linear program to be solved is given in standard form

$$\boxed{\begin{array}{c}\text{Maximise } f\,x \text{ subject to}\\[1em] A\,x = a \quad (1)\\[1em] x \geq 0 \quad (2)\end{array}}$$

where A is a full-rank matrix; its rows are indexed by $M = \{1, 2, \dots, m\}$ and its columns by $N = \{1, 2, \dots, n\}$.

At each iteration, a basis I is considered, i.e. a subset I such that :

$$\left|\begin{array}{l} I \subset N \\ |I| = m \\ A^I \text{ invertible}\end{array}\right.$$

where A^I, the basic matrix relative to I, is composed of the columns A^j, $\forall\, j \in I$.

To the basis I is associated the so-called basic solution of the basis I, defined by

$$x_I = (A^I)^{-1}\, a$$

$$x_{\bar{I}} = 0$$

where $\bar{I}$ is the complement of I in N.

The successive bases generated by the Simplex method, are such that $x_I \geq 0$; hence, the considered basic solutions are all feasible (they satisfy conditions (1) and (2)).

An iteration consists of changing the basis I into a neighboring basis I', that is a basis obtained by exchanging an index $r \in I$ with an index $s \in \bar{I}$:

$$I' = I - r + s \tag{3}$$

To determine r and s, one can compute, in order :

$$u = f^I \, (A^I)^{-1} \tag{4}$$

$$d^{\bar{I}} = f^{\bar{I}} - u\, A^{\bar{I}} \tag{5}$$

where u, f^I, $d^{\bar{I}}$ are row-vectors. This allows the candidate $s \in \bar{I}$, to be chosen with the condition $d^s > 0$. Then :

$$x_I = (A^I)^{-1} \, a \tag{6}$$

$$T^s = (A^I)^{-1} \, A^s \tag{7}$$

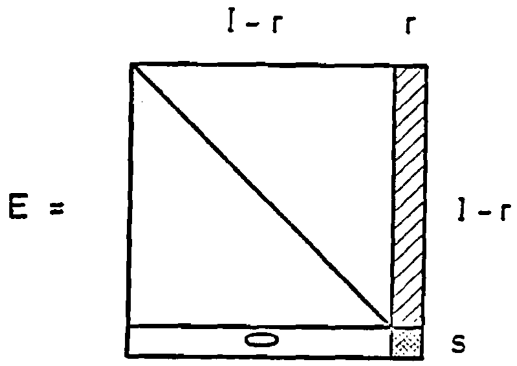
where x_I, a, T^s and A^s are column vectors. This gives $r \in I$ by the condition

$$\frac{x_r}{T^s_r} = \theta = \min \{\frac{x_i}{T^s_i} \mid i \in I,\ T^s_i > 0\} \tag{8}$$

Once r and s are determined, it remains to update the inverse of the basic matrix, i.e. to compute $(A^{I'})^{-1}$. This is classicaly done from $(A^I)^{-1}$ through the relation :

$$(A^{I'})^{-1} = E \, (A^I)^{-1} \tag{9}$$

where E is an elementary matrix, explicitly known (see figure 1).



$$E^{r}_{I-r} = -T^{s}_{I-r}\left(T^{s}_{r}\right)^{-1}$$

$$E^{r}_{s} = \left(T^{s}_{r}\right)^{-1}$$

Figure 1

Thus the necessary calculations are represented by relations (4) to (9), and the inverse of the basic matrix is used in (4), (6), (7). These last relations can be replaced by

$$u\,A^{I} = f^{I} \qquad (4')$$

$$A^{I}\,x_{I} = a \qquad (6')$$

$$A^{I}\,T^{s} = A^{s} \qquad (7')$$

i.e. three linear systems to solve. In the first one, the matrix is the transpose of the basic matrix, in the last two, it is the basic matrix itself : these systems enjoy the sparsity of the A matrix, and solving them can be done without storing and using the inverse.

3. - DIRECT RESOLUTION OF THE LINEAR SYSTEM

The systems (4'), (6'), (7') have long been successfully solved directly in the case of classical transportation problems. These very special linear programs can be stated :

$$\text{Minimize } \sum_{ij} c^{ij}\, x_{ij} \quad \text{subject to}$$

$$\sum_i x_{ij} = a_j \quad , \quad j = 1,2,\dots, p$$

$$\sum_j x_{ij} = b_i \quad , \quad i = 1,2,\dots, q$$

$$x_{ij} \geq 0 \quad , \quad \forall ij$$

Here the A matrix has no more than 2 nonzero elements per column, which are equal to 1, and the basic matrices are triangular. Thus solving the three linear systems is particularly easy and fast (it is not even necessary, here, to solve (7')).

An extension to problems of flow with gains was proposed by MAURRAS [4] in 1972. In this type of linear programs, the A matrix still has no more than 2 nonzero elements per column, but of any real value. Systems (4'), (6') or (7') are almost as simple as a triangular system. The method of solution consists of particularizing one unknown as a parameter, and in expressing one after the other the (m-1) remaining unknowns as functions of this parameter, using (m-1) equations. Eliminating these (m-1) unknowns from the last equation - not yet used - gives the value of the parameter. Plugging this value in the expression of the (m-1) unknowns completes the solution. The choice of the particularized unknown is guided by an interpretation of the structure of the A matrix, as incidence matrix of a graph. Of course, it is not possible to extend this theory to matrices with more than 2 nonzero elements per column. However, a study of many square matrices, very large and very sparse, issued from real problems, shows that they often have a triangular-band-wise structure (after suitable permutations of rows and columns); their band-width has the same order of magnitude as the average number of nonzero elements per column or per row. More precisely, these square matrices are such that

$$A_i^j = 0 \quad , \forall i, j : j > i + p$$

where p is the width of the band located above the diagonal. These matrices, of small thickness, correspond to linear systems that are easily solved by the pivoting method, called method of parameters, described in the next section. This method, which can be considered as an extension of that used by MAURRAS, uses a number of parameters equal to p. In practice, it reduces to solving a triangular system of dimension (m-p) with p right-hand sides, and solving a p × p system. In problems of flows with gains, one always has $p \leq 1$.

4. - THE METHOD OF PARAMETERS

Let the system to solve be

$$B\, x = b \tag{10}$$

where B is an invertible (m × m) matrix, such that

$$B_i^j = 0 \quad \forall i, j = 1,2,\ldots m : j > i + p \tag{11}$$

We call p the band-width of the triangular-band-wise matrix B.

The row i = 1 has at most p + 1 nonzero elements. We may suppose $B_1^{p+1} \neq 0$, possibly after having exchanged column p + 1 with some other. Therefore we can express x_{p+1} as a function of the variables x_j, $j = 1,2,\ldots,p$ considered as parameters :

$$x_{p+1} = \ell_{p+1}(x_1, x_2, \ldots, x_p) \tag{12}$$

where ℓ_{p+1} is an affine function.

If $B_2^{p+2} \neq 0$ we can express from the row i = 2, x_{p+2} as a function of x_{p+1} and of the parameters $x_1,\ldots, x_p$. Eliminating x_{p+1} with (12) we obtain

$$x_{p+2} = \ell_{p+2}\ (x_1,\ x_2,\ldots,\ x_p) \tag{13}$$

and so on. If, at each step k, corresponding to the use of the row k, we have $B_k^{k+p} \neq 0$, we obtain after (m-p) steps, the affine functions

$$\ell_{p+i}\ (x_1,\ x_2,\ldots,\ x_p) \quad i = 1,2,\ldots\ (m-p) \tag{14}$$

Storing the coefficients of these functions (including the affine terms) requires an array (m-p) × (p+1).

Only the first (m-p) equations have been used. Using (14) we can eliminate the variables x_{p+i} i = 1,2,..., m-p from the p remaining equations, and we obtain a system of p equations, where the p unknowns are the parameters $x_1,\ldots,\ x_p$. Solving this (p × p) system gives the values of the parameters, and then (14) gives the other unknowns.

The hypothesis $B_k^{k+p} \neq 0$ implies that a new unknown x_k does appear at step k. If this hypothesis is not satisfied, then the unknown x_k does not appear yet (nor any other, because of (11)); one parameter can be eliminated between equations k and k-1, which no longer contain the unknowns x_{p+i}, i = 1,2,..., (k-1), after use of (14). From then on, this eliminated parameter will become an unknown, expressed as a function of the remaining parameters. But later on, more than one unknown may appear at some step k' > k. It is then necessary to introduce new parameters, consisting of the excess unknowns.

Thus the set of parameters may fluctuate along the steps, in its dimensionality as well as in its content - see Figure 2. But it is sure, from (11), that it has never more than p elements.

In addition to the matrix B and the right-hand side b, the core requirement is at most m × (p+1) : (m-p) × (p+1) memories for the expressions (14), and p × (p+1) for the (p × p) system. Hence, in order to reduce the required storage, it is convenient to reduce the bandwidth p down to a value as small as possible, by means of suitable

$$\begin{array}{c|cccccccc|c|}
 & 1 & 2 & p+1 & & & 6 & & m & \\
1 & * & * & * & & & & & & * \\
2 & * & * & * & * & & & & & * \\
3 & * & * & * & * & * & & & & * \\
4 & * & * & * & * & * & & & & * \\
 & * & * & * & * & * & * & * & & * \\
m-p & * & * & * & * & * & * & * & * & * \\
 & * & * & * & * & * & * & * & * & * \\
m & * & * & * & * & * & * & * & * & *
\end{array}$$

Row	Parameters used
1	(1,2)
2	(1,2)
3	(1,2)
4	(1)
⋮	(1,6)
m-p	(1,6)
⋮	
m	

Figure 2

rearrangements of the matrix B. Various techniques, systematically tested by D. FAYARD and G. PLATEAU [1], and Y. HAUW [2], have led to a simple technique, described in the next section; it gives a band-width which, if not optimal, is a quite satisfactory approximation.

For the sake of theoretical curiosity, as has been pointed out in [2], when applied to (10) with a full matrix B, the method of parameters leads to a pivoting method of the diagonalization type, as with the Jordan method. But the operations are not the same, and a precise inventory of the calculations shows that the respective numbers of multiplications, divisions and additions, are exactly the same as in Gauss method (which is a triangularization method, cheaper than Jordan's). In section 9 a detailed comparison of these operations will be given.

Finally, it should be noted that for the steps not including eliminations the pivots used are original elements of the B matrix, for at step k, rcws k+1 to m have not yet been modified. This fact is important for the stability of the computations.

5. - OBTAINING IN PRACTICE THE MINIMAL BAND-WIDTH

To permute rows and columns of the B-matrix reduces to choose two permutation functions g and h, defined on the domain $M = \{1,2,\dots, m\}$. The optimal permutations, which give minimal band-width, solve the problem

$$\min_{g,h} \{\max \{(h(j) - g(i)) \ \beta_{ij} \mid i, j \in M\}\}$$

where $\beta_{ij} = 1$ if $B_i^j \neq 0$, $\beta_{ij} = 0$ otherwise.

No exact solution is known to this combinatorial problem, except through exhaustive enumeration - too expensive. Various heuristic approaches have been proposed, to solve this problem or similar ones. In the case of band-matrices of minimal band-width, we mention the process of Tewarson [7], which requires the resolution of an integer programming problem, without even guaranteeing an optimal solution.

In fact, concerning large and very sparse matrices, issued from real problems, some simple heuristics, based on intuitive considerations, have proved very efficient in a large number of cases. Rule 5.2 below is one of them.

5.1. - The full-rectangles rule

The nonzero elements of the B-matrix are squared into a string of rectangles, which touch one another by their diagonal corners, and whose upper-right elements are nonzero (see Figure 3).

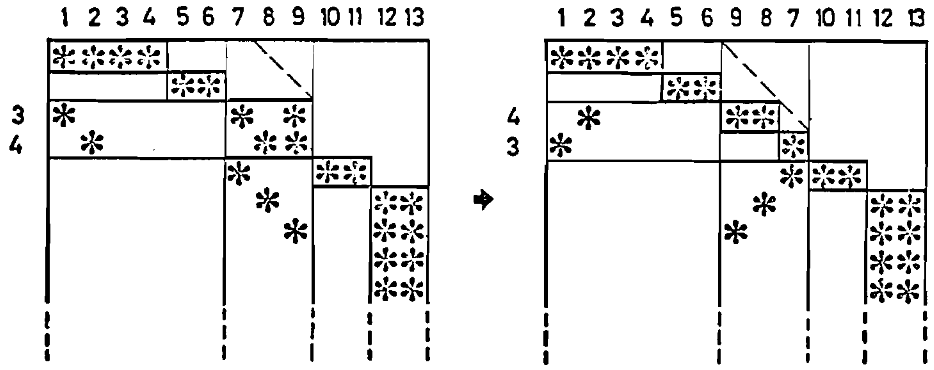


Figure 3

Any one of these which is not full can always be decomposed into smaller full rectangles, and this is only done by permutations which concern only rows and columns in that rectangle. Then the new band-width is not greater than the old band-width.

This process can be applied independently to every initial rectangle that is not full. But, as Figure 4 shows, it does not guarantee an optimal solution.

5.2. - Row of smallest relative degree

Let Γ_i be the set of indices corresponding to nonzero elements in the row i. Having fixed the first k rows of B, the number

$$d_j = |P_j| - 1$$

where $$P_j = \Gamma_j / \Gamma_j \cap (\bigcup_{i=1}^{k} \Gamma_i)$$

is called the degree of the row j, relative to the first k rows. P_j represents the set indexing the elements that are nonzero in the row j, but are zero in the first k rows. Thus, adding a row of 0 degree after the first k rows does not increase the number of parameters. A negative degree will decrease by 1 the number of parameters (through elimination). A positive degree increases that number by d_j.

Therefore a simple process consists of sorting the rows downwards : at each stage, one chooses a row that has the smallest relative degree among the remaining ones, and the new columns are moved so that the nonzero entries in the present new row are regrouped on the left.

It is this simple process that has finally been implemented in the code written by HAUW [2], after a number of extensive tests with matrices (20 × 20) and (100 × 100) have been performed. It seems that, with (100 × 100) matrices, the band-width has always been optimized within 2 or 3.

When several rows have the same relative degree at the same stage, it is attractive to use a secondary criterion to choose from among them. For example, their influence on the remaining rows may be considered. After having tried more than a dozen such criteria, none has proved

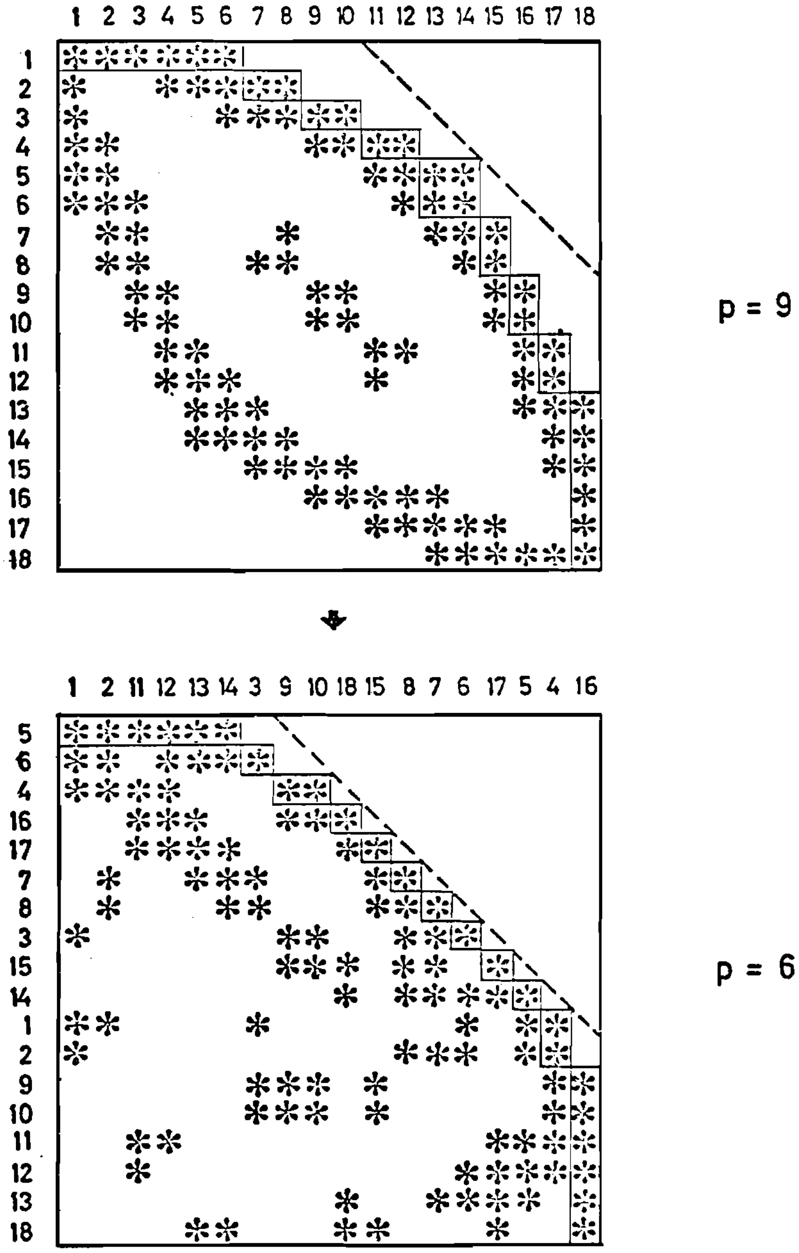

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
p = 9
1 2 11 12 13 14 3 9 10 18 15 8 7 6 17 5 4 16
5
6
4
16
17
7
8
3
15
14
1
2
9
10
11
12
13
18
p = 6

Figure 4

significant. Finally, the policy is to take the last encountered of the candidate rows (which leads to the easiest implementation).

It can be checked that this process automatically satisfies the rule 5.1 of full rectangles.

5.3. - Taking into account of special structures

Obvious permutations can be suggested by certain special structures. This is the case for example when slack variables are present (or, more generally, when the matrix A contains a diagonal submatrix).

It is straight forward to obtain a basic matrix A^I that has the pattern indicated on Figure 5 (where U is a unit matrix, corresponding to the slack variables in the basis). In practice, the slack-rows are placed in the bottom. Then only the B-matrix is processed, and its column permutations are also applied to C. It is the new triangular-band-wise matrix B' that imposes the number of parameters.

Note also that, when the basis is changed, the triangular-band-wise pattern of the basic matrix is only slightly affected. It can easily be seen that, through a very simple column permutation, the band-width is changed by 1, 0 or -1. Thus, a complete reor dering may be applied only from time to time.

6. - AVERAGE THICKNESS OF A MATRIX

An important question, before using the method of parameters, is to know what band-width is to be expected after reordering.

Or course, this question has no general answer, but one can try to have an idea by studying first the probability distribution of this band-width, for matrices whose elements are randomly generated. This is done in 6.1. Structured matrices are studied in the following sections :

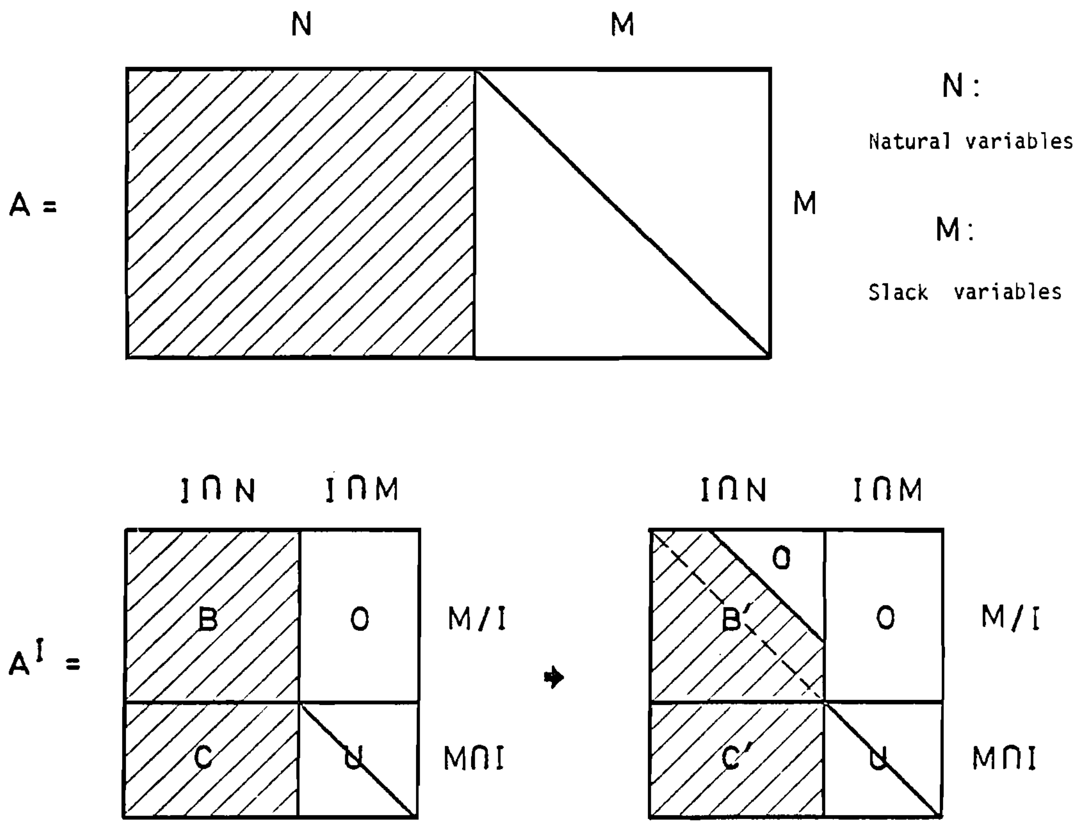

N
M
A =
M
N:
Natural variables
M:
Slack variables
I∩N
I∩M
A^I =
B
O
M/I
C
U
M∩I
I∩N
I∩M
O
B'
O
M/I
C'
U
M∩I

Figure 5

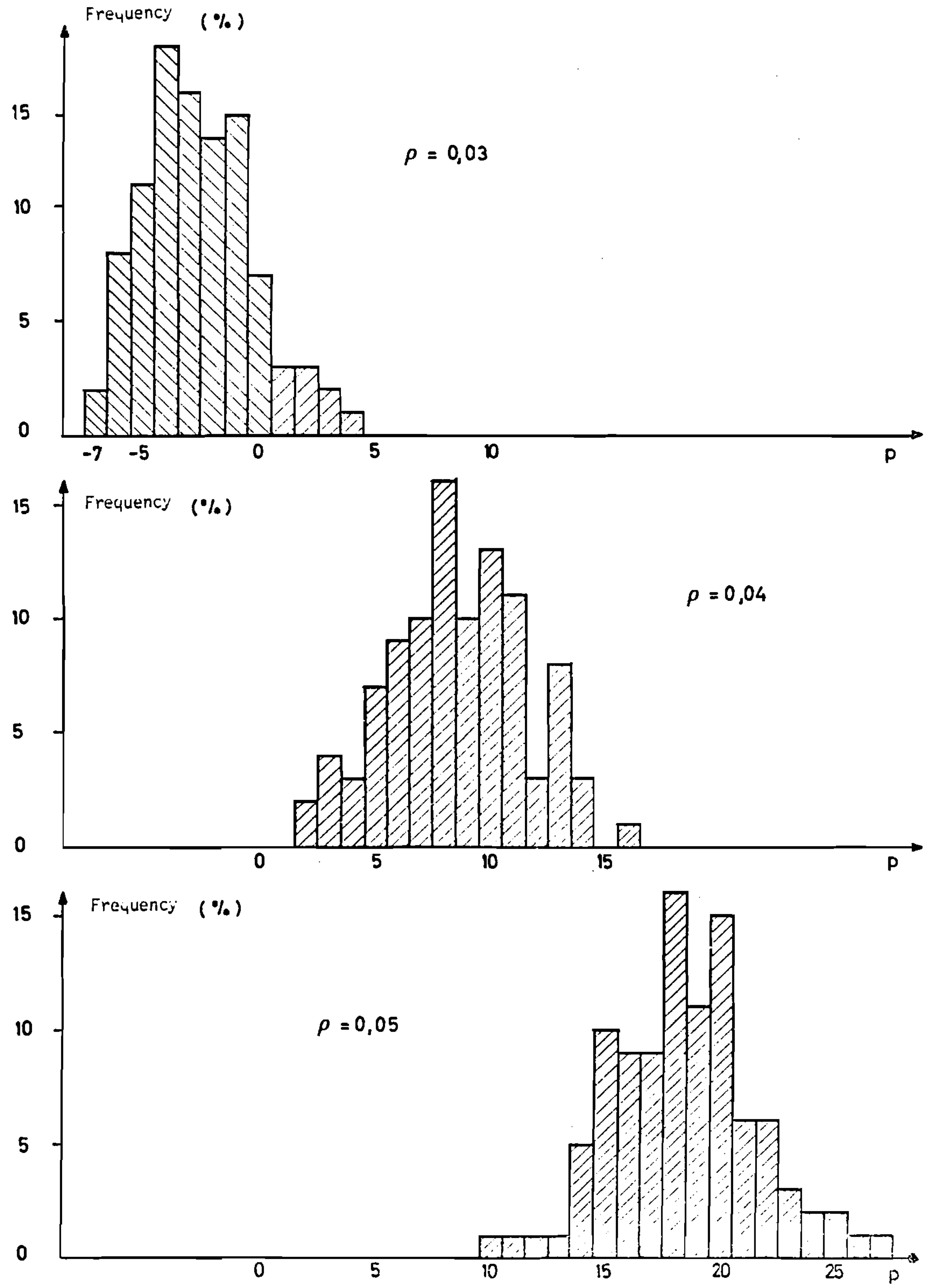

Frequency (%)
15
10
5
0
-7
-5
0
5
10
p
ρ = 0,03
Frequency (%)
15
10
5
0
0
5
10
15
p
ρ = 0,04
Frequency (%)
15
10
5
0
0
5
10
15
20
25
p
ρ = 0,05

Figure 6

in 6.2, pathological cases - fortunately artificial and rare - that give maximum band-width; in 6.3, highly structured matrices, issued from problems of electrical dispatching, always giving small band-widthes.

6.1. - Sparse matrices randomly generated

Three samples, of 100 matrices each, have been generated. These matrices are (100 × 100) and their elements, 0 or 1, are realizations of independent random variables with a probability ρ to get a 1. The samples correspond respectively to $\rho = 0.05, 0.04, 0.03$. Each matrix thus obtained is processed as described in 5.2, so as to obtain a band-width as small as possible.

Figure 6 indicates the frequency of the minimal band-width p. Note the dispersion of p, and its very quick variation, as a function of ρ: for $\rho = 5\%$, the average p is between 18 and 19, but reduces to 8-9 for $\rho = 4\%$, and practically vanishes for $\rho = 3\%$.

However, the density is not the only influential factor for the thickness of the matrix. From a remark of W. DE LA VEGA and J.F. MAURRAS [9] a randomly generated (1,000 × 1,000) matrix of exactly 10 nonzero elements per row (and hence with a density of 0.01) may have a null (333 × 333) submatrix with a probability almost equal to zero.(*) The "absolute" value of $|\Gamma_i|$ seems to play an important role.

Lastly, notice the numerical experiments of J. DENEL [8] concerning random matrices with, for each row i, a randomly generated value of the degree $|\Gamma_i|$ between 1 and d, and randomly generated ranks for the nonzero elements. The sizes of these matrices vary between 50 and 1,000, with d = 6, 10 and 20. The mean value of the degrees is thus d/2. Notice that almost all these matrices are structuraly singular. In the table below are given the mean values of ρ and p for each couple (d, m), corresponding to samples of 10 matrices ($m < 1,000$) or 20 matrices ($m = 1,000$).

d	6				10							20
m	50	100	200	1,000	50	100	200	400	600	800	1,000	1,000
ρ%	7	3.5	1.75	0.35	11	5.5	2.75	1.4	0.9	0.7	0.5	1
p	3	3.5	5	19	10	19	34	67	88	121	155	330

It is not really possible to draw practical conclusions from these experiments, because basic matrices of usual linear programs substantially deviate from these random matrices.

6.2. - Pathological cases

It is possible to construct matrices in which any pair of rows (and of columns as well) have only one nonzero element at the same place, i.e. :

$$|r_i \cap r_j| = 1, \qquad \forall\, i, j = 1, 2, \dots, m, \qquad i \neq j$$

(*) This theorical result was confirmed by numerical experiments : among twenty such random (1,000 X 1,000) matrices, the minimum value of p was 431 (See [8]).

Such $(m \times m)$ matrices, with k nonzero elements per row end per column, can be found by representing configurations or finite projective planes. A study is given in [6].

These matrices are characterized by the numbers m and k related by

$$k = q + 1$$

$$m = q^2 + q + 1$$

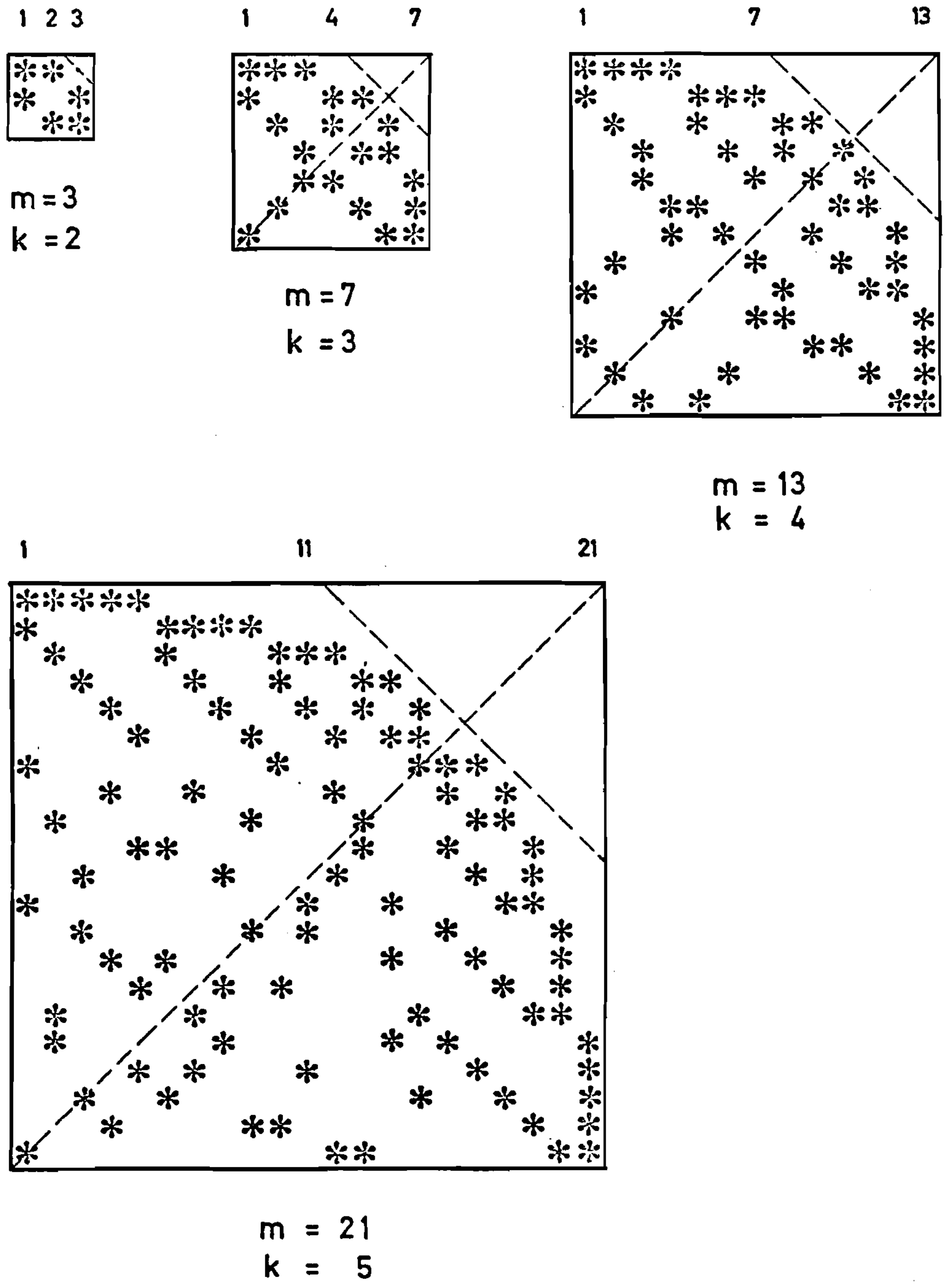
where q is a prime number, or a power of a prime number. One sees that the density $p = k/m$ becomes small when the size of the matrix increases.

Examples of such matrices are given in Figure 7 for $q = 1, 2, 3, 2^2$ We leave to the reader the pleasure to construct the case $q = 5$ $(m = 31, k = 6)$. He will then see that constructing such matrices is not a trivial task. It is fortunate that these matrices are some-what "rare", because it is easy to check that their minimal band-width is at least $k(k-1)/2$, or $(m-1)/2$, i.e. the same order of magnitude as m.

6.3. - Matrices of real motivation

Contrary to random matrices, matrices corresponding to linear programs coming from real problems, are highly structured. As a result, for the same proportion of nonzero elements, they have narrower bands.

Experiments with problems of electrical dispatching, have been conducted by FAYARD, HAUW and PLATEAU [1], [2]. A first series of 12 (20×20) matrices - issued from linear programs representing the CIGRE model of electrical network with 10 nodes - having many nonzero elements (20% to 35%) have given band-widthes ranking from 1 to 5, as indicated in the table below. The indicated p-values are the smallest ones obtained after various trials of permutations. But results were generally obtained with Procedure 5.2.



m = 3
k = 2

m = 7
k = 3

m = 13
k = 4

m = 21
k = 5

Figure 7

ρ%	19,75	23	25,25	27	27,75	28,75	30	31,25	32,5	33,75	34,5	35,25
P	1	2	2	3	4	4	3	3	3	4	4	5

Note that p generally increases with ρ, but with fluctuations, of course due to differences in the structures.

A second series of experiments has been conducted with 10 (100 × 100) matrices A of the following form :

$$A = \left[\begin{array}{c|c|c} \begin{matrix}1\\0\\0\end{matrix} & B' & B \\ \hline \begin{matrix}0\\0\\0\end{matrix} & B' & B \end{array}\right]$$

where B' is obtained from the given matrix B by removing one column. This general pattern is typical in problems of electrical dispatching, having similar constraints on reactive and active powers.

The chosen B matrices had structures frequently encountered in this type of problems, with ρ-values ranking from 6% to 10%. As indicated in the table below, the corresponding p-values vary from 5 to 20; they are the smallest values obtained after various trials of permutations.

ρ%	5,93	5,95	5,95	5,97	6,49	7,35	8,85	8,93	9,15	9,49
p	7	5	8	6	8	12	17	20	18	15

Here again the rough increase of p with ρ is patent, despite the variety of the chosen B-structures. Just for comparison with the random (100 × 100) matrices of Section 6.1, a smoothed extrapolation of the above results give approximately p = 3 for ρ = 5% (compare with Figure 6)

7. - NUMERICAL EXPERIMENTATION OF THE METHOD OF PARAMETERS [2]

The method of parameters has been experimented with some linear programs, the dimension of which ranking from 60 to 922 constraints. The results, displayed in the table below, show that the maximum number of parameters used during each resolution is always considerably smaller than the number of constraints; this established the interest of the method of parameters with respect to explicit use of the inverse of the basic matrix, as far as storage is concerned.

The very simple experimental code used in the tests was written by Y. HAUW [2]. It contains a switch for the computation d, T^S and x_I (see Section 2) either by "Explicit inverse" or by "Parameters". The two variants for a same problem normally give the same sequence of bases, except possibly by the end, in the case of very small values for d^s and of roundoff errors different in either method. This code is written in Fortran IV, H compiler.

The computation times indicated in the "Parameters" column are those obtained with this code on a CII IRIS 80 computer. Likewise for the "Explicit inverse" column for problems 1,2,3. For problem 4, it is the time obtained with the IMSL Code on IBM 3033 Computer, multiplied by 19.5 in order to compare with IRIS 80(*).

For problem 5, it is the time obtained with the APEX III code on CDC 7,700 (although this code factorizes the basic inverse) multiplied by 60. It may be remarked that the parameter's variant is quite competitive for the first four problems. For problem 5, the analysis of the computation time has shown that the re-ordering of the matrix needed almost all of the 11 seconds. The sorting routine used in the experimental code of Y. HAUW was a m^2 sequential sorting routine, which becomes prohibitive for large values of m. A new version of J. DENEL [8], based on an adaptation of the binary-tree HEAP-SORT procedure, has a cost of only N $\log_2 m$, N being the number of nonzero elements of the matrix. The time for ordering a (1,000 × 1,000) matrix is then divided by about 10, which for problem 5 should give a time per iteration similar to the one of the APEX III code.

(*) The number 19.5 is obtained by comparing the times needed to invert a matrix in double precision (COLVILLE standard program). These times are respectively 2.51 and 49.1 seconds.

Notice finally that the possibilities of saving permutations when the change of basis takes place, indicated at the end of section 5.3., have not yet been used in the code.

Problem	m	n	Nonzero elements		p	Mean time/iteration in sec.	
N°			Total	%	maxi	Explicit inverse	Parameters
1	60	92	328	6	6	0.13	0,17
2	100	120	600	5	8	0.45	0.41
3	170	218	793	2	6	1.36	0.75
4	249	487	954	0,8	4	4.60	1,14
5	922	1763	3738	0,23	3	1	11

Some notes about the origins of the problems considered

The dimensions are those of the standard form (equality constraints, non negative variables, slacks included, artificial variables excluded).

N° 1 : A program with 28 inequality constraints and 32 natural variables, non negative and upper-bounded. The bounds are taken as ordinary constraints, hence 28 + 32 = 60 slack variables, and 28 + 32 = 60 constraints.

N° 2 : A synthetic problem, the matrix being obtained by doubling a random (99 × 60) matrix - having linearly independent columns - and adding a bordering line, as described in CHARNES, RAIKE, STUTZ and WALTERS (ACM volume 17 number 10 (1974) 583-586).

N° 3 : Management of a reservoir, 122 inequality constraints and 48 natural variables, non negative and bounded. The bounds are treated as ordinary constraints, hence 122 + 48 = 170 slack variables, and 122 + 48 = 170 constraints.

N° 4 : Energy program, with two periods, 236 inequalities and 13 equalities, 25i natural positive variables. Hence 236 + 13 = 249 constraints and 236 slack variables.

N° 5 : Energy program over 8 periods, with 922 inequalities and 841 natural positive variables, hence 922 slack variables.

8. - ANNEX 1

COMPARISON OF THE REQUIRED CALCULATION, DURING ONE ITERATION OF THE SIMPLEX METHOD, BETWEEN THE EXPLICIT USE OF THE INVERSE, AND THE DIRECT RESOLUTION WITH PARAMETERS

In one iteration of the Simplex method, the matrix calculations that differ in the two variants are : (4), (6), (7), (9) for the explicit use of the inverse, and (4'), (6'), (7') for the solving of the linear systems with the method of parameters.

In the first variant, it is of course possible to avoid (4) by using the classical relation

$$u(I') = u - d^{s}(T^{s}_{r})^{-1} (A^{I})^{-1}_{r} \qquad \text{(A-1)}$$

where the values of u, d, T are those relative to the basis I, and hence are known. One can also compute directly

$$d(I') = d - d^{s}(T^{s}_{r})^{-1} (A^{I})^{-1}_{r} A \qquad \text{(A-2)}$$

Also, in either variant, it is possible to avoid (6) or (6'), using the classical relation :

$$\left.\begin{array}{ll} x_i(I') = x_i - T^{s}_{i}\ \theta & , i \in I \\ x_s(I') = \theta & \\ x_j(I') = 0 & , j \in \bar{I} - s \end{array}\right\} \qquad \text{(A-3)}$$

where θ is given in (8).

However, in large problems, with many iterations.roundoff errors may become important in these recursive calculations. In what follows, therefore, we suppose that both variants actually use (4), (6), (7) and (9), on the one hand, and (4'), (6'), (7') on the other.

The second variant (direct resolution) requires in addition rearranging rows and columns (actually : rearrangement of pointers), i.e. operations that can hardly be compared with arithmetic operations. Nevertheless, these operations being fast, we will disregard them in the analysis below.

Some more comments before going on : operations (6) and (7) cost the same. Operations (4'), (6') and (7') as well, but (6') and (7') concern the resolution of the same linear system with two different right-hand sides, which is little more expensive than just one resolution. Solving (4') corresponds to the transpose matrix, which enjoys the same reordering as the basic matrix (rows are just used in reverse order).

Therefore it suffices to detail the calculations for (6) on the one hand and for (6') with one and two right-hand sides. These calculations mainly consist of scalar products between rows and columns, so we take into account the zero-elements of these vectors to avoid corresponding multiplications : the amount of calculation is the expectation of the actual number of multiplications. The value of this number in a scalar product exploiting sparsity, is recalled in Section 8.1. Also, operations whose result is a value known in advance (0 or 1) will not be counted.

We recall that these schematic balances count only the arithmetic operations : multiplication, addition, division, that they analyse only parts that differ in the two variants and that they do not take into account possible computer adaptations, characteristic of each variant.

8.1. - Scalar product of two sparse vectors

Let u and v be two m-vectors, the components of which are independent random variables. Let p (resp. p') be the probability that a component of u (resp. v) is zero. We set $q = 1 - p$, $q' = 1 - p'$. Consider the scalar product

$$u \cdot v = \sum_{i=1}^{m} u_i \, v_i$$

If x is the number of multiplications with one zero at least, this number equals the total number of zeroes in u and v, minus the number of coïncidences $u_i = v_i = 0$. Hence :

$$E(x) = m(p + p' - pp')$$

$$E(x) = m(1 - qq') \qquad (A.4)$$

If y is the number of actual multiplications(u_i and $v_i \neq 0$), one has $y = m - x$, hence

$$E(y) = m\,qq' \qquad (A.5)$$

8.2. - Detailed calculations in the variant "explicit inverse"

8.2.1. - Updating the inverse

The new inverse is obtained by premultiplying $(A^I)^{-1}$ by an elementary matrix E. Thus, an element (i,j) of the new inverse $(A^{I'})^{-1}$ is calculated through the following scheme (see Figure 8).

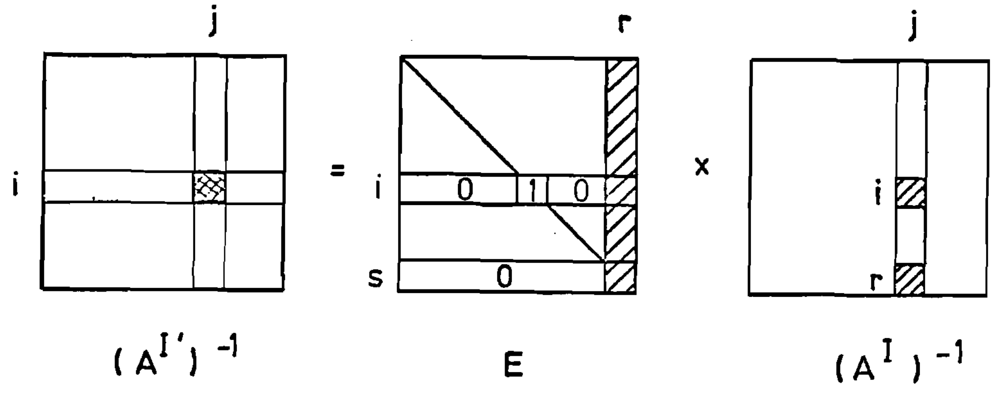


Figure 8

If $i \neq s$, i.e. $m(m-1)$ occurences : 1 addition and 1 multiplication.

If $i = s$, i.e. m occurences : only 1 multiplication.

The addition is done only if the element (i,j) of the old inverse is nonzero. The multiplication is done only if the element(i,r) of E and the element (r,j) of the old inverse are both nonzero.

If ρ' is the density of the basic inverse, and of the r-column of E (which is obtained, from the candidate column T^s, through m divisions), we finally obtain the following account :

×	$m((m-1)\rho' + 1) \rho'$
+	$m(m-1) \rho'$
÷	$(m-1) \rho' + 1$

(A.6)

8.2.2. - Product of the basic inverse by a vector

If ρ is the density of vectors f^I, a or A^s (supposed to be equal to that of A) we obtain

×	$m^2 \rho\rho'$
+	$m^2 \rho\rho'$
÷	0

(A.7)

8.2.3. - Total account

Summing up operations 8.2.1 and 8.2.2 (the latter being done three times) gives a total account :

			m large
×	$m(m(3\rho + \rho') + 1 - \rho')\rho'$		$m^2(3\rho + \rho')\rho'$
+	$m(m(1 + 3\rho) - 1)\rho'$		$m^2(1 + 3\rho)\rho'$
÷	$(m - 1)\rho' + 1$		$m\ \rho'$

(A.8)

8.3. - Detailed calculations in the variant "Parameters"

We study here the direct resolution of a linear system, considering simultaneously h right-hand sides. We have h = 1 when solving (4'), and h = 2 when solving simultaneously (6') and (7').

The (m × m) matrix of the system is supposed triangular-band-wise, having a band of width p (above the diagonal, diagonal excluded). Therefore the number of parameters to be used when solving this system is at most p. We will further suppose that this number is constantly equal to p (no temporary elimination of parameters).

There are four distinct phases in the calculation :

- Successive transformations of the first (m - p) lines, to express (m - p) unknowns as functions of the parameters.

- Construction of the (p × p) system to compute the parameters.

- Solving this system.

- Calculating the (m - p) other unknowns.

It is reasonable on the long run to take for the basic matrix the same proportion ρ as for the matrix A. However we cannot take the same value for the row-sections that lie below the null-triangle. We have to modify ρ according to the ratio of surfaces of the null-triangle and the matrix, and to take

$$\rho'' = \frac{\rho}{1 - \frac{(m - p - 1)(m - p)}{2 m^2}} \qquad (A.9)$$

If m is large with respect to p, one has approximately $\rho'' = 2$.

8.3.1. - Transforming line k (k = 1, 2, ..., m-p)

The first (k-1) rows (including the right-hand side(s)) have already been transformed by pivoting, and look like the sketch on the left of Figure 9 (where only one right-hand side is shown).

We suppose in this figure that the p parameters correspond to the p first columns, and that no parameter has been eliminated. We suppose that, from the previous operations, the first p columns are full, as well as right-hand sides. The other elements in the first (k-1) first rows are : 1 in (i, p + i) and 0 elsewhere.

The operations that transform the row k are :

- $\rho''(p + k + h - 1)$ divisions by the pivot (divisions of the non-zero elements, excluding the pivot but including the right-hand sides).

- $\rho''(p + h)(k - 1)$ multiplications (multiplying each row k' < k, by the same element, to obtain after addition a zero at the location (k, p + k')). Only the elements of the first p columns, as well as right-hand sides, are actually multiplied.

- $\rho''(p + h)(k - 1)$ additions (to each multiplication above, corresponds one addition to some element in the row k).

Summing up from k = 1 to k = m - p, we obtain :

×	$\frac{(m - p)(m - p - 1)(p + h)}{2} \rho''$
+	$\frac{(m - p)(m - p - 1)(p + h)}{2} \rho''$
÷	$\frac{(m - p)(m + p + 2h - 1)}{2} \rho''$

(A.10)

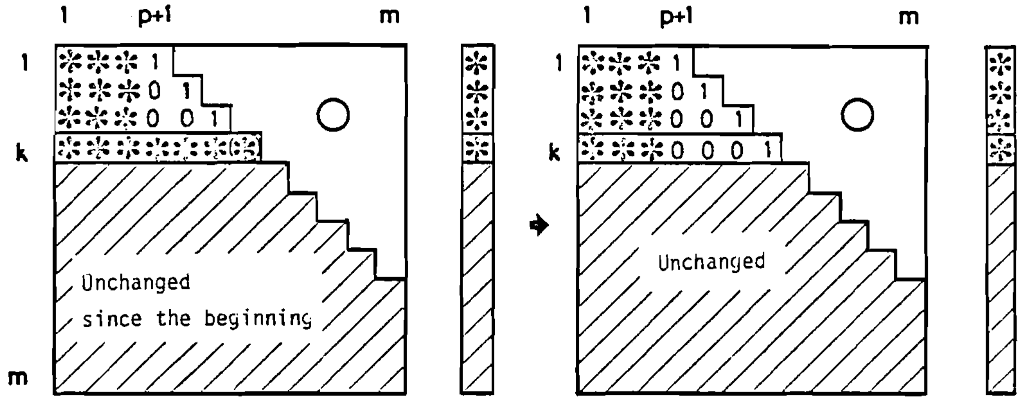


Figure 9

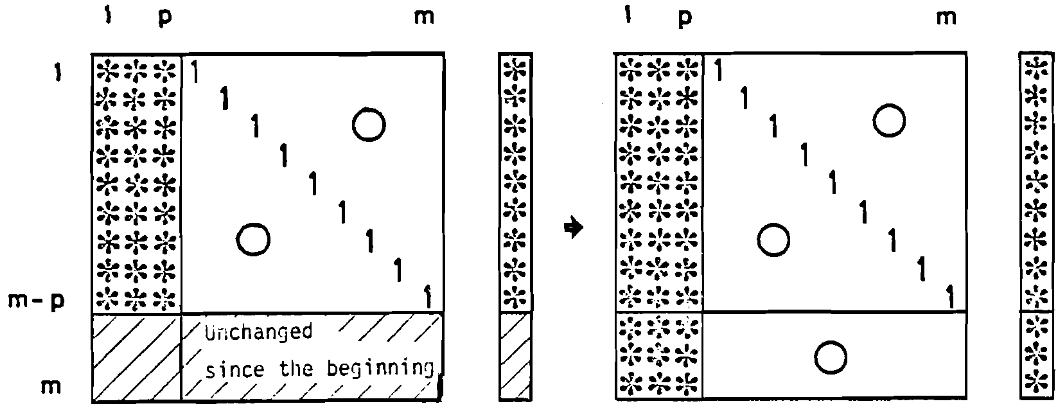


Figure 10

8.3.2. - Building the $(p \times p)$ system

Once the above operations have reached the row $k = m - p$, we have a matrix looking like the sketch on the left of Figure 10. Combining with the first $(m - p)$ rows, we eliminate the entries of the last p rows, columns $p + 1$ to m. If some entry is already 0, the operation is skipped.

We suppose, that the submatrix (i,j) $i = 1, 2, \ldots (m-p)$, $j = 1, 2, \ldots, p$ is full, but only the proportion ρ'' is to be considered in the submatrix (i,j), $i = (m - p + 1), \ldots, m$, $j = (p+1), \ldots, m$ because the transformations 8.3.1. have not affected the last p rows. Moreover, eliminating an entry of this submatrix does not change its other entries (yet it modifies the corresponding row of the $(p \times p)$ submatrix of the parameters).

We finally obtain after enumeration :

×	$p(p + h)(m - p)\ \rho''$
+	$p(p + h)(m - p)\ \rho''$
÷	0

(A.11)

8.3.3. - Solving the $(p \times p)$ system for the parameters

Its matrix is normally full. A classical pivoting method such as GAUSS's method requires :

×	$\frac{p(p - 1)(2p + 3h + 2)}{6}$
+	$\frac{p(p - 1)(2p + 6h - 1)}{6}$
÷	$\frac{p(p + 2h - 1)}{2}$

(A.12)

which amounts to p^3 order, and is negligible if p is small with respect to m.

8.3.4. - Calculating the other unknowns

To obtain the k-th unknown (k = 1,2,..., m-p) one has to multiply the p first entries of the k-th row (matrix on the right in Figure 10) by the corresponding parameter value and to substract the results from each right-hand side. Hence, for the whole of (m-p) unknowns and h right-hand sides :

×	$(m - p)\rho\, p$
+	$(m - p)\ p\ h$
÷	0

(A.13)

8.3.5. - Total account

In summary, solving (4'), (6'), (7') as a result of adding (A.10) to (A.13) for h = 1 and h = 2, requires the following operations :

×	$\frac{(m - p)(m + p - 1)(2\ p + 3)}{2}\ \rho'' + 2(m - p)p + \frac{p(p - 1)(4\ p + 13)}{6}$
+	$\frac{(m - p)(m + p - 1)(2\ p + 3)}{2}\ \rho'' + 3(m - p)p + 2\frac{p(p - 1)(p + 4)}{3}$
÷	$(m - p)(m - p + 2)\ \rho'' + p(p + 2)$

(A.14)

If m is large in front of p, $\rho'' \sim 2\rho$ and the orders of magnitude are :

×	$2\ m^2\ \rho\ p$
+	$2\ m^2\ \rho\ p$
÷	$2\ m^2\ \rho$

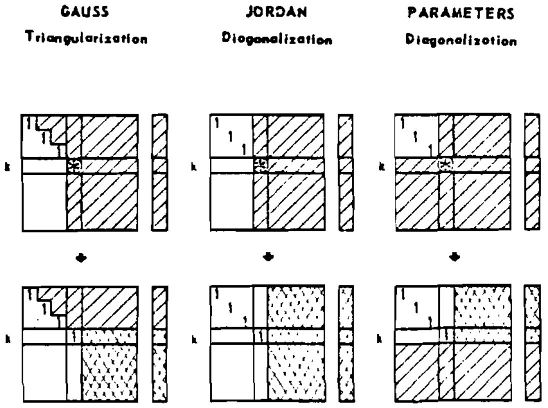
(A.15)

9. - ANNEX 2

COMPARISON BETWEEN THE PIVOTING METHODS OF GAUSS, JORDAN AND PARAMETERS

We suppose here that the considered matrix is full. This leads, in the method of parameters, to use m parameters (which of course presents no interest from a practical point of view). In this special situation, the method of parameters is a pivoting method with diagonalization, as JORDAN's method. However, its cost is exactly that of GAUSS's method, which is a pivoting method with triangularization.

In Figure 11, are given the details for the k-th stage for each method, together with the comparative account of the calculations.



x	$\frac{m(m-1)(2m+5)}{6}$	$\frac{(m-1)[m(m+1)-2]}{2}$	$\frac{m(m-1)(2m+5)}{6}$
+	$\frac{m(m-1)(2m+5)}{6}$	$\frac{(m-1)[m(m+1)-2]}{2}$	$\frac{m(m-1)(2m+5)}{6}$
÷	$\frac{m(m+1)}{2}$	m^2	$\frac{m(m+1)}{2}$

Figure 11

10. - REFERENCES

[1] D. FAYARD et G. PLATEAU - "Recherche du nombre minimum de paramètres". Note interne, Laboratoire de Calcul, Université de Lille I, 1977.

[2] Y. HAUW, Rapports internes, Laboratoire de calcul, Université de Lille I, 1977-1979.

[3] P. HUARD - "Problèmes de memorisation soulevés par la résolution numérique des grands problèmes d'optimisation". Colloque national d'analyse numérique, Port-Bail (24-29 may 1976). See too : "Résolution des programmes linéaires de grande taille". Bulletin de la Direction des Etudes et Recherches, EDF, Série C, n° 1, 1979.

[4] J.F. MAURRAS - "Optimization of the flow through networks with gains", Mathematical Programming 3 (1972) 145-156.

[5] W. ORCHARD-HAYS - "Factoring LP block-angular bases", Mathematical Programming Study 4 (1975) 75-92.

[6] H.J. RYSER - "Combinatorial mathematics" - John Wiley, 1963.

[7] R.P. TEWARSON - "Row column permutation of sparse matrices", Computer Journal 10 (1967) 300-305.

[8] J. DENEL - "Implémentation de la méthode des paramètres en optimisation linéaire", A.N.O. 23 (Septembre 1980), Laboratoire de calcul, Université de Lille I.

[9] W. DE LA VEGA et J.F. MAURRAS, Private communication, 1980.

EXPLOITING DEGENERACY IN THE SIMPLEX METHOD

André F. Peroldt[†]

Mathematical Sciences Department[††]
IBM T.J. Watson Research Center
Yorktown Heights, N.Y.

The simplex method often performs a large number of degenerate iterations on linear programs encountered in practice. This paper studies degeneracy from the point of view of reducing the computational effort per degenerate iteration. First, the simplex method is viewed as performing a sequence of nondegenerate iterations with the direction of movement at each such iteration being determined by an auxiliary linear program having as many rows as there are degenerate basic variables in the current solution. Then we show that the computations in this setting can be conveniently performed by means of a basis factorization method, achieving its savings by being able to perform degenerate iterations with only partial information. This method seems best suited for use with multiple pricing.

[†] I am grateful to George B. Dantzig for the stimulating discussions of which this work is an outgrowth.

[††] Currently at:

Graduate School of Business Administration
Harvard University

1. INTRODUCTION

A linear programming basic feasible solution is said to be *degenerate* when it contains zero valued basic variables. Call these *degenerate variables*. A *degenerate iteration* in the simplex method is a (feasible) change of basis with no improvement in the objective value.

The presence of degenerate solutions in linear programming is troublesome both theoretically and computationally. In the former, the possibility of cycling (an infinite number of iterations) cannot be ruled out without special pivot selection tiebreaking rules (e.g. [1], [2]). In the latter most problems encountered in practice exhibit some degree of degeneracy, and even though the simplex method almost never cycles on such problems, it nevertheless usually performs a high proportion of degenerate iterations [7]. (Our own experience indicates that a problem with on the average 20% of its variables degenerate usually results in approximately 50% of its iterations being degenerate.)

In this paper we study degeneracy from the point of view of reducing the computational effort per degenerate iteration. We begin by viewing the simplex method as performing a sequence of nondegenerate iterations, with the direction of movement at each such iteration being determined by an auxilliary linear program having as many rows as there are degenerate basic variables in the current solution. Then we show that the computations in this setting can be conveniently performed by means of a basis factorization method which achieves its savings by being able to perform degenerate iterations with only partial information. We indicate that this method should be best suited for use with *multiple pricing* [4], a technique that considers several candidates at once for introduction into the basis.

2. RESOLVING DEGENERACY: A SUBPROBLEM

Let the given problem be

$$\begin{aligned} &\text{minimize} && c^Tx \\ &\text{subject to} && Ax = b; \;\; x \geq 0. \end{aligned} \tag{1}$$

Denote a basic feasible solution generically by $x = (u,v,y)$, where u, v and y are respectively the basic variables at positive level (nondegenerate variables), basic variables at zero level (degenerate variables), and nonbasic variables. Let B denote the generic basis submatrix of A.

For a given feasible basis B, we may express the basic variables in terms of the nonbasic variables to obtain an equivalent problem

$$\begin{aligned} &\text{minimize} && p^Ty \\ &\text{subject to} && u + Cy = q \;\; (q>0) \\ & && v + Dy = 0 \\ & && (u,v,y) \geq 0 \end{aligned} \tag{2}$$

(We ignore the constant term difference between the objective values of (1) and (2)). The form of (2) will be considered generically, being equivalent to the usual canonical simplex tableau [2].

Suppose we now perform the simplex method (under a given pivoting rule) and that k (≥ 0) iterations occur before either a strict improvement in the objective value or a proof of optimality is obtained. Two observations are immediate:

1. Iterations $1,..,k$ will consist of exchanges of nonbasic variables (y) with degenerate basic variables (v).

2. If iteration k+1 yields a strict decrease in the objective value, this can only occur if $D_s \leq 0$, where y_s is chosen as the entering variable.

From this it is clear that in order to move from one basic solution to another with a strict improvement in the objective value, the simplex method is indeed solving the subproblem

$$\begin{array}{ll} \text{minimize} & p^T y \\ \text{subject to} & v + Dy = 0 \\ & (v,y) \geq 0. \end{array} \tag{3}$$

This is a linear program whose variables all remain at zero level until an unbounded solution is detected, at which point it is terminated. Once (3) has been solved, a change in the degeneracy structure occurs with a simultaneous exchange of degenerate (v) and nondegenerate (u) variables.

We wish to regard (3) as being distinct from the original problem for the following reasons:

1. Being generally much smaller in size, it may prove worthwhile to solve it on the side in some sense. If the simplex method performs many degenerate iterations or if the degeneracy structure does not vary greatly from one nondegenerate step to the next, then (3) represents that part of the tableau changing most rapidly. Exploiting this is the subject of the next section.

2. *Totally degenerate* linear programs such as (3)† are in a way very different from their nondegenerate counterparts, and may (at least a priori) be better solved by methods other than the usual pivoting rules. Firstly, feasibilty is always assured: every nonsingular submatrix of columns of (I,D) is a feasible basis for (3). Secondly, (3) can be solved by inspection if there is a column satisfying

$$D_s \le 0, \quad p_s < 0. \tag{4}$$

Thus if we had the updated tableau at our disposal, this should be the first criterion for an incoming column rather than, say, $p_s = \min \{p_i\}$. Further, if no such column exists, we wish to perform an exchange of columns with the hope that there will be such a column at the next step: to this end, there seems little justification for selecting an incoming column with $p_s < 0$, or restricting the pivotal element to be positive. In the revised simplex method [2] where for reasons of cost the full updated tableau is not available, choosing the incoming column with $p_s < 0$ may therefore be viewed as maximizing the probability that, in addition, $D_s \le 0$. With the added use of multiple pricing, however, a few columns of the updated tableau are kept at hand. In particular, this can be used to advantage in seeking a column satisfying (4), and is well suited for use with the method presented next.

† Every linear program (1) may be stated in a totally degenerate form: maximize t subject to $Ax = bt$, $A^Tw \le ct$, $c^Tx \le b^Tw$, $x \ge 0$ [2, p.290].

3. A DEGENERACY EXPLOITING BASIS FACTORIZATION METHOD

The heart of most implementations of the simplex method is the manner in which the basis is represented. Usually one chooses a factorization that can be used efficiently and stably in solving for the prices and the representation of the incoming column - as required in the revised simplex method, and can be easily updated from one iteration to the next.

The method proposed here is, like a great many others, based on partitioning and tearing (see [3]). Consider first the following general factorization scheme:

Partition

$$B = \begin{pmatrix} T & H \\ F & G \end{pmatrix} = (B^1, B^2) \tag{5}$$

arbitrarily but so that T is nonsingular. Then B may be factorized as the product

$$B = \begin{pmatrix} T & O \\ F & I \end{pmatrix} \begin{pmatrix} I & \bar{H} \\ O & \bar{G} \end{pmatrix} = L\ V \tag{6}$$

for appropriate $\bar{G}$ and $\bar{H}$. In order to solve equations with respect to B and B^T, it suffices to have T and $\bar{G}$ in factorized form. Typically, T is chosen to have a convenient form, e.g. triangular, so that most of the work centers around $\bar{G}$ and $\bar{H}$, F already being part of B. To save on storage, the requisite equations may also be solved without knowledge of $\bar{H}$, an approach we favor here (see e.g. [5]). For example, to determine the representation of the incoming column, a, the system

$$Bz = a$$

may be solved as follows:

$$\begin{aligned} Lw &= a \\ \overline{G}z^2 &= w^2 \\ L\binom{z^1}{0} &= a - B^2z^2 \end{aligned} \qquad (7)$$

where $w = (w^1, w^2)$ and $z = (z^1, z^2)$ are partitioned appropriately.

In the context of degeneracy, let B be partitioned so that B^1 and B^2 are the columns corresponding to nondegenerate and degenerate variables respectively[†]. Observe that $\overline{G}$ and $\overline{H}$ are then simply parts of the tableau updated relative to the basis L. In addition, $\overline{G}$ is the starting basis for the subproblem (3).

Suppose next that the simplex method applied to (1) with starting basis B performs some degenerate iterations followed by one that is nondegenerate. Since degenerate iterations involve replacements only of degenerate variables, the change in our factorization of B in (6) is localized to an exchange of columns in $\overline{G}$ alone (assuming we discard $\overline{H}$). Further, in solving for the representation of the incoming column we would only partially solve (7) in order to obtain z^2, which is all that is needed to perform a degenerate iteration. (z^2 here is D_s in the previous section).

$z^2 \leq 0$ indicates that the current iteration is nondegenerate. We would then solve the third system in (7) for z^1, and determine the leaving column by means of the usual minimum ratio test. At this point the update of the factori-

[†] For the moment we require only that T be nonsingular.

zation (6) is more cumbersome because a change in the degeneracy structure occurs. Restoring (6) in conformance with the new partition into degenerate and nondegenerate columns will most likely be too costly, and an easier method (both conceptually and computationally) may be to border $\bar{G}$ appropriately, leaving the factor L untouched.

More specifically, if the entering column, a, replaces a nondegenerate column of B, and moreover a subset α of the nondegenerate variables become degenerate, we would enlarge $\bar{G}$ to obtain

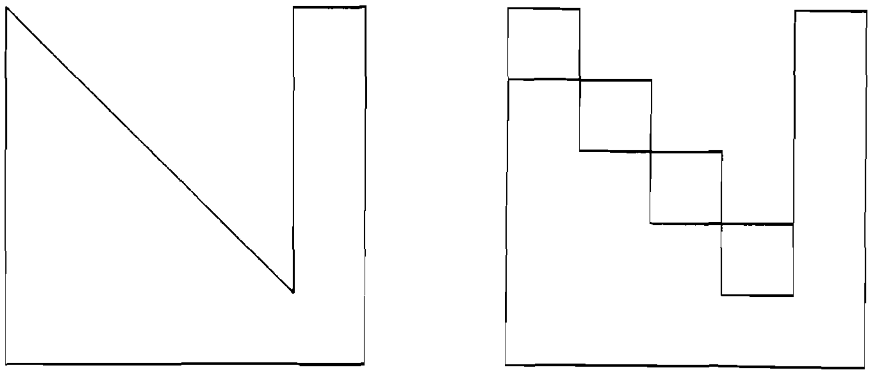
$$\bar{G}\ (\text{new}) = \begin{pmatrix} I & w^1_\alpha & \bar{H}_{\alpha.} \\ O & w^1_r & \bar{H}_{r.} \\ O & w^2 & \bar{G} \end{pmatrix}$$

where r is the pivot row and w is determined in (7). The work here is then to generate the required rows of $\bar{H}$ [†] followed by an update of whatever factorization is employed for $\bar{G}$. Note that $\bar{G}$ now represents the degenerate variables together with the nondegenerate variables that were initially degenerate.

Periodically we would begin the process from scratch by *reinversion*, indicated either by $\bar{G}$ requiring excessive storage or by loss of numerical accuracy. In the event that a large number of nondegenerate variables become degenerate at any one iteration, finding the rows of $\bar{H}$ for the bordering process may become prohibitive, and it may be profitable simply to treat these new degenerate variables as being nondegenerate, then performing reinversion earlier than otherwise.

[†] Finding a row of $\bar{H}$ requires the solution of a system with respect to L and the inner product of this solution and B^2.

Several existing factorization algorithms (e.g. [3], [5]) attempt to reduce storage requirements by permuting B to bordered triangular or block triangular form:



This corresponds in (5) to choosing T as large a (block) triangular matrix as possible, and can be adapted easily to our case by letting $\overline{G}$ initially represent both the degenerate variables and the bordered nondegenerate variables (called *spikes*). L, too, is then a (block) triangular matrix which between reinversions remains fixed in this desirable form.

In cases where a very sparse (or otherwise desirable) factorization of B is available that is not of this near (block) triangular form, the method still applies: Let L be all of B (in this desirable factorized form) and begin with $\overline{G}$ being the identity corresponding to the degenerate variables.

3. SUMMARY AND CONCLUSIONS

We have proposed a basis factorization algorithm intended to exploit the degeneracy that has been observed to occur in linear programs encountered in practice. A typical simplex iteration begins with the basis represented by two systems (see (6)): the first, L, is of a desirable form, e.g. triangular, remains fixed between iterations, and is associated largely with nondegenerate variables; the second, $\overline{G}$, represents most of the degenerate variables together with the remaining nondegenerate variables. The iteration proceeds as follows:

1. Select an incoming column, a, by a suitable pricing mechanism or otherwise. If there is none, the solution is optimal: stop.
2. Solve the equations

$$\begin{aligned} Lw &= a \\ \overline{G}z^2 &= w^2. \end{aligned}$$

3. If the degenerate part of z^2 has any positive components select the largest one as the pivotal element (or any other depending on the pivot rule), and go to step 4. Else go to step 5.
4. This is a degenerate iteration: exchange the column w^2 with the column leaving $\overline{G}$ as selected in step 3. Return to step 1.
5. This is a nondegenerate iteration: solve the system

$$L\begin{pmatrix} z^1 \\ 0 \end{pmatrix} = a - B^2z^2.$$

6. Select the pivot row by performing the usual minimum ratio test on z and the updated right hand side. If none can be selected, the solution is unbounded: stop.
7. Update the right hand side and determine the new degeneracy structure.

8. Update $\overline{G}$ by bordering it with the appropriate rows and columns determined by this column exchange and also by the occurrence of any new degenerate variables (if desired). Go to step 1.

This method should significantly reduce the time spent on degenerate iterations since it localizes the area of most rapid change in the basis factorization and allows one to execute these iterations with only partial information. Nevertheless the advantage gained could be offset by potentially large changes in the degeneracy structure of the basic variables. However, investigative test runs on a variety of problems have shown that the average change in the degeneracy structure from one nondegenerate step to the next is indeed very slight. Experimentation is currently under way with an adaptation of these ideas to the LU factorization, and will be reported in [6].

We remark, finally, that the advantages of this method should be sigificantly enhanced with the use of multiple pricing. This is so for two reasons: Firstly, the effects of being able to perform exchanges of columns cheaply are even more pronounced when pricing is carried out only, say, every 5 iterations. (In our experience it has been common to spend 50% of the iteration time computing the prices and pricing out the nonbasic variables). Secondly, as indicated in section 2, having part of the updated tableau at our disposal can result in fewer degenerate iterations because of increased flexibility in choosing the entering column. Only the degenerate part of the updated tableau is required in this case, being precisely what this method was intended to find efficiently.

REFERENCES

[1] R.G. BLAND, "New finite pivoting rules for the simplex method", *Mathematics of Operations Research* **2** (1977), 103-107.

[2] G.B. DANTZIG, *Linear Programming and Extensions,* Princeton University, Princeton, 1963.

[3] I.S. DUFF, "A survey of sparse matrix research", *Proc. IEEE,* **65** (1977), 500-535.

[4] W. ORCHARD-HAYS, *Advanced Linear Programming Computing Techniques,* McGraw-Hill, New York, 1968.

[5] A.F. PEROLD and G.B. DANTZIG, "A basis factorization method for block triangular linear programs", *Proceedings of the Symposium on Sparse Matrix Computations,* Knoxville, Tennessee, I.S. Duff and G.W. Stewart (Eds), 1978.

[6] A.F. Perold, "A degeneracy exploiting LU factorization for the simplex method", (1979), in preparation.

[7] P.S. WOLFE, A technique for resolving degeneracy in linear programming", *J. Soc. Indust. Appl. Math.* **11** (1963), 205-211

A DEGENERACY EXPLOITING LU FACTORIZATION FOR THE SIMPLEX METHOD*

André F. Perold

Mathematical Sciences Department†
IBM T.J. Watson Research Center
Yorktown Heights, N.Y.

For general sparse linear programs two of the most efficient implementations of the LU factorization with Bartels–Golub updating are due to Reid and Saunders. This paper presents an alternative approach which achieves fast execution times for degenerate simplex method iterations, especially when used with multiple pricing. The method should have wide applicability since the simplex method performs a high proportion of degenerate iterations on most practical problems. A key feature of Saunders' method is combined with the updating strategy of Reid so as to make the scheme suitable for implementation out of core. Its efficiency is confirmed by experimental results.

*Published in *Mathematical Programming* 19:239–254, 1980. Reproduced with permission. Copyright © 1980 The Mathematical Programming Society.

†Currently at:
Graduate School of Business Administration
Harvard University

1. INTRODUCTION

Implementations of the simplex method usually comprise two often independent aspects. The first is the manner in which the columns entering and leaving the basis are selected, the primary aim being a reduction in the overall number of iterations (e.g. Harris [7] and Goldfarb and Reid [5]). The second is the means of maintaining the basis in factorized form so that the requisite equations can solved efficiently, and therefore reduce the computational effort per iteration. One's choice of factorization method is usually guided by numerical stability, the structure of the problem, and the particulars of the computer.

For general large sparse linear programs two of the most efficient factorization and updating methods are due to Reid [15], [16], and Saunders [18], [19]. Both are implementations of the the LU factorization with Bartels-Golub updating [1]: Reid computes the factors using a Markowitz strategy [10] with threshold pivoting, and performs the updating with the use of row and column permutations on U so as to effectively minimize the growth of nonzeros. This method favors having a greater proportion of nonzeros in U, and requires that all of U be kept in core. Saunders' method, on the other hand, is aimed at keeping as much as possible in secondary storage, and is ideal for problems that are very large or that will otherwise require excessive paging. Here the LU factors are determined by the "bump and spike" structure of the basis. By collecting the spikes after Gaussian elimination has been performed most of the nonzeros go into L. All that is kept in core is the small upper triangular submatrix F of U which remains after deletion of the rows and columns of U corresponding to triangle pivots. Sparsity is well preserved during updating since the growth of nonzeros is confined to F. Recently, Gay [4] has experimented with an improvement over Saunders' implementation by updating F with Reid's method.

This paper describes an alternative implementation of the LU factorization that is more intimately connected with the iteration path of the simplex method. The main features of this approach are:

1. Degenerate simplex method iterations can be performed with far less computational effort;
2. It can be used profitably with multiple pricing to allow increased flexibility in choosing the entering column and so reduce the overall number of degenerate iterations;
3. It is similar to the method of Saunders in that primary storage need be allocated only for its analogous F matrix. This likewise facilitates the efficient use of Bartels-Golub updating, particularly as implemented by Reid.

Most of the underlying ideas here stem from a more theoretical discussion in Perold [14], although it is intended that this presentation be self contained.

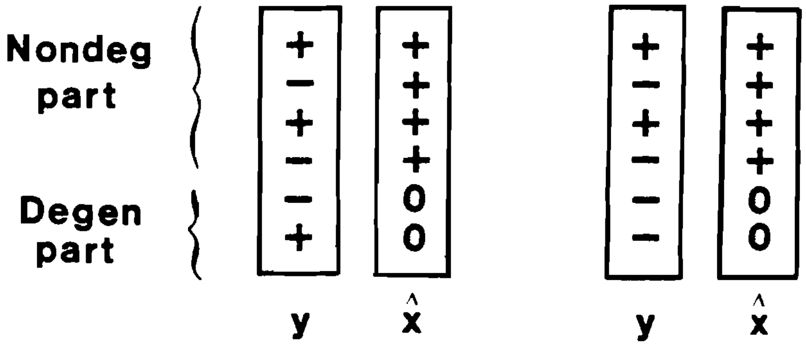
2. PRELIMINARY FACTS AND OBSERVATIONS

The method discussed here exploits two empirically observed phenomena of the bases of sparse practical linear programs: a moderate number of degenerate columns[1], perhaps between ten and thirty percent of the total number of basic columns, and a small number of spikes[1], somewhere between 1 and 100. We shall later indicate how it can be modified so as not to be adversely affected on problems having a large number of degenerate columns. However, its performance will deteriorate markedly as the number of spikes gets large.

2.1 Degeneracy

We call a column of a given feasible basis *degenerate* if its corresponding basic variable is at zero level. The presence of degeneracy is theoretically troublesome since the simplex method may cycle (an infinite repetition of a basis) without the use of special rules for selecting the entering and leaving columns (e.g. Dantzig [3], Bland [2]). On practical problems, however, cycling is rare. Nevertheless, degeneracy usually results in a great many *degenerate iterations,* these being feasible basis changes with no improvement in the objective value. Indeed, it is typical for a problem with an average of 20% of its basic columns degenerate to result in 50% of its iterations being degenerate.

Figure 1 illustrates the difference between degenerate and nondegenerate iterations. $\hat{x}$ is the updated right hand side and y is the representation of the entering column.



a) Degenerate iteration b) Nondegenerate iteration

Figure 1

Iteration (a) is degenerate because of the presence of a positive entry in the degenerate part of y. The points of note are the following:

1. Degenerate iterations can be carried out without any knowledge of the nondegenerate part of y. Only if the degenerate part of y has no positive elements is it necessary to consider the remainder of y in order to select the leaving variable by means of the usual minimum ratio test.
2. Degenerate iterations consist of replacements of degenerate columns only. Only during a nondegenerate iteration can (and usually does) the degeneracy structure change.

These facts lie at the heart of the method of this paper.

2.2 Near triangularity

Spikes are columns having nonzeros above the diagonal. These were considered first in the context of linear programming by Hellerman and Rarick [8] who observed that the bases of sparse practical linear programs could usually be permuted to a form that is near lower triangular in the sense of having very few spikes. They proposed a heuristic[1] P^3 to accomplish this, and then improved on it [9] by first determining the maximal block triangular structure of the basis[2] (this is unique) and then applying P^3 to the irreducible diagonal blocks, called *bumps*.

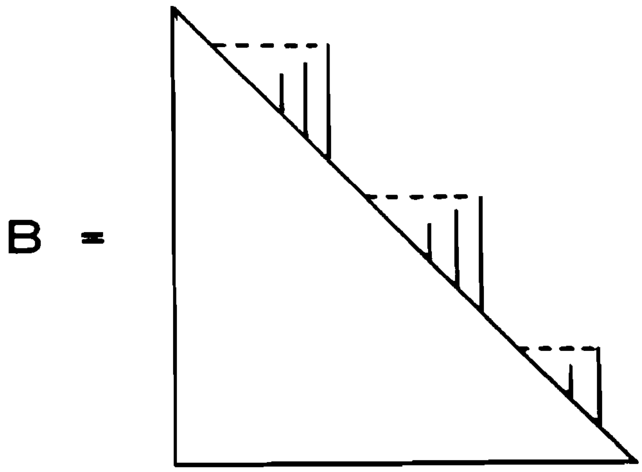


Figure 2: Bump and spike structure of B

[1] The problem of finding the minimal number of spikes is NP-complete [17].

[2] There are now very efficient algorithms to determine the block triangular structure, e.g. Gustavson [6].

Observe that by moving the spikes to the end of B in a principal rearrangement, we obtain a bordered triangular form (Figure 3). A recent efficient heuristic to permute a sparse matrix to this form with minimal border is due to Sangiovanni-Vincentelli and Bickart [17]. At the present time there are no comparative results with P^3.

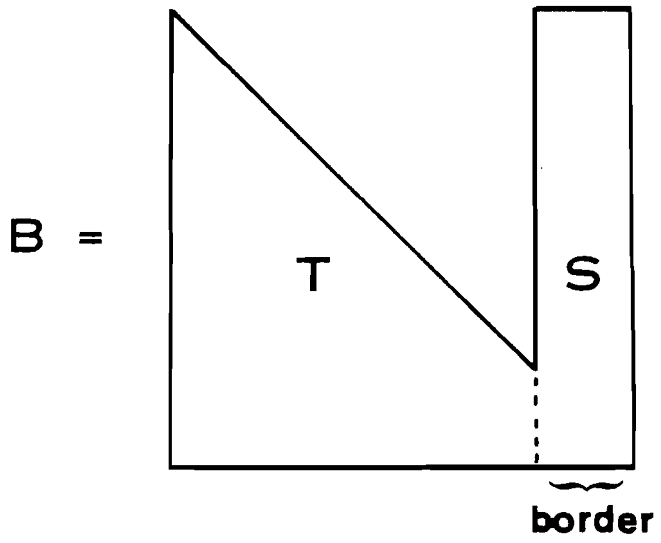


Figure 3: Bordered triangular form of B

The advantage of preprocessing the basis in either of the above ways is that the growth of nonzeros during Gaussian elimination is confined to the few spike (border) columns. Although the row and column order given by the bump and spike structure usually yields sparser LU factors than the order given by the bordered triangular form[1], the latter will nevertheless be more suitable for our purposes since it yields a much sparser L factor.

[1] The extent to which this is true is worthy of investigation.

3. THE LU FACTORS OF B

From the previous section we assume that B has the bordered triangular form depicted in Figure 3. Performing Gaussian elimination in this preassigned order yields L and U of the form in Figure 4, where T is the triangular part of B and remains unchanged in L, and R, F, and E represent the spike columns S transformed by pivoting first on the diagonal of T.

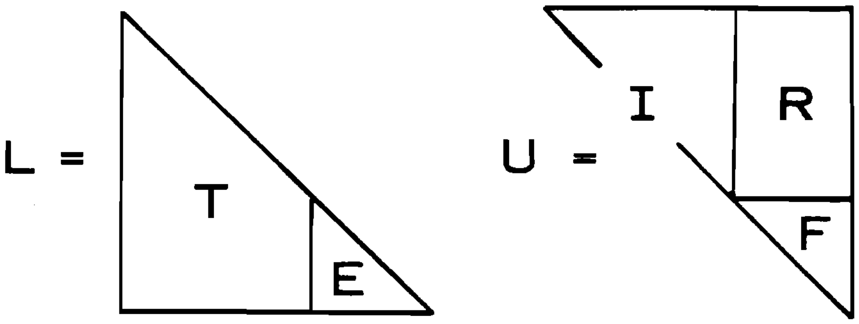


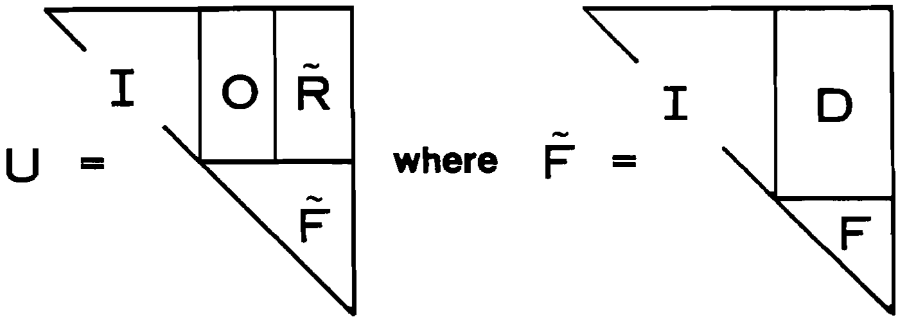
Figure 4

Remarks

1. Fill-in occurs in all three of R,. F and E.

2. It can be easily seen that F here would be the same as that obtained by Saunders when all the spikes appear at the ends of their bumps. Since this is usually the case for most spikes, we can expect the two F's to be very similar.

The next step is the further partitioning of U according to the degeneracy structure of B. The degenerate columns of B bear no relation to its bordered form although they will be made up mostly of triangle columns since there are usually so few spikes. Perform the principal permutation on U that collects all the rows D (say) of R corresponding to degenerate triangle columns and places them adjacent to F. This gives U the form depicted in Figure 5.



$$\tilde{R} = R \setminus D$$

Figure 5

With L and U now determined in this way, we consider performing a basis change. In the remainder of this paper we shall identify the columns of U and $\tilde{F}$ with their corresponding columns in B. Thus we call a column of $\tilde{F}$ degenerate if its corresponding column in B is degenerate.

4. PERFORMING A BASIS CHANGE

A substantial part of the computational effort in each iteration of the simplex method consists of selecting the column to leave the basis — for a given entering column a — and then updating the current to reflect this exchange.

In order to determine the leaving column we need to solve the system

$$By = a.$$

This we do by solving the systems

$$\begin{aligned} Lw &= a \\ Uy &= w. \end{aligned}$$

By partitioning $w = (w^1, w^2)$ and $y = (y^1, y^2)$ according to the above partition of U (Figure 5), it is clear that y can be obtained from w in a two step procedure:

$$\begin{aligned} \tilde{F}y^2 &= w^2 \\ y^1 &= w^1 - (0, \tilde{R})y^2. \end{aligned}$$

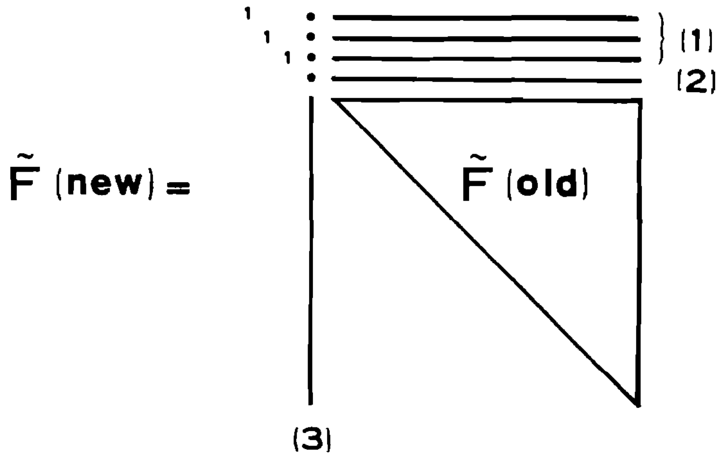
Following our discussion on degeneracy in section 2.1, we see that it suffices to have y^2 in order to perform a degenerate iteration since it contains all the degenerate components of y. Only if all of these are nonpositive is the computation of y^1 required. Further, since a degenerate iteration is the exchange of the entering column and a degenerate column, no new degenerate columns are created. Thus the factorization may be updated simply by an exchange of w^2 for the leaving column of $\tilde{F}$ (e.g. by Bartels-Golub updating).

The update during a nondegenerate iteration needs to be performed in two stages:

1. Perform the update corresponding to the exchange of columns. Unless the leaving column is a nondegenerate column of $\tilde{F}$, this will result in the formation of an additional spike which can be handled precisely as in Saunders' case with his F replaced by $\tilde{F}$.

2. Update the degeneracy structure. In principle several degenerate variables can become nondegenerate, and several nondegenerate variables can become degenerate. In practice, it is common for no more than 2 nondegenerate variables to become degenerate, and it is easiest to border $\tilde{F}$ appropriately to accommodate them (in the same way as F was bordered to obtain the initial $\tilde{F}$), leaving untouched any degenerate columns that may have become nondegenerate.

Schematically the new $\tilde{F}$ has the following form:



(1) The rows of $\tilde{R}$ corresponding to new degenerate columns not in $\tilde{F}$.
(2) The row of $\tilde{R}$ corresponding to the leaving column of B.
(3) Subvector of w = new spike to be eliminated by Bartels-Golub.

5. DISCUSSION

5.1 Discarding $\tilde{R}$

Observe that $\tilde{R}$ is required only during nondegenerate iterations: it is used in the solution of y^1 and for the bordering of $\tilde{F}$ with a few of its rows. During degenerate iterations it is accessed only to save the subvector w^1. The above steps can all be performed without knowledge of $\tilde{R}$. By looking at the rows of B corresponding to y^1 as depicted in Figure 3, it is clear that there is a triangular submatrix $\tilde{T}$ of T and a submatrix $\tilde{S}$ of S so that y^1 satisfies

$$\tilde{T}y^1 + (0, \tilde{S})y^2 = a^1$$

where $a = (a^1, a^2)$ is partitioned accordingly. Thus y^1 may be determined by solving a triangular subsystem of L. Likewise, the p^{th} row of $\tilde{R}$ may be obtained by solving the system

$$\tilde{T}^T z = e_p \qquad (p^{th} \text{ unit vector})$$

and forming the inner product $z^T\tilde{S}$. This can be used to save substantially on storage since L, $\tilde{T}$ and $\tilde{S}$ can be embedded as part of the constraint matrix. Additional storage would then be required only for E and $\tilde{F}$.

In an out-of-core implementation the storage aspect is not all that important, however, and we would probably store L, $\tilde{T}$ and $\tilde{S}$ separately so as to be more easily accessible. Nevertheless, since $\tilde{T}$ and $\tilde{S}$ are generally much less dense than $\tilde{R}$, it may still pay to perform the calculations with them instead.

5.2 Excessively many degenerate columns

The success of this method hinges on its ability to confine most of the work to the small triangular submatrix $\tilde{F}$ which is to be kept in core. The order of $\tilde{F}$ is determined primarily by the number of degenerate columns, and may become too large for two reasons:

1. If, say, 70% of the columns are degenerate then $\tilde{F}$ constitutes most of U, and the savings during degenerate iterations will probably be slight.
2. Even if $\tilde{F}$ is a proportionately small part of U, it may nevertheless require too much core, as may happen with extremely large problems.

In either case, this method can still be made viable by treating sufficiently many degenerate columns as being nondegenerate (for the purposes of this factorization only) so as to keep the core requirements of $\tilde{F}$ manageable. Thus even though y^2 no longer contains all the degenerate components of y, we still require y^1 only if there are no positive degenerate components in y^2. A good strategy would seem to be to keep $\tilde{F}$ as large as possible subject to core availability and/or to there still being some benefit over a method that keeps all of U in core (with perhaps sparser L and U factors).

5.3 Large changes in the degeneracy structure

In the rare event that a large number k of nondegenerate columns not currently in $\tilde{F}$ become degenerate (only possible during a nondegenerate iteration), updating $\tilde{F}$ by bordering it with the corresponding k rows of $\tilde{R}$ can become expensive. This would be especially so if we need to generate these rows because $\tilde{R}$ is not maintained. In such a case it may be better to temporarily treat these added degenerate columns as nondegenerate columns — in a fashion similar to the approach in section 5.2 — and then perform refactorization earlier than usual.

5.4 Summary

The procedure for performing a basis change may now be summarized below. We assume that the entering column a has been selected. As before, $\hat{x}$ is the current updated right hand side.

1. Solve $$Lw = a$$ $$\tilde{F}y^2 = w.$$

2. If the degenerate part of y^2 has no positive components go to step 4. Else select one of them as the pivot element.

Degenerate iteration

3. Exchange w^2 for the leaving column in $\tilde{F}$, and add w^1 to $\tilde{R}$ if $\tilde{R}$ is being maintained. Restore $\tilde{F}$ to upper triangularity (by Bartels-Golub updating). *End of iteration.*

Nondegenerate iteration

4. Solve one of $$\tilde{T}y^1 = a^1 - (0, \tilde{S})y^2$$ $$y^1 = w^1 - (0, \tilde{R})y^2.$$

5. Determine the leaving column by means of the usual minimum ratio test on $\hat{x}$ and y.

6. If the leaving column is not in $\tilde{F}$ generate row p (say) of $\tilde{R}$ corresponding to the pivot row, either by retrieval from secondary storage, or, if $\tilde{R}$ is not being maintained, by solving $\tilde{T}^T z = e_p$ and forming $z^T\tilde{S}$. Augment $\tilde{F}$ with this row.

7. Proceed as in step 3.

8. Update the right hand side.

9. Determine the new degneracy structure, and generate (as in step 6) the rows of $\tilde{R}$ corresponding to new degenerate columns not already in $\tilde{F}$.

10. Augment $\tilde{F}$ with these rows and appropriate unit columns, maintaining upper triangularity. (This step requires no arithmetic). *End of iteration.*

Remark

Note that each iteration requires the solution of systems with respect to L and $\tilde{F}$, and the elimination of a single spike to restore $\tilde{F}$ to upper triangularity. As such, it is important to have L and $\tilde{F}$ in as compact a form possible: permuting the spikes to the end before performing Gaussian elimination brings us much closer to this goal. An alternative may be to find the best bordered form from amongst only the nondegenerate columns (i.e. a rectangular matrix). This should yield a "thinner" border, but may result in much more fill-in in the degenerate columns. While not considered here, this approach seems worthy of investigation.

5.5 Use with multiple pricing

Computing the prices and determining the incoming column can often cost as much as 50% of the iteration time. Multiple pricing [12] is intended to save on most of this by selecting several columns at once for introduction into the basis. Typically between 5 and 10 columns are selected and introduced one at a time subject to remaining profitable. Their representations are kept in core and are updated as if in a tableau. In addition to the savings in pricing, one can reduce the overall number of iterations by choosing from amongst these, for example, the column yielding the greatest decrease in the objective value.

With this factorization the savings during degenerate iterations will be even more pronounced when pricing is not performed at every iteration. For each of the 5 to 10 columns we would compute and store their w^2 and y^2 subvectors as usual and then try to select from these a column whose degenerate part is nonpositive. From one of these "minor" iterations to the next the w^2's can be updated by the transformations used to update $\tilde{F}$. During nondegenerate minor iterations the y^2's gain additional components. These can be easily found by using the rows of $\tilde{R}$ being added to $\tilde{F}$. Then the tableau updating formulae apply as usual.

6. IMPLEMENTATION

In order to investigate the behavior of this factorization algorithm, particularly with respect to the distribution of nonzeros and the relative times spent on degenerate and nondegenerate iterations, we implemented the foregoing proposals in an experimental code DELUX (Degeneracy Exploiting LU simpleX). DELUX was written in FORTRAN IV and run on an IBM 370/168 under VM (FORTHX compiler, OPT = 2). The important aspects are the following:

1. The constraint matrix is stored column wise with row pointers. Upper and lower bounds on the variables are kept in two separate arrays[1].
2. The maximal bump finding algorithm and P^3 were implemented as by Saunders in the code MINOS [11], [19].
3. To save on storage $\tilde{R}$ is discarded.
4. All triangle columns of B are represented by pointers into the constraint matrix. They are pivoted on first before any spikes are considered. Any of these with unacceptably small pivot elements (relative to the elements in the rest of the column) are rejected for pivoting at this stage and treated as spikes.
5. The square "remaining matrix" of the transformed spike columns is fed to Reid's routine LA05A [16] to be factorized into the product EF (see Figure 4 and the remark below). LA05A stores F row wise with column pointers, together with an additional set of row pointers used only to indicate the nonzeros column wise.
6. $\tilde{F}$ is formed by augmenting F with the rows D (Figure 5). This involves the insertion of these additional nonzeros row wise at the end of the file for F,

[1] In this case a basic column is degenerate if its variable is at its upper or lower bound.

followed by an update of the column structure and a permutation array. ($\tilde{F}$ and F are permuted upper triangular matrices).

7. During nondegenerate iterations, L is used in place of $\tilde{T}$ for the solution of y^1 and the generation of the required rows of $\tilde{R}$. Advantage is taken of the fact that many of the columns of L can be skipped during these transformations.

8. During updating, augmentation of $\tilde{F}$ takes place first (when necessary) as mentioned in 6. Then the column swap is performed on $\tilde{F}$ by Reid's routine LA05C [16].

Remark

Factorizing the remaining matrix in the already determined bump and spike pivot order may be a more efficient means of computing the initial E and F. However it was much easier implementationally to call on LA05A. This also has the added long run benefit of placing more weight into $\tilde{F}$: L can only grow in size while $\tilde{F}$ can actually shrink if a dense column is replaced by a sparse one; a sparser L yields a sparser transformed column w, which in turn yields a slower growth of nonzeros.

7. EXPERIMENTAL RESULTS

As our test problems we used 3 small- to medium-scale LP models.

Problem	Rows	Columns	Nonzeros	% Density	Iterations
PILOT8	626	1376	6026	0.7	500
SCSD8	398	2750	11334	1.0	456
L84MAV	114	1994	11120	4.9	1043

Table 1: Problem statistics

The first two are time period models: PILOT8 has an 8 period staircase structure with a few nonzeros in the lower block triangle; SCSD8 has a 39 period staircase structure. Earlier experience with these models on MINOS and LPBLK (an LP code employing a block triangular factorization of the basis) is reported in Perold and Dantzig [13]. L84MAV is a set covering problem and was chosen because such linear programs are known to be highly degenerate. All runs had the refactorization frequency set to 100 and were started from advanced feasible bases. These were the same starting bases for PILOT8 and SCSD8 as reported in [13]. Only PILOT8 was terminated short of optimality.

7.1 The initial LU

Two tolerances are used in determining the initial factorization:

1. u_T is the minimum acceptable ratio of the pivot element of a triangle column (of B) to the largest element beneath it. Triangle columns unacceptable in this way are moved to the end of B and treated as spikes.

2. u_M is the threshold used by LA05A in conjunction with the Markowitz strategy.

	PILOT8	SCSD8	L84MAV
Rows	626	398	114
Nonzeros	3388	1552	585
Density (%)	.86	.98	4.5
Slacks	37	1	10
Initial spikes	120	48	12
Triangle rejects	24	0	0
Dimension of F	144	48	12
Degenerate cols	52	117	21
Degenerate spikes	6	18	1
Dimension of $\tilde{F}$	190	147	32
Nonzeros[1]			
T[2]	2076	1311	503
E	2559	88	56
$L = T + E$	4635	1399	559
F	1473	203	75
D	88	241	36
$\tilde{F} = F + D$	1561	444	111
$\tilde{R}$[3]	3041	2575	342
Total: $L + \tilde{F} + \tilde{R}$	9237	4418	1012

[1] Refer Figures 4 and 5 in section 3.

[2] Embedded in the constraint matrix.

[3] These were not stored.

Table 2: Statistics for the initial LU

Problems SCSD8 and L84MAV were not very tolerance dependent. PILOT8 on the other hand was very sensitive to the tolerance u_T, having a large number of rejected triangle columns even with $u_T = .001$. The best result was obtained with $u_T = .0001$ and $u_M = .1$, this being barely satisfactory numerically. These tolerances were also used for the figures reported here for SCSD8 and L84MAV.

Table 2 summarizes the statistics for the initial LU. Of particular interest is the low proportion of nonzeros in $\tilde{F}$, even though the dimension of $\tilde{F}$ in all cases is approximately one third that of B. Perhaps more remarkable, and indeed very surprising, is the fact that the number of nonzeros in D (i.e. what is added to F to get $\tilde{F}$) is far out of proportion to its size relative to $\tilde{R}$. On PILOT8, for example, D has approximately 10% of the rows of $\tilde{R}$, yet less than 3% of its nonzeros. The only explanation for this is that the nonzeros in R are distibuted asymmetrically: very few at the top and a great many at the bottom.

Table 3 gives the initial LU statistics for MINOS on PILOT8 and SCSD8 from runs recorded earlier for [13]. On PILOT8 the same tolerance $u_T = .0001$ was used, resulting in the same number of triangle rejects. As expected, the factorization performed by DELUX has:

1. A much sparser L
2. A denser F (due mostly to the Markowitz strategy of LA05A) although not much more so in terms of the total number of nonzeros
3. A very much denser R.

While the total number of nonzeros in the factorization is less important for DELUX (since only L and $\tilde{F}$ are used for a large part of the time) it is worth noting that MINOS produces 24% more nonzeros on PILOT8 and 48% fewer nonzeros on SCSD8. The "almost catastrophic" fill-in in $\tilde{R}$ produced by

DELUX on SCSD8 is a result of the problem's staircase structure (39 stairs with approximately 10 rows in each). A staircase matrix with a high degree of partitioning is probably a worst case example for this type of behavior.

	PILOT8	SCSD8	L84MAV
T	2076	1311	Not. run
E[1]	6760	675	
L = T + E	8836	1986	
F	1323	68	
R	1255	244	
Total: L + F + R	11414	2298	

[1] E here consists of the subdiagonal parts of the filled-in spike columns.

Table 3: Nonzeros in the initial LU of MINOS

7.2 Some degeneracy statistics

The method of the paper is based in part on the assumptions that a relatively small number of degenerate columns result in a relatively large number degenerate iterations, and that large changes in the degeneracy structure at any nondegenerate iteration are rare. From Table 4 we see that the first assumption holds for the three test problems.

	Degenerate	columns	% Degenerate
	Mean number	Mean %	iterations
PILOT8	52	8.3	40
SCSD8	131	33	78
L84MAV	36	32	70

Table 4: Degenerate columns and degenerate iterations

Changes in the degeneracy structure are important only in so far as they affect the size of $\tilde{F}$. The frequency diagrams below summarize the distribution of the growth in dimension of $\tilde{F}$. This growth is made up of a new spike and/or the number of new degenerate columns that are not already in $\tilde{F}$. Note that the growth will slow down as $\tilde{F}$ gets larger (until the next refactorization) so that these figures should be interpreted as averages. From Figure 6 we see that for by far the bulk of the nondegenerate iterations, the dimension of $\tilde{F}$ either remains constant or goes up in size by one. On SCSD8 some isolated large increases were reported, most notably one of size 22.

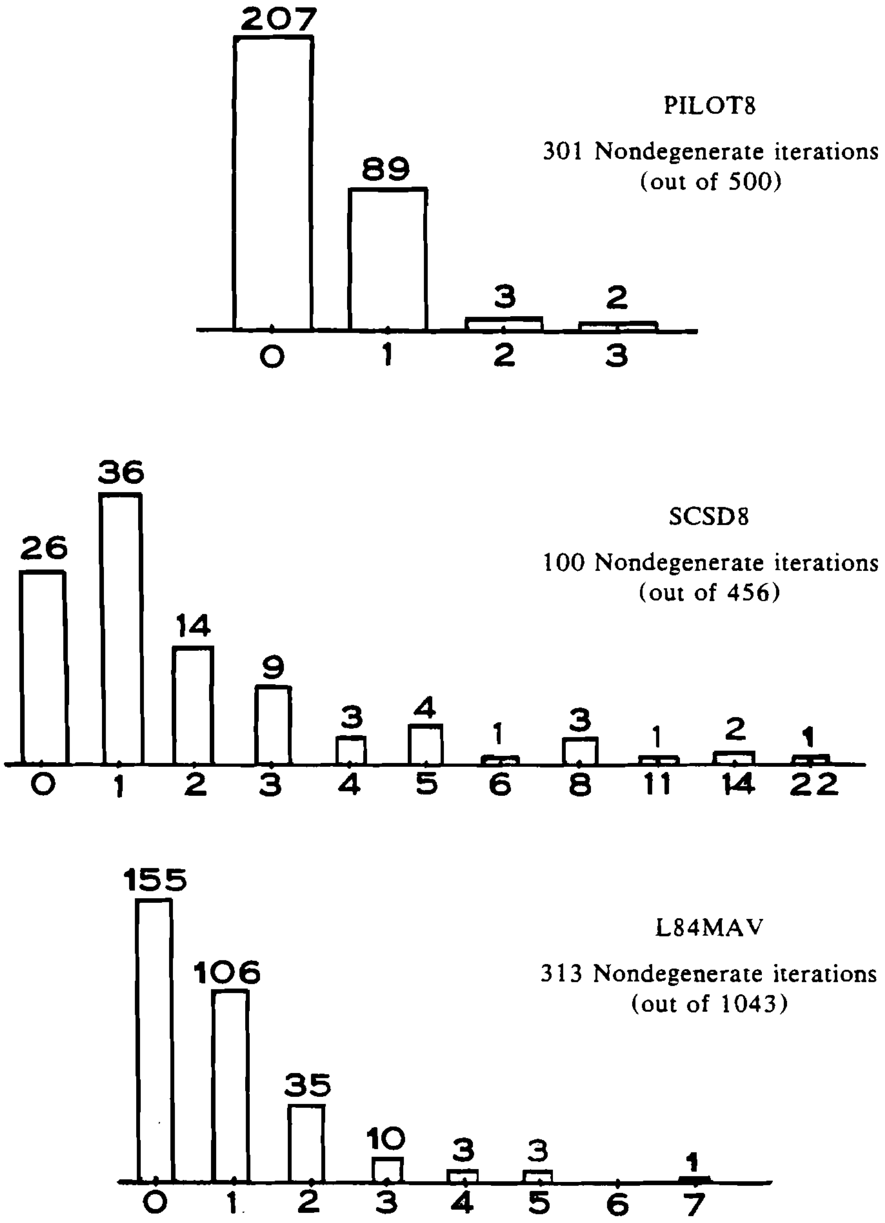


Figure 6: Frequency diagrams of the growth in dimension of $\tilde{F}$ during nondegenerate iterations

Table 5 indicates that the average growth in dimension of $\tilde{F}$ is slow even for nondegenerate iterations. The (overall) average growth in dimension of MINOS's F was .39 on PILOT8 and .73 on SCSD8. These are expected to be higher than those of DELUX since F is smaller than $\tilde{F}$.

	PILOT8	SCSD8	L84MAV
Nondeg itns	.34	1.93	.77
All iterations	.20	.42	.23

Table 5: Average increase in dimension of $\tilde{F}$

7.3 Growth of nonzeros during updating

The figures in Table 6 show a remarkably slow growth of nonzeros in L and $\tilde{F}$. This and the high proportion of iterations during which no growth took place in L are due both to Reid's updating method and the initial low density of L. With MINOS the growth rates for L were almost twice these: 18.8 for PILOT8 and 5.8 for SCSD8; SCSD8 had no growth in L 25% of the time. (This figure for PILOT8 and the growth of nonzeros in F were not available).

	% itns with no growth in L	Average growth in L	Average growth in $\tilde{F}$
PILOT8	44	9.9	16.3
SCSD8	46	3.3	11.9
L84MAV	50	5.4	7.6

Table 6: Average growth of nonzeros

7.4 CPU times

	PILOT8	SCSD8	L84MAV
Solve for prices	334	162	72
Select incoming col	108	81	62
1. Solve: y^2	255	91	51
2. Solve: y^1	130	66	34
3. Update: LA05C	113	42	42
4. Augmenting $\tilde{F}$	204	299	60
Degenerate basis change: 1+3	368	133	93
Nondeg basis change: 1+2+3+4	702	498	187

Table 7: Average CPU time per iteration (seconds x 10^{-4})

[1] The incoming column was selected by partial pricing, i.e. cyclic scanning of partitions of the constraint matrix. 3 equal partitions were used for PILOT8 and 10 for SCSD8 and L84MAV.

Table 7 shows nondegenerate basis changes taking twice as long as degenerate ones on PILOT8 and L84MAV, and much longer on SCSD8. Note, however, that much of the time for nondegenerate iterations went into generating the rows of $\tilde{R}$ to be added to $\tilde{F}$ (especially true on SCSD8). This can be improved upon by a more careful implementation in several ways:

1. Store $\tilde{T}$ and $\tilde{S}$ row wise (in secondary storage) instead of using all of L and S (stored column wise) as was the case here.

2. Do not discard $\tilde{R}$, and obtain the required rows directly from it. In addition, depending on the density of $\tilde{R}$, use whichever method is most economical to solve for y^1.

Even with a sharp reduction in the time spent on augmenting $\tilde{F}$, the large savings during degenerate basis changes is nevertheless clear. With the added use of multiple pricing (see section 5.5) the high proportion of the iteration time spent in selecting the incoming column should diminish to about 10%. This would make the total time for degenerate iterations about 35% faster than that for nondegenerate iterations.

8. CONCLUSIONS

We have presented a new implementation of the LU factorization that achieves fast execution times for degenerate simplex method iterations, especially when used in conjunction with multiple pricing. The scheme possesses a major benefit of Saunders' method, viz. requiring only part of U (i.e. $\tilde{F}$) in core. This greatly reduces primary storage requirements while simultaneously facilitating the efficient use of Bartels-Golub updating, particularly as handled by Reid.

Preliminary experimental runs indicate that the method might typically achieve a 35% savings in the run time for degenerate iterations. In so far as the available data allow for a comparison with Saunders' method, we conclude that while this method initially requires more storage for $\tilde{F}$ than his F, this is still only a fraction of the total number of nonzeros. This difference is in any event offset by a growth of nonzeros about half that of his, aside from the savings in time during degenerate iterations. Further testing is warranted in order to bring these tentative results into sharper focus.

REFERENCES

[1] R.H. BARTELS and G.H. GOLUB, "The simplex method of linear programming using LU decomposition", *Communications of the ACM*, **12** (1969), 266-268.

[2] R.G. BLAND, "New finite pivoting rules for the simplex method", *Mathematics of Operations Research* **2** (1977), 103-107.

[3] G.B. DANTZIG, *Linear Programming and Extensions,* Princeton University Press, Princeton, New Jersey (1963).

[4] D.M. GAY, "On combining the schemes of Reid and Saunders for sparse LP bases", *Proceedings of the Symposium on Sparse Matrix Computations,* Knoxville, Tennessee, I.S. Duff and G.W. Stewart (eds), November 1978.

[5] D. GOLDFARB and J.K. REID, "A practicable steepest edge simplex algorithm", *Mathematical Programming* **12** (1977), 361-371.

[6] F. GUSTAVSON, "Finding the block lower triangular form of a sparse matrix", in *Sparse Matrix Computations,* J.R. Bunch and D.J. Rose (eds), Academic Press, New York, New York (1976), 275-289.

[7] P.M.J. HARRIS, "Pivot selection methods of the Devex LP code", *Mathematical Programming* **5** (1973) 1-28.

[8] E. HELLERMAN and D.C. RARICK, "Reinversion with the preassigned pivot procedure", *Mathematical Programming* **1** (1971), 195-216.

[9] E. HELLERMAN and D.C. RARICK, "The partitioned preassigned pivot procedure (P^4)", pp. 67-76 of *Sparse Matrices and Their Applications,* D.J. Rose and R.A. Willoughby, Plenum Press, New York (1972).

[10] H.M. MARKOWITZ, "The elimination form of the inverse and its application to linear programming", *Management Science* **3** (1957), 255-269.

[11] B.A. MURTAGH and M.A. SAUNDERS, "A large-scale nonlinear programming (for problems with linear constraints) --user's guide",

Technical Report SOL 77-9, Feb 1977, Department of Operations Research, Stanford University, Stanford, California.

[12] W. ORCHARD-HAYS, *Advanced Linear Programming Computing Techniques*, McGraw-Hill, New York, 1968.

[13] A.F. PEROLD and G.B. DANTZIG, "A basis factorization method for block triangular linear programs", *Proceedings of the Symposium on Sparse Matrix Computations,* Knoxville, Tennessee, I.S. Duff and G.W. Stewart (Eds), 1978.

[14] A.F. PEROLD, "Exploiting degeneracy in the simplex method", (1979), in preparation.

[15] J.K. REID, "A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases", Report CSS 20, Computer Science and Systems Division, AERE Harwell, Oxon., England (1975).

[16] J.K. REID, "Fortran subroutines for handling sparse linear programming bases", Report AERE - R8269, AERE Harwell, Oxon., England (1976).

[17] A. SANGIOVANNI-VINCENTELLI and T.A. BICKART, "Bipartite graphs and an optimal bordered triangular form of a matrix", Memorandum No. UCB/ERL M78/74, Electronics Research Laboratory, University of California at Berkeley, California, July 1977.

[18] M.A. SAUNDERS, "A fast, stable implementation of the simplex method using Bartels-Golub updating", in *Sparse Matrix Computations,* J.R. Bunch and D.J. Rose (eds), Academic Press, New York, New York (1976) 213-226.

[19] M.A. SAUNDERS, "MINOS System Manual", Technical Report SOL 77-31, December 1977, Department of Operations Research, Stanford University, Stanford, California.

CONTROLLING THE SIZE OF MINIKERNELS

Richard D. McBride

Finance and Business Economics Department
University of Southern California

In the bump triangular dynamic factorization algorithm the basis is partitioned in such a manner that the simplex method can be executed from a series of small inverses, called minikernels, and the basis itself. Methods are presented which can help control the size of the minikernels. One particular problem solved concerns the potential existence of bumps with a large number of spikes obtained from Hellerman and Rarick's P^4 procedure. Artificial inverses are used to keep the minikernels small in dimension.

INTRODUCTION

Recently, a method was published [5] which permits the simplex method to be executed from a series of minikernels or mini-inverses. The efficiency of the method depends on controlling the size of these minikernels. The method utilizes directly the block triangular structure of the basis induced by Hellerman and Rarick's [4] P^4 procedure. The spikes within each bump on the diagonal are moved to the right of the bump thereby inducing a triangular submatrix and an inverse (equal in dimension to the number of spikes within the bump) for each bump. In [5] procedures are presented which permit this partition to be maintained from one pivot to the next. Difficulty is encountered in this method in solving those few problems that have bumps containing a large number of spikes. Partitioning procedures are presented in this paper which can be used to reduce the size of the minikernels which result from bumps having a large number of spikes.

Section 2 develops the basic partitioned inverse and Section 3 presents the procedures that can be used to reduce the size of the minikernels.

2. DEVELOPMENT OF PARTITIONED INVERSE WITH MINIKERNELS

Consider the partitioned simplex basis after possible row and column interchanges

$$B = \begin{bmatrix} A_{11} & 0 \\ A_{21} & I \end{bmatrix} \qquad (1)$$

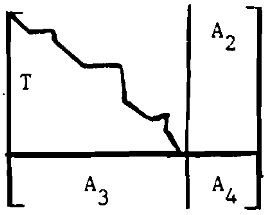
where the a-type columns correspond to the basic structural variables. The basis inverse corresponding to the partitioned basis (1) is

$$B^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \qquad (2)$$

A_{11}^{-1} is the essential part of B^{-1} and is called the kernel [2]. The kernel can be used as the working inverse in the simplex method. Although the dimension of A_{11} may be considerably smaller than that of B, a further significant reduction can be made by taking advantage of the block triangularity of A_{11} (after possible run and column interchanges) for large sparse LP problems.

After the application of the P^4 procedure [4], we get the following partition of A_{11}:

$$A_{11} = \begin{bmatrix} T & A_2 \\ A_3 & A_4 \end{bmatrix} =$$



(3)

where T is lower block triangular. The partitioned inverse (3) is:

$$A_{11}^{-1} = \begin{bmatrix} T^{-1} + T^{-1} A_2 H_0^{-1} A_3 T^{-1} & -T^{-1} A_2 H_0^{-1} \\ -H_0^{-1} A_3 T^{-1} & H_0^{-1} \end{bmatrix} \quad (4)$$

where $H_0 = A_4 - A_3 T^{-1} A_2$. H_0^{-1} is called either a subkernel [3] or a minikernel. The dimension of H_0^{-1} will usually be much smaller than the dimension of A_{11}^{-1}. Immediately after reinversion the dimension of H_0^{-1} is typically zero. The simplex method can be executed using H_0^{-1} and T^{-1}. Due to the bump triangularity of T all operations requiring the use of T^{-1} can be replaced by solving bump triangular systems of linear equations with T as the coefficient matrix.

The submatrix T can be partitioned

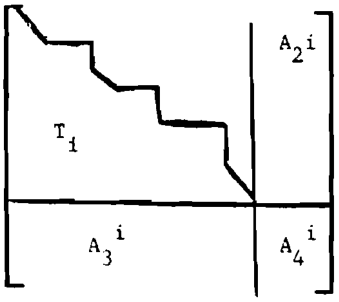
$$T = \begin{bmatrix} D_1 & & & & \\ & D_2 & & & \\ & & \ddots D_i & & \\ & & & \ddots & \\ F_1 & F_2 \cdots & F_i & \cdots & D_k \end{bmatrix}$$

The solutions of the triangular systems

$$Tx = a \qquad \text{or} \qquad x^T T = a^T$$

require that the subsystems $D_i x^i = \bar{a}^i$ and $(x^i)^T D_i = (\bar{a}^i)^T$ be solved. These subsystems can easily be solved if D_i is triangular. If D_i is not triangular then it is called a bump. If the P^4 procedure is repeatedly applied to the bump D_i after removal of spikes, then D_i typically takes the following form after row and column interchanges:

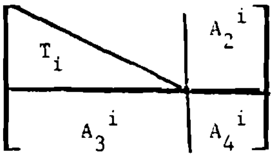
D_i =



(5)

The structure of D_i in (5) is the same as T in (3). If all of the spikes of D_i are moved to the right of D_i then D_i takes the following form:

D_i =



(6)

Experience indicates that the number of spikes in D_i is usually quite small. The partitioned inverse of each bump D_i has the same structure as (4) and yields the minikernel H_i^{-1}. The structure of D_i given in (6) is the structure of D_i implemented in [5]. As mentioned above, experience indicates that the number of spikes in D_i is usually small and therefore the dimension of H_i^{-1} is small. It is very common for the dimension of H_i^{-1} to range from one to eight.

To solve the subsystem $D_i x^i = \bar{a}^i$ one must compute

$$x^i = \begin{bmatrix} x_1^i \\ \\ x_2^i \end{bmatrix} = D_i^{-1} \begin{bmatrix} \bar{a}_1^i \\ \\ \bar{a}_2^i \end{bmatrix} = \begin{bmatrix} z + 1 \\ \\ s \end{bmatrix}$$

where

$$\begin{aligned} T_i z &= \bar{a}_1^i, \qquad & s &= H_i^{-1}(a_2^i - A_3^i z) \\ T_i 1 &= -A_2^i s, \qquad & H_i &= A_4^i - A_3^i T_i^{-1} A_2^i. \end{aligned} \tag{7}$$

When H_i^{-1} is available, we compute x^i by solving two triangular systems of equations involving T_i and performing some matrix arithmetic. The subsystem $(x^i)^T D_i = (\bar{a}^i)^T$ is solved in a similar manner.

Computational experience is given in [5] which illustrates the effectiveness of the use of minikernels in representing the basis inverse. One minikernel is required for each bump in T in addition to H_0^{-1}. Experience indicates at least a reduction of one third in the number of nonzero elements needed to represent the basis inverse when compared to Reid [6, 7] at the expense of a slight increase in computational time. The above techniques with D_i partitioned as in (6) works well for most problems. In the next section we discuss the partition that can be used when that occasional problem is encountered that contains a bump with a large number of spikes.

3. PARTITIONING BUMPS WITH LARGE NUMBERS OF SPIKES

When a bump is encountered with a large number of spikes the partition given in (6) will produce a minikernel large in dimension. In the PILOT1 energy model [1] it is common for a bump to be encountered with more than 100 spikes. A minikernel of dimension of more than 100 is not desirable. In this case partition (5) is preferred over (6). When using (5) one would solve bump triangular systems of linear equation in (7) rather than solving purely triangular systems.

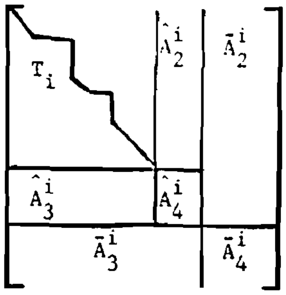
In a particular PILOT1 basis studied a bump was encountered with dimension 424 and 117 spikes. When the 21 tallest spikes are moved to the right of the bump, the bump decomposes into:

H_i Dimension	0	6	0	1	1	0	1	3	0	5	0	24	0
T_i Dimension	6	5	11	1	1	7	1	4	3	26	16	46	4
	0	0	6	0	3	0	6	0	2	1	0	1	0
	8	17	5	2	3	5	11	16	4	1	8	1	3
	2	5	17	0	0	1	0	1	0	1	0	1	
	7	12	42	1	1	1	3	1	3	1	7	1	

The bump decomposes into 38 subbumps with minikernels ranging in dimension from 1 to 24. In this case the space required to represent the bump (carry the minikernels) reduced from 13,689 to 1,566.

It is possible to use an iterated form of (5):

$D_i =$



$$= \begin{bmatrix} \bar{D}_i & \bar{A}_2^i \\ \bar{A}_3^i & \bar{A}_4^i \end{bmatrix} \qquad (8)$$

Here we have

$$\begin{aligned} \bar{H}_i &= \bar{A}_4^i - \bar{A}_3^i \, \bar{D}_i^{-1} \, \bar{A}_2^i \\ \hat{H}_i &= \hat{A}_4^i - \hat{A}_3^i \, T_i^{-1} \, \hat{A}_2^i \end{aligned} \qquad (9)$$

Carrying the partition in (8) to its extreme would permit representing D_i by a series of one by one minikernels equal in number to the number of spikes in D_i. However, this would require too much computational work to execute the simplex method.

The subsystem $D_i x^i = \bar{a}^i$ would be solved using (8) and the inverses of (9) as follows:

$$x^i = D_i^{-1} \begin{bmatrix} \bar{a}_1^i \\ \\ \bar{a}_z^i \end{bmatrix} = \begin{bmatrix} \bar{z} + \bar{l} \\ \\ \bar{s} \end{bmatrix}$$

where

$$\bar{D}_i\bar{z} = \bar{a}_1^i \,, \qquad \bar{s} = \bar{H}_i^{-1}\,(\bar{a}_2^i - \bar{A}_3^i \;\; \bar{z})$$

$$\bar{D}_i \; \bar{l} = \bar{A}_2^i \quad \bar{s} \overset{\Delta}{=} \bar{b}_1^i \quad = \begin{bmatrix} \bar{b}_{11}^i \\ \\ \bar{b}_{12}^i \end{bmatrix} \tag{10}$$

and

$$\bar{z} = \bar{D}_i^{-1}\;\bar{a}_1^i = \begin{bmatrix} \hat{z}_1 + \hat{l}_1 \\ \\ \hat{s}_1 \end{bmatrix}, \quad \begin{array}{l} \hat{T}_i\;\hat{z}_1 = \bar{a}_{11}^i \,, \qquad \hat{s}_1 = \hat{H}_i^{-1}\,(\bar{a}_{12}^i \;-\; \hat{A}_3^i\;\;\hat{z}_1) \\ \hat{T}_i\;\hat{l}_1 = -\hat{A}_2^i\;\;\hat{s}_1 \end{array}$$

and

$$\bar{l} = \bar{D}_i^{-1}\;\bar{b}_1^i = \begin{bmatrix} \hat{z}_2 + \hat{l}_2 \\ \\ \hat{s}_2 \end{bmatrix}, \quad \begin{array}{l} \hat{T}_i\;\hat{z}_2 = \bar{b}_{11}^i \quad, \qquad \hat{s}_2 = \hat{H}_i^{-1}\,(\bar{b}_{12}^i \;-\; \hat{A}_3^i\;\;\hat{z}_2) \\ \hat{T}_i\;\hat{l}_2 = -\hat{A}_2^i\;\;\hat{s}_2 \quad . \end{array}$$

In computing $\bar{z}$ and $\bar{l}$ notice that $\hat{z}_1$ and $\hat{z}_2$ can both be computed in the same phase through the columns of $\hat{T}_i$. The same is also true for $\hat{l}_1$ and $\hat{l}_2$. Note that the net effect of partitioning the spikes on the right in D_i requires no additional passes through the columns of the bump triangular submatrix $\hat{T}_i$. Using this iterated strategy it is possible to replace a 2n x 2n by two n x n minikernels with a resultant 50% reduction in memory requirements.

4. CONCLUSION

Bumps with a large number of spikes can be efficiently handled by a repeated application of the basic partitioning strategy. It is possible to further reduce the size of minikernels by an iterated application of the basic partitioning scheme which yields a further 50% reduction in memory requirements. Clearly, the space required to represent the basis inverse (in addition to the basis itself) can be reduced to equal the number of spikes in the kernel. The practitioner must choose that level of partitioning to obtain the fine balance between his particular memory and execution time requirements.

REFERENCES

[1] Dantzy, G.B., The tests were performed using PILOT1, a SIGMA version of Dantzy's energy model. Systems Optimization Laboratory, Department of Operations Research, Stanford University.

[2] Graves, G.W., "A complete constructive algorithm for the general mixed linear programming problem," Naval Research Logistics Quarterly. 12 (1965), 1-34.

[3] Graves, G.W. and R.D. McBride, "The factorization approach to large-scale linear programming," Mathematical Programming, 10 (1976), 91110.

[4] Hellerman, E., and D. Rarick, "The Partitioned Preassigned Pivot Procedure (P^4)," in: D.J. Rose and P.A. Willoughby, eds., Sparse Matrices and Their Applications (Plenum Press, New York, 1972), pp. 67-76.

[5] McBride, R.D., "A bump triangular dynamics factorization algorithm for the simplex method," Mathematical Programming, 18 (1980), 49-61.

[6] Reid, J.K., "A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases," Report CSS 20, Computer Science and Systems Division, A.E.R.E., Harwell, England (1975).

[7] Reid, J.K., "FORTRAN subroutines for handling linear programming basis," Report AERE-R 8269, Computer Science and Systems Division, A.E.R.E., Harwell, England (1976).

ALGORITHMS FOR BLOCK TRIANGULARIZATION OF BASIS MATRICES AND EXPLOITATION OF DUAL DEGENERACY IN THE DUAL SIMPLEX METHOD

Eugeniusz Toczyłowski

Institute of Automatic Control
Technical University of Warsaw

This paper studies two topics essential to large-scale linear programming. First, algorithms used in restructuring a basis matrix to create a sparse representation of the inverse are considered. We will show that the best strategy for block triangularization of a basis matrix is in general *not* the execution in sequence of a maximum matching algorithm and an algorithm for finding the strong components of the directed graph associated with the basis matrix, but rather the multiple execution of an algorithm for finding the strong components of appropriate directed graphs as a subroutine in an algorithm for finding a maximum matching. We also present a new algorithm for finding a maximum matching based on the Hopcroft and Karp approach. In the second part of the paper we present a modification of the dual simplex algorithm efficient in the case of the dual degeneracy typically found with integer programming algorithms.

1. NOTATION

The LP problem is

$$\max \; x_1$$

$$(1) \qquad Ax = b$$

$$l \leqslant x \leqslant u$$

where $x=(x_1,\dots,x_n)$, $A=[a_1,a_2,\dots,a_n]$ is $m \times n$ matrix with columns a_j, and $-\infty \leqslant l_i \leqslant u_i \leqslant +\infty$, $i=1,\dots,n$. Let $\mathcal{N}$ be the index set for the nonbasis variables, and $\beta=\{\beta_1,\dots,\beta_m\}$ be the index set for the basis variables. The system $Ax = b$ can be written in the form

(2) $$Bx_\beta + \sum_{j\in\mathcal{N}} a_j x_j = b$$

where $B = [a_{\beta_1}, \dots, a_{\beta_m}]$ is the basis matrix and x_β the vector of basic variables, or in the form

(3) $$x_\beta + \sum_{j\in\mathcal{N}} B^{-1} a_j x_j = B^{-1} b$$

If a basis matrix B is reinverted, permutation matrices P and Q are required such that PBQ is in block triangular form with diagonal blocks having bordered band lower triangular structure.

Let us denote by $p_i(q_i)$ row (column) index of the i-th row (column) of the matrix PBQ. Permutation matrix P with elements P(i,j) is equivalent to the vextor $p=(p_1,\dots,p_m)$ by the equivalence relationship $P \Leftrightarrow p$ iff $P(p_i,i)=1$. Analogously $Q \Leftrightarrow q$ iff $Q(i,q_i) = 1$.

We associate with an m-square matrix $B = [b_{ij}]$ a directed graph $G(B)$ which consists of a set of m vertices $\{1,2,\dots,m\}$ and a set of arcs $\{(i,j): b_{ji} \neq 0\}$.

2 . ALGORITHMS FOR RESTRUCTURING BASIS MATRICES.

Block triangularizing of a basis matrix B can be done in two stages. The first stage is finding a maximum matching (or maximum transversal), the second stage is finding the strong components of the directed graph associated with the matrix B.

2.1. An algorithm for maximum matching.

Most algorithms for finding a maximum matching are based on one devised by Hall [4]. These algorithms are of Complexity O(m.t) where m is the number of rows and t is the number of non-zeros in the matrix. The Hopcroft and Karp algorithm [5] allows to simultaneously stretch an assignment with several paths of minimal lenght, and thus is of complexity $O(m^{1/2}.t)$.

Since this complexity orders are obtained from a worst-case analysis, there is no evidency which algorithm is more efficient in the typical performance. There is, however, very little published work on comparing these algorithms. In such analysis randomly generated matrices have often been used, though basis matrices from real life LP problems are not random in structure. In a recent paper of Darby - Dowman and Mitra [6] an interesting comparison of two versions of each of the algorithms have been done based on the analysis of a set of medium-size practical problems with the number of rows in the bumps ranging from 75 to 435 and with a comparable sparsity. Their conclusion was that Hopcroft and Karp algorithm for finding a maximum matching compares unfavourably with the algorithms based on Hall´s method . We have studied, however, some characteristics that are important in this algorithms. The most important indicators of efficiency of the algorithms in addition to the overal CP time are:

(i) the number of iterations in function of the number of rows. By one iteration we mean in Hall algorithm a nontrivial assignment with reasignments in an augmenting path, while in Hopcroft and Karp algorithm - forming a graph containing the set of all augmenting paths of shortest lenght and performing a set of reassignments resulting from the set of shortest augmenting paths.

(ii) the average CP time per iteration in function of the number of rows.

Comparison of this characteristics are given in Fig.1 and Fig.2. Figure 1a indicates that in Hall algorithm the number of nontrivial assignments, though drastically smaller than the number of rows, increases lineary with the number of rows. Figure 1b shows that the analogous characteristic for Hopcroft and Karp algorithm is a slower growing function (approximately a square-root function).

Figure 2a and 2b indicate, that the average CP time per iteration increases approximately lineary as the number of rows increases. Thus, the Hopcroft and Karp algorithm tends,

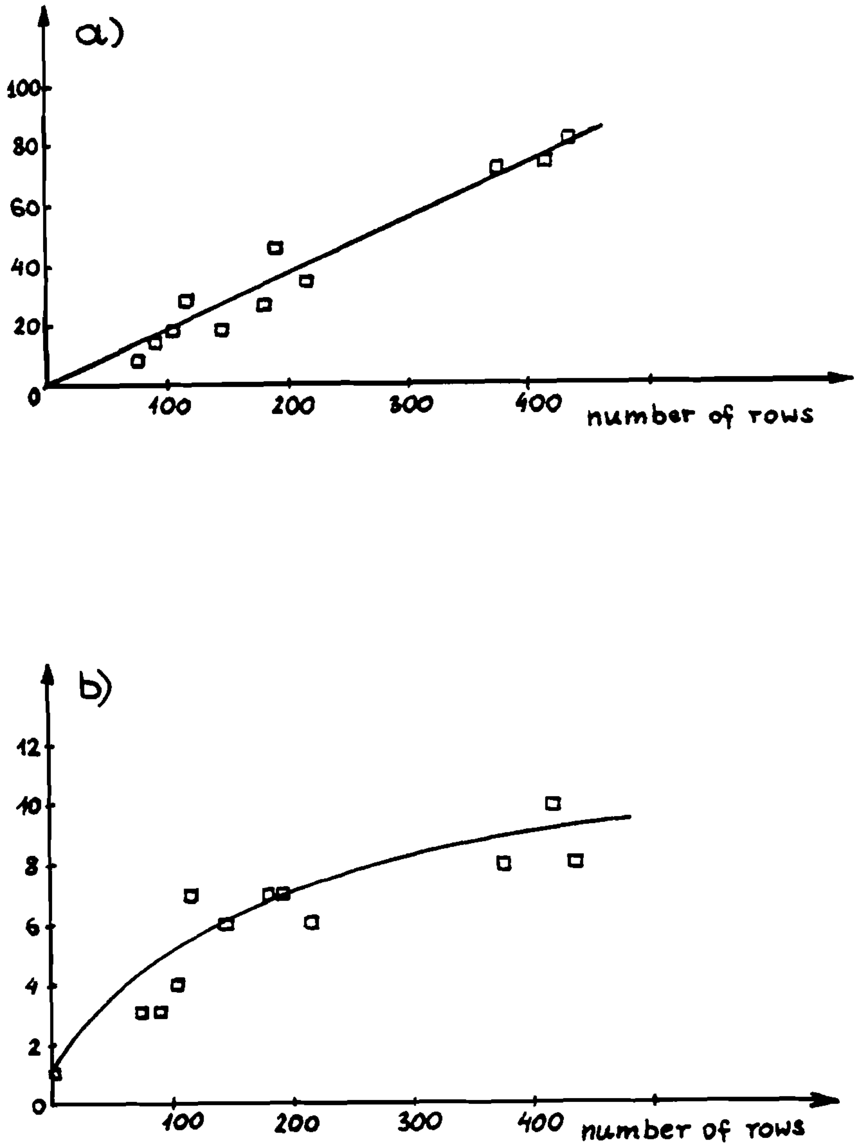


Fig.1. Number of iterations in a) Hall,
b) Hopcroft and Karp algorithms.

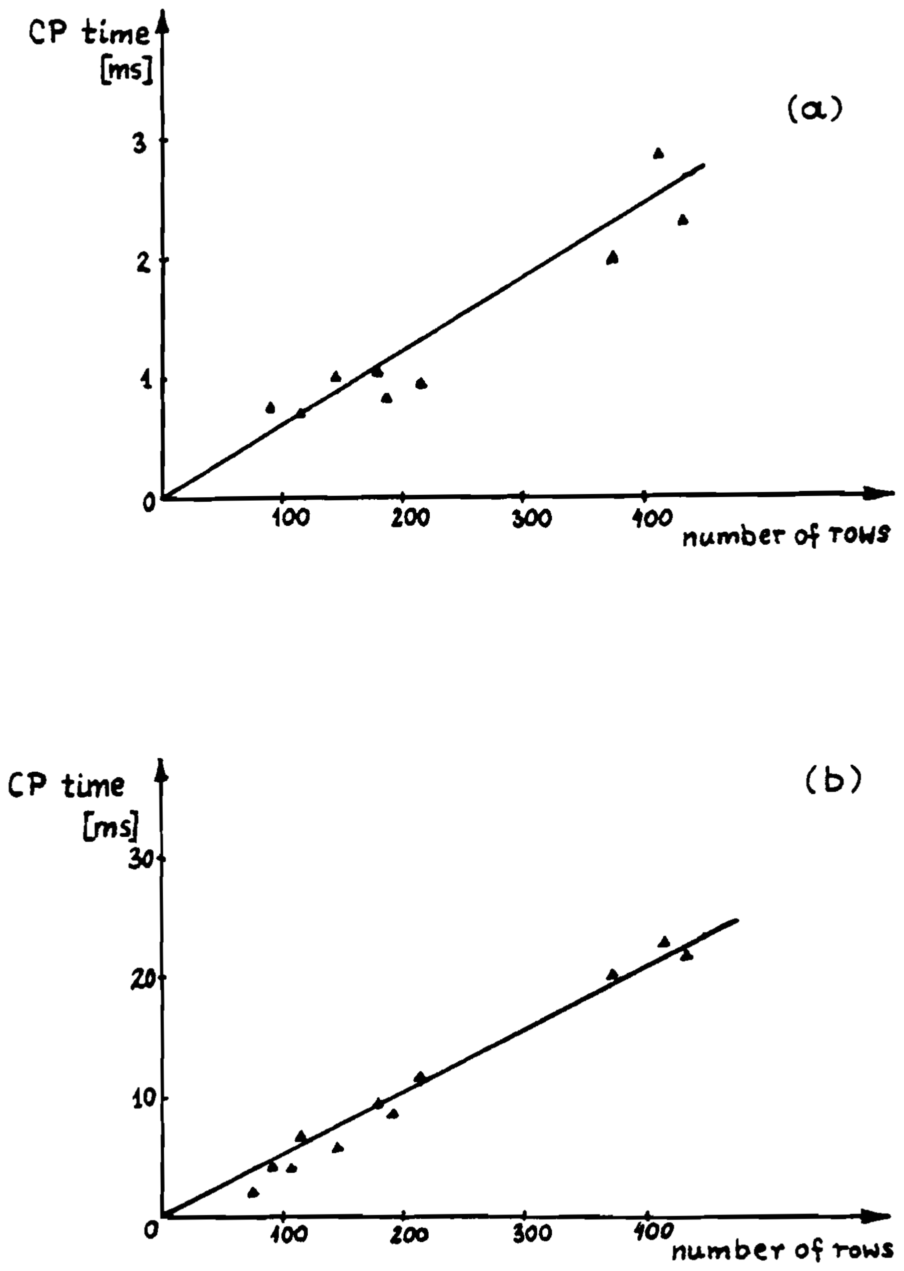


Fig.2. The average CP time per iteration in a) Hall, b) Hopcroft and Karp algorithms.

to be favorable with the Hall algorithm if the size of matrix increases.

The minimal size of matrices for which the H-K algorithm is more efficient than the Hall algorithm depends on the sparsity of the matrix. It probably tends to decrease as the average number of nonzeros per one row decreases. In this paper we present an improvement of H-K algorithm. We use an observation that an appropriate modification of graphs formed by H-K algorithm can considerably increase the number of assignments found per graph and thus reduce the number of H-K iterations. The idea of the algorithm is to form a graph containing larger set of augmenting paths, not only of shortest lenght. This can be done as follows (the notation and definitions used here may be found in [5]) Let $X = \{1,\ldots,m\}$ be the set of rows, and $Y = \{1,\ldots, m\}$ be the set of columns of the square matrix $B = [b_{ij}]$. Then we form the bipartite graph $G = (V,E)$ with vertex set V containing X and Y, and the edge set E such that each edge of G joins a vertex coresponding to row i in X with a vertex corresponding to column j in Y if and only if $b_{ij} \neq 0$. A set $M \subseteq E$ is a matching if there is no vertex $u \in V$ incident with more than one edge in M. A matching of maximum cardinality is called a maximum matching. A vertex $v \in V$ is free if it is incident with no edge in M. A path (without repeated vertices)

$$P = (v_1, v_2), \ (v_2, v_3), \ldots, (v_{2k-1}, v_{2k})$$

is called an augmenting path if its endpoints v_1 and v_{2k} are both free and its edges are alternatively in $E \setminus M$ and in M. It is easy to verify [5], that if M is a matching and $P_1,\ldots,P_t$ are vertex disjoint augmenting paths relative to M, then

$\bar{M} = M \oplus P_1 \oplus P_2 \oplus \ldots \oplus P_t$, (where $\oplus$ denotes the symmetric difference) is a matching, and $|\bar{M}| = |M| + t$.

Now we discuss how to find a maximal vertex-disjont set of augmenting paths $P_1,\ldots,P_t$ relative to M. First we assign directions to the edges of G in such a way that augmenting paths relative to M become directed paths.

This is done by directing each edge in M so that it runs from a row to a column and each edge in E-M so that it runs from a column to a row. The resulting directed graph is denoted by $\bar{G} = (V,\bar{E})$. Now assume, that the graph $\bar{G}$ contains strongly connected components $\bar{G}_i = (V_i,\bar{E}_i)$, $i=1,\dots,K$. Then the edges of $\bar{G}$ fall into 2 classes.

(i) some are edges joining vertices of the same component

(ii) other join vertices of different components. These are called cross-links.

Theorem [10]. If $N \subset E$ is a maximum matching in G, then N does not contain cross-links of $\bar{G}$.

Since the finding of the strongly connected components of G can be done by the depth - first search algorithm of Tarjan in $O(|E|)$ space and time, the elimination of cross-links from consideration may increase the efficiency of the matching algorithm in the case of very large and sparse matrices.

Assume, that in block triangularizing of a basis matrix the Tarjan algorithm for finding the strong components of the directed graph associated with the basis matrix is used repeatedly with the maximum matching algorithm after performing, say, k iterations of the matching algorithm. The efficiency of the block triangularizing algorithm evidently depends on k. The optimal value of k depends on such parameters of the basis matrices as the number of rows and the average number of nonzeros per one row. The most desired strategy must be obtained empirically.

In the remaining part of this paragraph we will present a modyfication of Hopcroft and Karp algorithm for maximum matching. In one iteration of the modified algorithm the graph $\hat{G} = (\hat{V},\hat{E})$ containing larger set of augmenting paths $P_1,\dots,P_t$ is formulated and then the new matching $\bar{M}$ is defined by $\bar{M} = M \oplus P_1 \oplus P_2 \ .. \oplus P_t$.

Let $\bar{E}_0 := \{(y,x):(y,x) \in E_i$ for some $i\}$.

Now we extract from graph $(V,\bar{E}_0)$ a subgraph $\hat{G}$ with the property that the directed path of $\hat{G}$ running from a free column to a free row correspond one-to-one to an augmenting path in G relative to M. This is done as follows.

Let L_0 be the set of free rows, and let

$$L_i^0 = L_i \wedge \{ \text{free columns} \}$$
$$L_i = L_i \setminus L_i^0$$
$$E_i = \{ (u,v) ; (u,v) \in \overline{E}_0 , v \in L_i^1 , u \notin L_0 \cup L_1 \ldots \cup L_i \}$$
$$L_{i+1} = \{ u : \text{for some } v , (u,v) \in E_i \}$$

for $i = 0,1,2,\ldots$

Let $i^* = \max \{ i : L_i^0 \neq 0 \}$

Then we define the graph $\hat{G} = (\hat{V},\hat{E})$, where

$$\hat{V} = L_0 \vee L_1 \cup \ldots \cup L_{i^*-1} \cup L^0_{i^*}$$

$$E = E_0 \cup E_1 \cup \ldots \cup E_{i^*-1} \wedge (L^0_{i^*} \times L_{i^*-1})$$

The graph $\hat{G} = (\hat{V},\hat{E})$ in comparison to the graph formed by the original Hopcroft and Karp algorithm ([5], p.229) has the following properties:

(i) the graph formed by H-K algorithm containts only the shortest augmenting paths relative to M and is a subgraph of $\hat{G}$

(ii) $\hat{G}$,contains also augmenting paths relative to M of greater lenght.

An algorithm for finding a maximal vertex-disjoint set of paths is given in [5]. For our purpose we should order the set of free columns in such a way that the algorithm will find first the shortest augmenting paths and then the augmenting paths with increasing length.

There are three characteristics of the presented algorithm for maximum matching,important for large-scale graphs: (a) storage requirements, (b) CP time per one iteration, (c) number of iterations. In the absence of actual implementation, the following analysis will be somewhat superficial.

(a) Storage requirements. Since $\hat{G}$ contains at most all vertices of G, storage requirements in all steps of the algorithm are linear in number of vertices and edges. From numerical experience, the subgraph of $\hat{G}$ formed by

H-K algorithm contains typically from 60 to 80 percentages of vertices of G and this percentage is vertex-size independent. Thus $\hat{G}$ may contain typically at most 10 to 40 percent more vertices of G than H-K graph.

(b) CP time per one iteration. Complexity is linear in number of vertices and edges. Time is increased in comparison to H-K algorithm by a factor similar to (a)

(c) Number of iterations in comparison to H-K algorithm may be considerably decreased. This supposition folllows from handy-made analysis of small problems and from analysis of the characteristic of H-K algorithm presented in Fig.3.

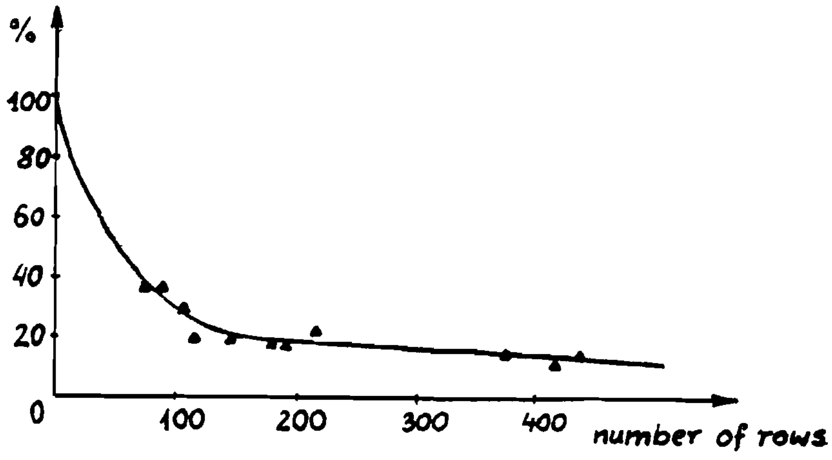


Fig.3. The part of the graph vertices belonging to vertex disjoint augmenting paths.

The figure shows the size of subgraph of $\bar{G}$ which contains all shortest vertex disjoint augmenting paths. This subgraph of $\bar{G}$ contains only small fraction of vertices of $\bar{G}$

and moreover, this fraction is decreasing with the size of problems. Thus, after removing from $\hat{G}$ all shortest vertex disjoint augmenting paths there remains a considerable part of $\hat{G}$ containing augmenting paths of greater lenght.

We conjecture that the number of iterations of the presented algorithm is a slower growing function of the size of the problems in comparison to H-K algorithm.

2.2. An algorithm for refinding lower block triangular structure of an updated basis matrix.

It is now accepted that the most efficient algorithm for finding strongly connected components of a directed graph is due to Tarjan [9]. However, if the basis matrix is updated, the algorithm can be modified to enable performing the search in a restricted part of the graph of the updated basis. The need for such an algorithm results from the possibility of using an additional rule in multiple pricing which will result in producing at each iteration basis matrices with the simplest "bumb and spike" structure.

Let us assume that the r-th column of the basis matrix B is replaced by a column a_k, $k \in \mathcal{N}$, The bump structure of the updated basis matrix may be created by the following modification of Tarjan algorithm, in which the depth-first search is restricted to a part of the updated basis. From the previous step we will use the following information: for each row i there is known its bumb number S(i), where

$$S(i) = \min_t \{ p(t) : t \text{ and } i \text{ lie in the same bump} \}$$

Algorithm

1^0 Compute $\mu_k = \min_i \{ S(i) : a_{ik} \neq 0 \}$ and $M_k = S(r)$

If $a_{rk} \neq 0$ go to 2^0, otherwise denominate a_k as the "free" column and row r_k as the "free" row, and find an augmenting path in the graph of the basis matrix leading from the free column to the free row using the depth-first search restricted to the rows with p(i) such that $\mu_k \leq p(i) \leq M_k$. Make the reasignment of the rows and columns belonging to the augmenting path .GO to 2^0

2° In the subgraph of the basis matrix, containing vertices with row indices $p(i)$ such that $\mu_k \leq p(i) \leq M_k$ perform the algorithm of Tarjan that finds the strongly connected components of the subgraph.

The complexity of the above algorithm is $O(t)$, where t is the number of nonzero elements q_{ij} of the updated basis matrix with row and column indices lying between μ_k and M_k.

Suppose that in multiple pricing we have a set of columns a_j, $j \in K$ and we want to choose the column a_k giving the simplest bump structure of the updated basis matrix. If the precise algorithm for updating the bump structure is too expensive, the following suboptimizing criterion may be used:

From the set of columns a_j, $j \in K$ select a column a_k such that

$$\Delta_k = \min_{j \in K} \Delta_j$$

where $\Delta_j = S(r_j) - \min_i \{S(i): a_{ij} \neq 0\}$

and r_j is the index of the column,leaving the basis B after entering a_j to the basis.

3. EXPLOITING DUAL DEGENERACY IN THE DUAL SIMPLEX ALGORITHM.

In some integer programming algorithms the "power" of the succeeding iterations is usually hampered by the massive degeneracy and/or the severe round-off errors. This occurs in the cutting plane algorithm of integer forms as well as in the composite integer algorithm having cuts incorporated into the branch-and-bound scheme. In this section we discuss a technique presented in [11] that allewiates this difficulty.

3.1. A modified dual LP algorithm

Though cycling resulting from degeneracy is not so serious a problem in practice (it may be prevented by the use of a perturbation scheme such as lexicographic ordering of row or column vectors or by choosing the smallest index or any other handy rule preventing cycling at least from empirical evidence) there remains the related problem of slow convergence caused by many iterations with zero changes of the objective function. The difference between degenerate and nondegenerate iterations in the LP algorithms results from the fact, that in the absence of degeneracy the simplex algorithm has the stepest descent property, while in the presence of degeneracy the lexicographic ordering assures only finitness of the iterations.

To reduce the number of degenerate simplex iterations we have incorporated into the dual simplex algorithm a mechanizm which assures the stepest descent property of the algorithm also in the case of severe degeneracy.

Let us consider the LP problem (1,2) and assume, that the basis is:

(i) dual feasible, i.e. $\alpha_{1j} := e_1^T B^{-1} a_j \geqslant 0$, $j \in \mathcal{N}$.

(ii) primal infeasible, i.e. there is nonempty set T of negative basic variables x_{β_i}, $i \in T$ (for simplicity we assume, that $l_i = 0$, $u_i = +\infty$).

We also assume, that the basis is dual degenerate, i.e. the set $\mathcal{N}_0 = \{j:\ \alpha_{1j} = 0\}$ of degenerate nonbasic variables is nonempty. In the dual simplex method moving from one degenerate basic solution to another is indeed solving the totally degenerate subproblem

$$\max \quad x_{\beta_1}$$

$$(4) \qquad x_\beta + \sum_{j \in \mathcal{N}_0} (B^{-1} a_j) x_j = B^{-1} b$$

$$x_\beta ,\ x_j \geqslant 0$$

which has primal infeasible basis B. In order to solve this subproblem we can maximize an auxiliary function

$w = \sum_{i \in T} x_{\beta_i}$ which measures the primal infeasibility of (4) as in the first phase of the Orchard-Hays composite simplex algorithm [8] . After solving (4) the new basis is updated according to the ordinary rules of the dual LP algorithm. Our experience shows that this modification significantly improves the performance of the dual LP algorithm in

the case when the number of nonfeasible basic variables is equal one, as it occurs in the Gomory's cutting plane algorithm after adding a new cut or in the branch-and-bound algorithm after branching to a new vertex. It follows from the fact that in this case, once (4) has been solved, a change in the basis occurs with a simultaneous exchange of dual degenerate and nondegenerate variables.
Now we will present one iteration of the algorithm under assumption that the dual LP algorithm uses the same data format as the primal algorithm.
Inially set $\theta := -\infty$, k:=0 and select $r \in T$.
Algorithm (one iteration)

1° Execute backward transformation routine, compute the pricing forms

$$\pi_1 = e_1^T B^{-1}$$
$$\pi_r = e_r^T B^{-1}$$
$$\pi_w = d^T B^{-1}, \text{ where } d = \sum_{i \in T} e_i$$

Go to 2°

2° If there exists a nonbasic column a_j, $j \in \mathcal{N}$ not considered yet, compute $\alpha_{1j} := \pi_1 \cdot \alpha_j$ and go to 3°; otherwise set j:= k and go to 5°.

3° If $\alpha_{1j} > 0$ then go to 4°. Otherwise compute $\alpha_{wj} := \pi_w \cdot a_j$. If $\alpha_{wj} > 0$ go to 4°. Otherwise select the basic variable x_{β_t} reaching first its bound after entering x_j to the basis (this is pivot selection rule of the primal simplex algorithm); r: = t , Go to 5°

4° Compute $\alpha_{rj} := \pi_r \cdot \alpha_j$; If $\alpha_{rj} < 0$ and $\theta < \frac{\alpha_{1j}}{\alpha_{rj}}$ then set $\theta := \frac{\alpha_{1j}}{\alpha_{rj}}$ and k: = j. (this is pivot selection of the dual algorithm). Go to 2°.

5° If j=0, LP is not feasible. Otherwise update the basis with the pivot pair (r,j). This involves creation new η and solution columns.

3.2. Computational results.

The modified dual algorithm has been tested by solving the set of test integer programming problems, relatively difficult to solve by cutting plane method. The problems have been choosen from [2] and from [7] .

The following algorithms have been compared.

LIP1 - a version of the method of integer forms developed by Haldi and Isaacson [3] known as LIP 1

KAL - a version of the method of integer forms developed by Kaliszewski [7]

TO - the method of integer forms for ILP with bounded variables in the all-integer floating-point representation, with the basic dual simplex algorithm, and a source row selection that yields the largest decrease in the objective function

TO-M - the method TO with two modifications: dual simplex algorithm is replaced by the modified dual algorithm described in section 3.1 and an additional source row selection (criterion 3 in [2] p. 165) which breaks the ties in the source row selection in the algorithm TO

The results of the algorithms LIP1 and KAL were reported in [2], p.380 and in [7] . Computations of the algorithms TO and TO-M were performed on the computer Odra 1325. The results are in Table 1

Table 1

Test problem	Algorithm	Cuts	Simplex iterations	Simplex iterations per cut
Haldi 5	LIP1	>	>	
	TO	>	>	
	TO-M	176	222	1.2
Haldi 6	LIP1	123	?	
	TO	49	121	2.8
	TO-M	41	69	1.6
Haldi 9	LIP1	42	?	
	TO-M	8	14	1
IBM 9 (m=35,n=40)	LIP1	953	?	
	TO	>	>	
	TO-M	22	155	6
TO-20	KAL	40	146	3.5
	TO	15	69	4.1
	TO-M	1	7	1
TA-20	KAL	4	18	3
	TO	1	7	1
	TO-M	1	7	1
0-19	KAL	>160	>613	3.8
	TO-M	17	39	1.8
0-17	KAL	> 190	> 518	2.7
	TO-M	47	80	1.6

The comparison of the algorithms indicate that:

(i) the choise of the cuts leading to the severe dual degeneracy in the algorithm TO-M is advantageous from the efficiency point of view.

(ii) the use of the modified dual simplex algorithm reduces the average number of simplex iterations per one cut.

References

1. Bisschop,J.Levy,Y.and Meeraus,A. "Programs for structure analysis of sparse matrices" Development Research Center, World Bank, Washington, 1979.
2. Garfinkel,R.S.,Nemhauser,G.L.,Integer Programming" John Wiley Sons, 1972.
3. Haldi,J., and L.M.Isaacson,Opns,Res. 13, 964-959,1965
4. Hall,M."An algorithm for distinct representations". Amer.Math.Monthly, Vol 63 pp. 716-717, 1956.
5. Hopcroft,J.E. and Karp, R.M. "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs SIAM J. Comput, Vol.2,No 4,pp.225-231, 1973.
6 Darby-Dowman,K.and Mitra,G. "An investigation of algorithms used in the restructuring of linear programming basis matrices prior to inversion" Brunel Univ. STR/33 Report, 1979.
7. Kaliszewski.I., "Efficiency of the cutting plane method in integer programming" Thesis, IBS PAN, Warsaw, 1978 (in Polish).
8. Orchard-Hays,W."Advanced linear programming computing techniques" MCGraw-Hill, 1968.
9. Tarjan,R.E."Depth first search and linear graph algorithms" SIAM J. Comput.Vol 1.pp. 146-160, 1972.
10. Taha H.A."Integer Programming", Academic Press, 1975.
11. Toczyłowski, E. "On algorithms for maximum matching" submitted to Discrete Applied Mathematics.
12. Toczyłowski,E "A composite integer linear programming algorithm" X Symp.on Math.Programming,Montreal, 1979.

THE SIMPLEX METHOD FOR DYNAMIC AND BLOCK-ANGULAR LINEAR PROGRAMS

A RESOURCE-DIRECTIVE BASIS DECOMPOSITION ALGORITHM FOR WEAKLY COUPLED DYNAMIC LINEAR PROGRAMS*

Tatsuo Aonuma

Kobe University of Commerce

This paper presents a decomposition algorithm for dual angular linear programs, which also can be extended to a wider class of structured linear programs. The method is closely related to the algorithm by Martin Beale on the parameterization of the linking variables. In the algorithm, the linking variables are first fixed at given values to partition the problem into several subproblems. Secondly an optimal setting of the linking variables is determined, given that the bases for the subproblems are fixed. Then, the bases for the subproblems are changed so as to improve the entire problem. The computational experience indicates that the number of cycles to adjust the linking variables required for optimality is nearly equal to, or less than the number of the subproblems, and is smaller than the earlier computational results in the column-generation scheme, and that the computing time is much faster than in the direct simplex approach.

*This is a revised version of the paper presented at the XXIV International Meeting of the Institute of Management Sciences, June 18–22, 1979, in Hawaii.

Introduction. A two-level algorithm for two-stage linear programs has been presented in Aonuma [2]. The algorithm was developed with an intention of solving the two-stage linear programs arising from a nested approach to multi-period planning [1], in a manner of interactive preference optimization for considering uncertainty in the future. In the present paper we extend the same decomposition approach to a wider class of structured linear programs, especially to dual angular linear programs and also report computational experience in using it for weakly coupled dynamic linear programs.

The dual angular linear program we address is written as follows:

$$\text{Max} \quad \sum_{i=1}^{K} c^i x^i + c_Y y \qquad (0.1)$$

$$\text{s.t.} \quad A^i x^i + A_Y^i y = b^i \qquad (0.2)$$

$$x^i , y \geqq 0 \quad (i=1,2, , , ,K) \qquad (0.3)$$

We call y the linking variables and A_Y^i the linking matrix for the i-th block. In a dynamic linear program we get together all of the linking variables between two consecutive periods into one block.

Our decomposition method is closely related to Beale's approach [6] on the parametrization of the linking variables and is not of column-generation scheme. We begin by choosing initial values for the linking variables, and then the problem is decomposed into several subproblems when the y-variables are fixed. After optimizing these subproblems, the optimal setting of the linking variables are determined, given that the bases for the subproblems are fixed. For this purpose, we solve a coordination problem. Subsequently, a direction-finding problem for every non-optimal subproblem is solved for the purpose of exchanging the basis so as to improve the entire problem. We call the process "coordination" of the y-variables. The coordination process terminates when there is no improving the bases for the subproblems. In a sense of planning process [12], this type of coordination is considered to be resource-directive [8] and of two-level.

The time-consuming jobs throughout the whole computation in the algorithm are solving the subproblems at the initial stage and solving the coordination problems during the coordination process. We call the number of times of solving the coordination problem *the number of coordination cycles.* A built-in linear programming subroutine is required for solving both the subproblems and the coordination problems. As the number of rows of the coordination problem is equal to that of the linking variables, the largest problem to be solved by the subroutine can be the coordination problem in such a case that the number of periods, K, in a dynamic case is very large. That is one of the reasons why weakly coupled dynamic models are computationally preferable and effective for our algorithm for the purpose of solving much larger-scale models, where "weakly coupled" implies that the number of the linking variables between two consecutive periods is relatively small. The second reason is that it is possible for us to estimate "good" initial values for the linking variables in weakly coupled cases. The computational experience indicates that a good setting of the y-variables makes the algorithm work effectively.

An experimental code, named MULPS, has been written in FORTRAN for HITAC 8250 in order to solve dynamic linear programs having up to 180 rows and 6 periods. The SEXOP developed by R.E.Marsten [15] is used in the MULPS as an LP subroutine for solving the linear programs.

In the present experiments we mainly focus on the number of coordination cycles. From our experiments the number of the cycles seems to be nearly equal to, or less than the number of periods, and seems to be very small in comparison with thatin the earlier algorithms of column-generation scheme. For comparison with a direct simplex approach we tentatively convert the MULPS to a new large computer, FACOM M-160S(comparable to IBM 370/148), which has virtual storage in its operating system, and we use a version of the original SEXOP [15] as the FORTRAN linear programming code for the direct method. We observe that, for two test problems of 6 periods, the CPU computing time by the MULPS is about a quarter of those by the direct method, and that only a half of storage in the direct method is required in the MULPS.

It has been lately suggested by several researchers that our method is closely related to Gass' dualplex method [18] and Winkler's method [19]. We shall describe in [20] that there are three computationally different points from Gass' method. We happened to report before in [2] that the number of cycles required for optimality in our algorithm was less than that when employing a selection rule of Gass' type to obtain an improved basis of the subproblem in the 2-stage case, where we compared with the Beale's rule [6] which is also the same as the Gass' rule. And also, we can understand that our algorithm gives a concrete optimal strategy to Winkler's framework. However, our algorithm will be regarded as a coordination method rather than as a simplex method for large-scale problems.

Section 1 presents some methods on parametrization and transformation in linear programs, which will basically give a solution method to the coordination problem. In Section 2 the decomposition algorithm is presented for a simplified form of our problem above. The justification of the algorithm is shown in a constructive manner with several theorems and its finite convergence is also proved. At the end of the section the computational procedure is summarized. Section 3 contains the computational experience.

1. Parametrization and Transformation in Linear Programs

Consider the linear program

$$
\begin{array}{lll}
LP[B:y] & & \max\ c_Y(B)y \\
 & \text{s.t.} & Ix_B + A_Y(B)y = b(B) \\
 & & x_B,\ y \geq 0
\end{array}
$$

where I is a suitable identiy, $A_Y(B)$ is an $m \times n_y$, and the other vectors are of conformable dimensions. LP[B:y] represents the cannonical form of a linear program with respect to a given basis B and is the analogus notation adopted in Marsten and Shepardson [16] for expressing conveniently a linear program updated with respect to a given basis.

For the purpose of formulating the resource-directive coordination process in our two-level algorithm we consider a transformed linear program derived from LP[B:y]. Suppose that, at first, the y-variables are fixed at given values, y^0, and then they are adjusted through "new parameters", λ, around y^0. We have the following transformed problem:

$$
\begin{array}{llll}
LP_\lambda[B:y=y^0+I\lambda\ ] & \max & c_Y(B)\lambda + c_Y(B)y^0 & (1.1) \\
 & \text{s.t.} & Ix_B + A_Y(B)\lambda = x_B(y^0) & (1.2) \\
 & & -I\lambda + Iy = y^0 & (1.3) \\
 & & x_B,\ y \geq 0 &
\end{array}
$$

where $x_B(y^0) = b(B) - A_Y(B)y^0$. The x_B and y play the role of slack variables for the constraints (1.2) and (1.3) respectively. Let the dual form of LP_λ denote as follows:

$$
\begin{array}{lll}
DP_B[I:y=y^0+I\lambda\ ] & \min & ux_B(y^0) + vy^0 + c_Y(B)y^0 \\
 & \text{s.t.} & uA_Y(B) - Iv = c_Y(B) \\
 & & u,\ v \geq 0
\end{array}
$$

where u and v are the dual variables associated with (1.2) and (1.3) respectively.

Let D be a dual feasible basis for DP_B and let ρ and ψ_D denote the corresponding primal basic solution of DP_B and the dual one respectively; i.e., $\rho^t = D^{-1}c_Y^t(B)$. We update DP_B with respect to D to obtain the following form:

$$\begin{array}{lll} DP_B[D:y=y^0+I\lambda] & \min & u\{x_B(y^0) - A_Y(B)\psi_D\} + v\{y^0+\psi_D\} + c_Y(B)\{y^0+\psi_D\} \\ & \text{s.t.} & u\,A_Y(B)(D^{-1})^t - v\,(D^{-1})^t = \rho \\ & & u, v \geqq 0 \quad . \end{array}$$

Again, let us consider the dual of $DP_B[D]$:

$$\begin{array}{llll} DDP_B[D:\lambda] & \max & \rho\lambda + c_Y(B)\{y^0 + \psi_D\} & \\ & \text{s.t.} & A_Y(B)(D^{-1})^t\lambda \leqq x_B(y^0) - A_Y(B)\psi_D & (1.4) \\ & & -(D^{-1})^t\lambda \leqq y^0 + \psi_D \quad . & (1.5) \end{array}$$

Let the slack variables for (1.4) and (1.5) be x_B and y respectively. Then, $DDP_B[D:\lambda]$ can be regarded as a transformed form of $LP_\lambda[B]$. We have the following obvious and useful result.

THEOREM 1. Let D be a dual feasilbe basis for $DP_B[I:y=y^0+I\lambda]$ and let ψ_D denote the corresponding dual solution. If y^0 is a feasible solution to LP[B:y], then

(i) $y^1 = y^0 + \psi_D$ is also a feasible solution to LP[B:y], and y^1 becomes an optimal solution if D is an optimal basis.

(ii) $DDP_B[D:\lambda] = LP_\lambda[B:y=y^1 + (D^{-1})^t\lambda]$,

(iii) there are at least n_y zero components among the $x_B(y^1)$, y^1, where n_y is the dimension of y and also the number of rows in DP_B. The number of zeros among the $x_B(y^1)$, y^1 is equal to n_y under the non-degeneracy assumption, which we shall assume hereafter in the coordination problems that will be defined later.

Let us define $T^1 = T^0(D^{-1})^t$ where $T^0 = I$ (identity). Then, we have the new relationship between the y and the parameters λ

$$y = y^1 + T^1\lambda \qquad (1.6)$$

through which the y-variables will be adjusted around the y^1 again. $LP_\lambda[B{:}y = y^1 + T^1\lambda]$ is the problem updated with respect to the new relationship (1.6) and so is the $DDP_B[D{:}\lambda]$. We call T^1 the Parametric Transformation Matrix (P TM hereafter).

2. The Decomposition Algorithm

We use the same notaiton as in [16] for expressing a linear program with respect to a given basis. For simplicity in terminology, we represent the linear program (0.1)-(0.3) as follows:

$$\begin{array}{lll} LP[I{:}y] & & \max\ cx \\ & \text{s.t.} & Ax + A_Y y = b \\ & & x,\ y \geqq 0 \end{array}$$

where A is an m x n matrix having K blocks, each of which contains m_i rows and n_i columns, i.e., $m = \sum_{i=1}^{K} m_i$ and $n = \sum_{i=1}^{K} n_i$, A_Y is m x n_y, and the other vectors are of conformable dimensions. We also assume without loss of generality $c_Y = 0$ in (0.1), because we can always alter the original problem to the above type of problem, by adding a constraint $c_Y y - z^+ + z^- = 0$, $z^+, z^- \geqq 0$ to the x-block.

For the purpose of proving the finiteness of the algorithm we shall assume some ordinary non-degeneracy assumptions like in Theorem 1 (iii) when necessary. And also we assume, for simplicity, the boundedness of the problem.

INITIALIZATION STAGE

The Subproblem. Firstly, we choose initial values, y^0, for the y-variables, and when $y = y^0$ is fixed we have the subproblem

$$\text{SP}[I:y=y^0] \qquad \begin{aligned} &\max\ cx \\ \text{s.t. } &Ax = b - A_Y y^0 \\ &x \geqq 0 \quad . \end{aligned}$$

Notice that the subproblem actually consists of K smaller subproblems of the same type, each of which is of an $m_i x\ n_i$ dimension. We assume, for simplicity, that the subproblem has a finite optimal solution.

Now, let B_o be an optimal basis and let π_{B_o} denote the corresponding dual variables; $\pi_{B_o} = c_{B_o} B_o^{-1}$ where c_{B_o} is the components of c corresponding to the basic variables x_{B_o}. Then, the subproblem updated with respect to B_o becomes

$$\text{SP}[B_o:y=y^0] \qquad \begin{aligned} &\max\ \bar{c}_N x_N \\ \text{s.t. } \quad &x_{B_o} + \bar{A}_N(B_o)\ x_N = B_o^{-1}(b-A_Y y^0) \\ &x_{B_o},\ x_N \geqq 0 \end{aligned}$$

where x_N denote the non-basic variables, and we have $\bar{c}_N \leqq 0$ and $B_o^{-1}(b-A_Y y^0) \geqq 0$ because of optimality. Likewise, the LP[I:y] is updated with respect to B_o as follows:

$$\text{LP}[B_o:y] \qquad \begin{aligned} &\max\ \bar{c}_N x_N - \pi_{B_o} A_Y\ y \\ \text{s.t. } \quad &x_{B_o} + \bar{A}_N(B_o)\ x_N + \bar{A}_Y(B_o)\ y = \bar{b}(B_o) \\ &x_{B_o},\ x_N,\ y \geqq 0 \end{aligned}$$

where $\bar{A}_Y(B_o) = B_o^{-1} A_Y$ and $\bar{b}(B_o) = B_o^{-1}b$.

The First Coordination Problem. We define the coordination problem for the purpose of determining an optimal setting of the y-variables, given that the B_o for the subproblem is fixed. Assume that the y-variables are adjusted through the parameters, λ, around y^0 according to the linear relationships

$$y = y^0 + T^0\lambda \ ,\ T^0 = I(\text{identity}). \qquad (2.1)$$

Then, LP[B_o:y] can be equivalently written as the following transformed form, in the same way as for LP_λ in Section 1:

$$\begin{array}{llrl}
LP_\lambda[B_o:y=y^0+T^0\lambda] & \max\ \bar{c}_N\,x_N - \pi_{B_o}A_YT^0\lambda & & \text{dual var.} \\
\text{s.t.} & x_{B_o} + \bar{A}_N(B_o)\,x_N + \bar{A}_Y(B_o)T^0\lambda & = x_{B_o}(y^0) & :\ u \\
 & -\,T^0\lambda\ +y & = y^0 & :\ v \\
 & x_{B_o},\ x_N,\ y \geqq 0 & &
\end{array}$$

where $x_{B_o}(y^0) = B_o^{-1}(b - A_Yy^0) = \bar{b}(B_o) - \bar{A}_Y(B_o)\,y^0$. Let $DP_{B_o}[I:y=y^0+T^0\lambda]$ denote the corresponding dual problem.

In the above problem, adjusting through the λ involves both adjusting the y-variables and the basic variables, x_{B_o}, but the non-basic variables x_N remain locked at zero. This mechanism will be formulated as a coordination problem. Now, the coordination problem is defined in a primal form as

$$\begin{array}{llrl}
CP_{B_o}[I:\ y=y^0+T^0\lambda] & \max\ -\pi_{B_o}A_YT^0\lambda & & \text{dual var.} \\
\text{s.t.} & x_{B_o} + \bar{A}_Y(B_o)\,T^0\lambda & = x_{B_o}(y^0) & :\ u \\
 & -\,T^0\lambda + y & = y^0 & :\ v \\
 & x_{B_o},\ y \geqq 0. & &
\end{array}$$

Notice that the problem is obtained by dropping all of the non-basic x-variables, x_N, from $LP_\lambda[B_o:y=y^0+T^0\lambda]$. In our algorithm the dual form of the coordination problem is actually solved, and simply the coordination problem shall imply its dual problem in the future description of the computational procedure. Let $DCP_{B_o}[I:y=y^0+T^0\lambda]$ denote the corresponding dual problem.

We assume that CP_{B_o} is bounded. If it is unbounded, so is LP[I:y]. Let D denote an optimal basis for $DCP_{B_o}[I:y=y^0+T^0\lambda]$ and let ψ_D denote the corresponding dual solution. From Theorem 1 we have $y^1 = y^0 + \psi_D$ as an optimal setting for the y-variables, given that the basis B_o for LP[I:y] is fixed. And also, we have

$$\textit{the dual of } DCP_{B_o}[D:y=y^0+T^0\lambda] = CP_{B_o}[I:y=y^1+T^1\lambda]$$

where T^1 denotes the PTM, and

$$T^1 = T^0\,(D^{-1})^t = (D^{-1})^t. \qquad (2.2)$$

Let u_D and v_D be the basic variables of $DCP_{B_o}[I:y=y^0+T^0\lambda]$ for the dual variables, u and v, respectively. For simplicity, we assume that those basic variables are placed in the basis, D, in such an order as (v_D,u_D); this means that if we put

$$-\pi_{B_o}A_Y(D^{-1})^t=\rho=(\rho_v,\rho_u), \qquad (2.3)$$

then the corresponding basic solution is

$$v_D=\rho_v \text{ and } u_D=\rho_u .$$

$LP[B_o:y=y^0+T^0\lambda]$ can be updated with respect to y^1 and the new PTM, T^1 :

$$\begin{array}{llr}
LP_\lambda[B_o:y=y^1+T^1\lambda] & \max\ \bar{c}_N(B_o)x_N+\rho\lambda & \text{dual var.}\\
\text{s.t.} & x_{B_o}+\bar{A}_N(B_o)\,x_N+\bar{A}_Y(B_o)\,T^1\lambda=x_{B_o}(y^1) & :\ u\\
 & -T^1\lambda+y=y^{1^0} & :\ v\\
 & x_{B_o},\ x_N,\ y\geq 0. &
\end{array}$$

We also have the updated subproblem, $SP[B_o:y=y^1]$,corresponding to the above.

In view of our non-degeneracy assumption described in Theorem 1 (iii), there are exactly n_y zeros among the $x_{B_o}(y^1)$, y^1. The corresponding positions are associated with the u_D and v_D. Let the set of the rows in which the corresponding components of $x_{B_o}(y^1)$ are equal to zero denote Γ. In the case of $LP[B_o:y=y^1+T^1\lambda]$, we say simply that the rows belong to the Γ-set, or,in the case of $DCP_{B_o}[D:y=y^0+T^1\lambda]$, that the corresponding columns belong to the Γ-set. The basic variables, among the x_{B_o}, which are in the rows belonging to the Γ-set correspond to the variables called "pseudo-basic" in Beale [6].

All of the rows of $\bar{A}_N(B_o)$ in LP $[B_o:y=y^1+T^1\lambda]$ are classified into either the Γ-set or otherwise. We assume, for simplicity, that all of the Γ-rows are placed at the bottom of $\bar{A}_N(B_o)$, and let $\bar{A}_{N\Gamma}(B_o)$ denote the corresponding part of $\bar{A}_N(B_o)$. The assumption above means, together with the assumed order of (v_D,u_D), that the updated linking matrix, $\bar{A}_Y(B_o)\,T^1$, has the following structure:

$$\bar{A}_Y(B_o)\, T^1 = \begin{bmatrix} R & Q \\ 0 & H \end{bmatrix} \}u_D \tag{2.4}$$

where H is a square matrix composed of the unit row-vectors, which correspond to the columns associated with the u_D in $DCP_{B_o}[D:y=y^0+T^0\lambda]$; H is a kind of permutation matrix and we have $H^{-1}=H^t$.

Similarly, if we assume that all of the zero components among the y^1 are placed at the top, the updated linking matrix, $-T^1$, can be written as follows:

$$-T^1 = \begin{bmatrix} H_o & 0 \\ Q_o & R_o \end{bmatrix} \}v_D \tag{2.5}$$

where H_o denotes the collection of the unit row-vectors, which correspond to the columns associated with the v_D in $DCP_{B_o}[D:y=y^0 + T^0\lambda]$. H_o is also a permutation matrix and $H_o^{-1} = H_o^t$.

OPTIMALITY TEST

THEOREM 2. If we have

$$\rho_u H^t \bar{A}_{N\Gamma}(B_o) - \bar{c}_N(B_o) \geqq 0, \tag{2.6}$$

then the basic solution, $x_{B_o} = x_{B_o}(y^1)$, $y = y^1$, in $LP_\lambda[B_o:y^1+T^1\lambda]$ is optimal for $LP[I:y]$.

Proof. Notice that if the basic solution is optimal for $LP_\lambda[B_o:y^1+T^1\lambda]$, it is also optimal for $LP[I:y]$. Put $u^* = (0,\rho_u H^t) \geqq 0$ and $v^* = (\rho_v H_o^t, 0) \geqq 0$. The condition (2.6) means that the $u = u^*$, $v = v^*$ is a feasible solution to $DP_{B_o}[I:y=y^1+T^1\lambda]$ because of (2.3), (2.4) and (2.5). We have clearly complementary slackness, and the solution is optimal. ||

CHANGING THE BASIS OF THE SUBPROBLEM

The Direction-finding Problem. Now suppose that $\rho_u H^t \bar{A}_{N\Gamma}(B_o) - \bar{c}_N(B_o) \not\geqq 0$, so that we are not finished. Therefore, we try to change the present basis B_o to an attractive one. Put

$$p_N(B_o) = -\rho_u H^t \bar{A}_{N\Gamma}(B_o) + \bar{c}_N(B_o) \nleq 0. \tag{2.7}$$

We define the direction-finding problem:

$$F_{B_o}[I:y=y^1] \qquad \begin{array}{ll} & \max\ p_N(B_o)\ x_N \\ \text{s.t.} & I\ x_{B_o\Gamma} + \bar{A}_{N\Gamma}(B_o)\ x_N = 0 \\ & x_{B_o}\ ,\ x_N \geq 0 \end{array}$$

where $x_{B_o\Gamma}$ denotes the components of x_{B_o} in the Γ-set and plays the role of slack variables.

It is very important for us to notice that solving $F_{B_o}[I:y=y^1]$ means changing the present basis,B_o, of $SP[B_o:y=y^1]$ to an improved basis by restricting the candidates of pivotal rows to the Γ-set and by using the modified objective function. We assume that the degenerate program $F_{B_o}[I:y=y^1]$ is solved by the perturbation method. Notice that the direction-finding problem has either a bounded null solution or unbounded solutions.

If $F_{B_o}[I:y=y^1]$ has an bounded optimal solution, the associated basis-change also induces the basis-change for the subproblem, $SP[B_o:y=y^1]$. Let B_1 denote the induced basis for the subproblem: the subproblem has been updated with respect to the B_1 and we have $SP[B_1:y=y^1]$.

If $F_{B_o}[I:y=y^1]$ is unbounded, we need an extra-operation for the purpose of obtaining such a basis that,among the corresponding basic variables, there is at least one basic variable, the objective coefficient of which is exactly positive.

The Extra-operation in the Unbounded Case. Let x_∞ denote such a variable that its simplex criterion is negative and all the components of its updated column are non-positive.

If x_∞ is not a component of the $x_{B_o\Gamma}$,that is, not a slack variable, then we perform a pivoting operation for bringing x_∞ into the basis instead of a basic variable which is a component of the $x_{B_o\Gamma}$, that is, a slack variable in the basis.

If x_∞ itself is a component of the $x_{B_o\Gamma}$, there is at least one basic variable in the present basis , the objective coefficient of which is positive. Because the corresponding objective value tends to infinity by letting the x_∞ increase. In this case we do not need the extra-operation for our purpose.

Thus, in the unbounded case we also have had a new basis for $F_{B_o}[I:y=y^1]$. Let B_1 denote the induced basis for the subproblem as well as in the bounded case. We have the updated subproblem, $SP[B_1:y=y^1]$, as well. The possibility of performing the extra-operation is insured by the following lemma.

LEMMA 1. When $F_{B_o}[I:y=y^1]$ is unbounded, we can claim the following facts:

(i) There has to be at least one variable chosen among the $x_{B_o\Gamma}$-variables in the present basis, or else the x_∞ itself is a component of the $x_{B_o\Gamma}$.

(ii) Unless the x_∞ is a component of the $x_{B_o\Gamma}$-variables, there has to be at least one non-zero component in the updated column of the x_∞. The non-zero component appears on some of the rows of the basic $x_{B_o\Gamma}$-variables in the present basis.

Proof. (i) From (2.7),$F_{B_o}[I:y=y^1]$ can be equivalently written as

$$\max \quad \rho_u H^t x_{B_o\Gamma} + \bar{c}_N(B_o)\, x_N \qquad (2.8)$$

$$\text{s.t.} \quad I\, x_{B_o\Gamma} + \bar{A}_{N\Gamma}(B_o)\, x_N = 0 \qquad (2.9)$$

$$x_{B_o\Gamma},\ x_N \geqq 0.$$

As we have $\bar{c}_N(B_o) \leqq 0$, the problem does not show the unboundedness if all of the basic variables and x_∞ are the components of the x_N-variables.

(ii) If the x_∞ is a component of the x_N-variables, at least one of the basic $x_{B_o\Gamma}$-variables in the basis has to become positive by letting the x_∞ increase, because of $\bar{c}_N(B_o) \leqq 0$. This means that the claim (ii) is true. ||

Concerning the new basis, B_1, for the subproblem, which is induced from solving the direction-finding problem, we have the following result:

LEMMA 2. Let β denote the basis matrix for $F_{B_o}[I:y=y^1]$, which is associated with the B_1-basis for the subproblem, and let x_β and p_β denote the corresponding basic variables and the corresponding objective coefficients respectively. Then, there is at least one positive component, $p_{\beta j}>0$, among the p_β.

Proof. In the unbounded case of $F_{B_o}[I:y=y^1]$, it is clear owing to the extra-operation above. So, we shall prove it in the bounded case. Let $x_\beta(\varepsilon) > 0$ be the values of the optimal basic variables for the perturbed problem of $F_{B_o}[I:y=y^1]$ for sufficient small $\varepsilon > 0$, and $\lim_{\varepsilon \to o} x_B(\varepsilon) = 0$. Suppose that $p_\beta \leqq 0$. A case $p_\beta = 0$ causes dual infeasibility, because there is at least one positive component among the $p_N(B_o)$: This is impossible. If $p_\beta \neq 0$, we have $p_\beta x_\beta(\varepsilon) < 0$. This contradicts the optimality of $x_\beta(\varepsilon)$, because a null solution becomes an feasible one to the perturbed problem. ||

Expressing the B_1 by the β and the B_0. Let $\bar{F}(B_o)$ denote an enlarged matrix of the basis,β,for $LP[B_o:y]$, i.e., under the assumptions for simplicity, we have

$$\bar{F}(B_o) = \begin{bmatrix} I & \alpha \\ 0 & \beta \end{bmatrix} \quad \text{and} \quad \bar{F}^{-1}(B_o) = \begin{bmatrix} I & -\alpha\beta^{-1} \\ 0 & \beta^{-1} \end{bmatrix} \qquad (2.10)$$

where α denotes the componnents outside the Γ-set of $LP[B_o:y]$ in the same columns as β. Notice that if some variables in the β are chosen from the $x_{B_o\Gamma}$, the corresponding components of α are null. Then, we have

$$B_1^{-1} = \bar{F}^{-1}(B_o)\, B_o^{-1} \quad , \qquad (2.11)$$

which is called Dantzig's Factorization in Marsten and Shepardson [16].

Likewise,

$$\bar{\pi}_{B_1}(B_o) = \bar{c}_{B_1}(B_o)\ \bar{F}^{-1}(B_o) \tag{2.12}$$

where $\bar{c}_{B_1}(B_o)$ denote the objective coefficients of the basic variables corresponding to the basis $\bar{F}(B_o)$ in $SP[B_o:y=y^1]$. From (2.10) we have

$$\begin{aligned}\bar{c}_{B_1}(B_o) &= c_{B_1} - \pi_{B_o} B_1 \\ &= (0,\ \bar{c}_\beta(B_o)\ )\end{aligned} \tag{2.13}$$

where $\bar{c}_\beta(B_o)$ denote the component of $\dot{\bar{c}}_N(B_o)$ corresponding to the β.

Let π_{B_1} denote the dual variables associated with the B_1 for $SP[B_o:y=y^1]$. Then, from (2.11), (2.12) and (2.13) we have

$$\pi_{B_1} = \pi_{B_o} + \bar{\pi}_{B_1}(B_o)\ B_o^{-1}\ , \tag{2.14}$$

which was also shown in [16]. In addition,

$$\bar{A}_Y(B_1) = \bar{F}^{-1}(B_o)\ \bar{A}_Y(B_o) \tag{2.15}$$

Thus, we have obtained the updated $LP[B_1:y]$, as well as the subproblem, $SP[B_1:y=y^1]$.

THE SUBSEQUENT COORDINATION PROBLEMS

Now, we would like to define the subsequent coordination problem for the updated subproblem $SP[B_1:y=y^1]$. As well as the first problem, the purpose is to determine an optimal setting of the y-variables, given that the new basis, B_1, for the subproblem is fixed.

First of all, we shall define a new relationship between the y-variables and the λ-parameters for the purpose of reducing the amount of work required for updating the linking matrices with respect to B_1.

The β-*transformation*. Let us define the intermediate PTM, T^1*, as

$$T^1* = T^1 \begin{bmatrix} I & 0 \\ 0 & H^t\beta \end{bmatrix} \tag{2.16}$$

and we consider the following new relationship :

$$y = y^1 + T^1{*}\lambda \qquad (2.17)$$

The linking matrices in $LP_\lambda[B_1:y=y^1 + T^1{*}\lambda]$ are obtained as follows:

$$\bar{A}_Y(B_1)T^1{*} = \bar{F}^{-1}(B_o)\ \bar{A}_Y(B_o)\ T^1{*}, \qquad (2.18)$$

by (2.4) and (2.10) ,

$$= \begin{pmatrix} R & Q - \alpha\beta^{-1}H \\ 0 & \beta^{-1}H \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & H^t\beta \end{pmatrix} = -\begin{pmatrix} R & QH^t\beta - \alpha \\ 0 & I \end{pmatrix}$$

$$T^1{*} = -\begin{pmatrix} H_o & 0 \\ Q_o & R_o \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & H^t\beta \end{pmatrix} = -\begin{pmatrix} H_o & 0 \\ Q_o & R_oH^t\beta \end{pmatrix} \quad . \qquad (2.19)$$

By (2.14),

$$-\ \pi_{B_1} A_Y\ T^1{*} = -\ \pi_{B_o} A_Y T^1{*} - \bar{\pi}_{B_1}(B_o)B_o^{-1} A_Y\ T^1{*} \quad ,$$

by (2.12)

$$= -\ \pi_{B_o} A_Y\ T^1{*} - \bar{c}_{B_1}(B_o)\ \bar{A}_Y(B_1)\ T^1{*} \quad , \qquad (2.20)$$

and by (2.3) and (2.13)

$$= (\rho_v,\rho_u H^t\beta) - (0,\bar{c}_\beta(B_o))\ \bar{A}_Y(B_1)\ T^1{*}$$

$$= (\rho_v,\rho_u H^t\beta) - (0,\bar{c}_\beta(B_o)) \text{ , by(2.18),}$$

$$= (\rho_v,\rho_u H^t\beta - \bar{c}_\beta(B_o)) \ . \qquad (2.21)$$

We should pay attention to the component of $\rho_u H^t\beta-\bar{c}_\beta(B_o)$. Then, we note that these components associated with the slack variables in the basis, β, are not changed from the corresponding components in $LP_\lambda[B_o:y=y^1+ T^1\lambda]$, because the corresponding components of $\bar{c}_\beta(B_o)$ are null. Furthermore, the other components associated with the basic variables chosen from the x_N are simply replaced by the corresponding components of $-\ p_\beta(B_o)$ in $F_{B_o}[I:y=y^1]$.

In conclusion, the linking part can be easily updated with the new relationship between the y and the λ as (2.18), (2.19) and (2.21), only by using the basic matrix, β, for the direction-finding problem. We call simply those transforming operations the β-transformation hereafter.

The Second Coordination Problem. Now, we can define the coordination problem derived from $LP_\lambda[B_1{:}y=y^1+T^1*\lambda]$ as well as before.

$$
\begin{array}{llrl}
CP_{B_1}[I{:}y=y^1+T^1*\lambda] & & \max & -\pi_{B_1} A_Y T^1*\lambda \\
 & \text{s.t.} & x_{B_1} + \bar{A}_Y(B_1)\, T^1*\lambda & = x_{B_1}(y^1) \\
 & & -\, T^1*\lambda + y & = y^1 \\
 & & x_{B_1},\ y \geq 0\ . &
\end{array}
$$

Similarly, $DCP_{B_1}[I{:}y^1+T^1*\lambda]$ denotes its dual form.

THEOREM 3. Under the non-degeneracy assumption in Theorem 1 (iii), $LP[I{:}y]$ is strictly improved after solving the $CP_{B_1}[I{:}y=y^1+T^1*\lambda]$.

Proof. It is sufficient for us to show that the objective function for $CP_{B_1}[I{:}y=y^1+T^1*\lambda]$ has a positive value. From Lemma 2 and (2.21), there is at least one negative objective coefficient in $CP_{B_1}[I{:}y=y^1+T^1*\lambda]$. And that, the corresponding component of the λ belongs to the Γ-set. This shows that strict improvement in $CP_{B_1}[I{:}y=y^1+T^1*\lambda]$ is insured under the non-degeneracy assumption. $||$

As well as in the first coordination problem, we solve $DCP_{B_1}[I{:}y=y^1+T^1*\lambda]$ to obtain the new y-values, y^2, and the new PTM, T^2, and then perform the optimality test. This completes one major iteration of the algorithm.

Notice that the bases for the subproblem, B_1, B_2, ..., are generally not dual feasible in the subproblem except B_o at the initialization stage. The possibility of performing the extra operation in $F_{B_k}[I{:}y=y^{k+1}]$ is insured only under the optimality of the subproblem by Lemma 1. Accordingly, we may, on rare occasions, fail to find the negative pivotal element in the unbounded column. Only when such a case happens, we need re-optimize the subproblem for $y = y^{k+1}$.

THE ALGORITHM

Our algorithm may now be summarized as follows:

Step 0. Choose y^0 and set $T^0 = I$ as the starting PTM, and $k = 0$.

Step 1. Solve the subproblem, $SP[I:y=y^k]$, to obtain the optimal basis B_k.

Step 2. Solve the first coordination problem $DCP_{B_k}[I:y=y^k + T^k\lambda]$ to obtain the optimal setting of y, y^{k+1}, and the new PTM, $T^{k+1} = T^k(D^{-1})$, where D is the optimal basis of DCP_{B_k}, and the corresponding solution, ρ^k.

Step 3. Identify the Γ-set.

Step 4. If $p_N(B_k) \equiv \rho_u^k H^t \bar{A}_{N\Gamma}(B_k) - \bar{c}_N(B_k) \geq 0$, stop : $x_{B_k}(y^{k+1})$, y^{k+1} is optimal for $LP[I:y]$.

Step 5. If the Γ-set is not void, solve the direction-finding problem $F_{B_k}[I:y^{k+1}]$. If $F_{B_k}[\beta:y^{k+1}]$ is bounded, then let the induced basis for the subproblem be B_{k+1}, and then go to Step 7. If it is unbounded, then go to Step 6 for the extra-operation.

If the Γ-set is void, then set $k = k+1$, and go to Step 1 by fixing $y = y^{k+1}$.

Step 6. Check the existence of a negative element in the unbounded column.

If there is not, then set $k = k+1$, and go to Step 1 by fixing $y = y^{k+1}$. Otherwise, perform the extra-operation for $F_{B_k}[I:y^{k+1}]$ to obtain the basis, β, and let B_{k+1} be the induced basis for the subproblem.

Step 7. Perform the β-transformation for the matrices and the RHS vector in $DCP_{B_k}[D:y=y^k + T^k\lambda]$ to obtain the intermediate PTM, $T^{k+1}*$, and the second coordination problem $DCP_{B_{k+1}}[I:y=y^{k+1} + T^{k+1}*\lambda]$.

Step 8. Solve $DCP_{B_{k+1}}[I:y=y^{k+1} + T^{k+1}*\lambda]$ to obtain the new y-values, y^{k+2}, and the new PTM, T^{k+2}. Set $k = k+1$ and go to Step 3.

Finite Convergence. The finiteness of the algorithm is insured by the following theorem.

THEOREM 4. The proposed algorithm terminates in a finite number of the major iterations under the non-degeneracy assumption.

Proof. It is seen by Theorem 3 that the problem LP[I:y] is strictly improved through the coordination of the y-values and the subsequent basis-change under the non-degeneracy assumption. The y-values are optimally set with respect to the given basis, B_k, in the coordination problem. This implies a finite termination of the algorithm, because there are only a finite number of possible bases B_k's for the subproblem in a course of selecting y^k in the algorithm. ||

Implementation. In order to implement the algorithm for the dynamic linear programs, we have to solve K subproblems separately, and we need to bring out K permutation matrices like H from the coordination problem, which are used for the β-transformation of K blocks. The dimension of the dual coordination problem to be solved at every cycle becomes $n_y \times (\sum_{i=1}^{K} m_i + n_y)$, which shows that the linear program has extremely many columns as compared to the number of rows. See [3] for the detailed procedure to implement the algorithm.

3. Computational Experience

MULPS. An experimental code, named MULPS (Multi-period Linear Programming System), for the weakly coupled linear programs was written in FORTRAN using the SEXOP[15] for HITAC 8250 Computer. The computer has 160 KB main storage and disc storage devices. Its operating system does not have virtual memory. The SEXOP is used for solving all linear programs in the MULPS. A version of the SEXOP for the HITAC 8250[5] runs by overlay between the main storage and the disc storage. The MULPS can solve dynamic linear programs having up to 30 linking variables and 6 periods, each having up to 30 rows and 50 columns.
The Purpose of the Experiments. The present experiment primarily focuses on the number of cycles required for optimality by the algorithm, and also we observe the degrees of optimality throughout the whole coordination process. The computing time is secondarily observed, because the number of cycles will have a great influence on the computing time, and the MULPS has not been designed and coded with the intention of investigating strictly the computing time.

For the purpose of comparing the algorithm with the direct simplex approach, we tentatively convert the MULPS to a large computer, FACOM M-160S(comparable to IBM 370-148) having 768 main storage and virtual memory. We use the SEXOP for the direct simplex method.
The Test Problem. Our test problems were mainly derived from (i) a version of Gilmore and Gomory's model of cutting stock problems[9], (ii) Manne's model of multi-period economic planning[7], and (iii) fictitious refinery production planning models. These problems are listed in Table 1.

Generally speaking, in a case of multi-period models it is relatively easy to estimate the "good" initial values for the linking variables, so that these make easily the problem feasible. However, in the present experiment, the initial values are set at zero except for R1B. For R1B, the optimal values of y for R1A are used.

The Results. The number of cycles required for optimality and the CPU computing time are summarized in Table 2. Table 3 illustrates the degree of optimality at every cycle. In Table 4 the CPU computing time up to every cycle throughout the optimization is described in detail for Problem MA1. In Figure 1 the total CPU computing time and that per cycle are plotted for the corresponding number of periods for the six problems G3A - G6B. Notice that those problems have subproblems of the same dimension, but a different number of periods.

In Table 5 we compare both the CPU computing time and the amount of storage required in the system with those by the direct simplex method(SEXOP) for MA1.

The Conclusion. From Table 2 we note that the number of cycles required for optimality is almost equal to, or less than the number of periods. For the purpose of comparing it with that by the algorithms of column-generation scheme, we shall refer to the earlier results of Glassey's algorithm [10] and of Ho and Manne's one [14].

In Glassey [10] the computational result for almost the same model as MA1 derived from [7] was presented. The number of cycles was reported to be 31, which shows to be five times larger than that for MA1. In Ho and Manne [14] the two test problems coded SC50A and SC50B have 6 periods and the dimensions are rather smaller than R1 and R2 among our problems. The number of cycles was reported to be between 25 and 35, whcih shows to be six or eight times larger than ours. However, it is reported in the recent comparative study of their method, Ho and Loute [13], that the number of cycles is greatly reduced. We could not trace the same problems in the present experiment.

From Table 3 we note that the process of convergence is fairly fine and the "long tail" of convergence scarcely occurs. The degree of optimality attains a very high position at a relatively early coordination cycle. The degree at the first coordination is beyond 70% in almost all cases such that the initial values for the y-variables make the problem feasible at the initial stage. This feature seems to be significant in a practical use, and a near-optimal strategy may work effectively.

From Table 4 we note that the CPU computing time per cycle tends to decrease slightly. All subproblems are optimized before solving the first coordination problem. Therefore, much more time is consumed at the first cycle. Except some special occasions, solving the subproblems are skipped and the direction-finding problems are solved only for the non-optimal blocks. We have observed so far that the number of non-optimal blocks gradually decreases according as the coordination proceeds.

Table 5 shows that the MULPS is four times faster than the direct method concerning the computing time, and requires only a half of memory for the direct simplex method in the case of MA1.

Acknowledgment

The author wishes to thank Professor Roy E. Marsten, The University of Arizona for releasing the SEXOP program to him and for comments and suggestions which have results in an improved version of the paper, and also Professor Leon S. Lasdon, The University of Texas at Austin, for the suggestion on using the SEXOP program while the author was at Case Western Reserve University during the latter half of 1974. The author is much indebted to Messrs. M.Morita and H.Nishimoto, his students at Kobe University of Commerce, for helping to code the MULPS and to generate the test problems.

REFERENCES

[1] T. Aonuma, "A Nested Multi-level Planning Approach to a Multi-period Linear Planning Model", Working Paper No.35, Kobe University of Commerce (Kobe, January 1977).

[2] " , "A Two-level Algorithm for Two-stage Linear Programs", *Jour. of Operations Research Society of Japan* 21, 171 - 187 (1978).

[3] " , "An Extension of the Two-level Algorithm to Optimizing Weakly Coupled Dynamic Linear Systems", Working Paper No.41, Kobe University of Commerce (Kobe, March 1978).

[4] " , "Computational Experience in Using a Decomposition Algorithm (MULPS) for Multi-period Dynamic Linear Programs", Working Paper No.44, Kobe University of Commerce (Kobe, July 1978).

[5] T.Aonuma and N.Nakahara, SEXOP/HITAC 8250 - User's Manual - (in Japanese), Research Report No.9, The Computer Center, Kobe University of Commerce (Kobe, December 1977).

[6] E.M.L.Beale,"The Simplex Method for Structured Linear Programming Problems", *Recent Advances in Mathematical Programming(ed. R.Graves and P.Wolfe)*, McGraw-Hill, New York, 1963, 133-148.

[7] C.R.Blitzer, H.Cetin, and A.S.Manne, "Dynamic Five-Sector Model for Turkey, 1967-82", Memorandum No.70, Research Center in Economic Growth, Stanford University (April 1969).

[8] A.M.Geoffrion, "Primal Resource-directive Approaches for Optimizing Nonlinear Decomposable Systems", *Opns. Res.* 18, 375-403 (1970).

[9] D.C.Gilmore and R.E.Gomory, "A Linear Programming Approach to the Cutting Stock Problem Part II", *Opns. Res.* 11, 863-888(1963).

[10] C.R.Glassey, "Nested Decomposition and Multi-stage Linear Programs", *Mgmt. Sci.* 20, 282-292(1973).

[11] R.C.Grinold, "Steepest Ascent for Large Scale Linear Programs", *SIAM Review* 14, 447-467(1972).

[12] G.H.Heal, *Theory of Economic Planning,* North-Holland, Amsterdam, 1973.

[13] J.H.Ho and E.Loute, "A Comparative Study of Two Methods for Staircase Linear Programs", mimeograph BNL-23156, August 1977.

[14] J.H.Ho and A.S.Manne, "Nested Decomposition for Dynamic Models", *Math. Prog.* 6, 121-140(1974).

[15] R.E.Marsten, *User's Manual for SEXOP,* Sloan School of Management, Massachusetts Institute of Technology (Cambridge, Februray 1974).

[16] R.E.Marsten and F.Shepardson, " A Double Basis Simplex Method for Linear Programs with Complicating Variables", Working Paper, College of Business and Public Administration, University of Arizona (Tucson, August 1978).

[17] T.Aonuma, M.Morita and H.Nishimoto, *User's Manual for MULPS/HITAC 8250* (in Japanese), Research Report No.13, The Computer Center, Kobe University of Commerce(Kobe, December 1978).

[18] S.I.Gass, "The Dualplex Method for Large-Scale Linear Programs", ORC66-15, Operations Research Center, University of California (Berkeley,June 1966).

[19] C.Winkler, "Basis Factorization for Block-Angular Linear Programs: Unified Theory of Partitioning and Decomposition Using the Simplex Method",RR-74-22, IIASA (Schloss Laxenburg, November 1974).

[20] T.Aonuma,"A Decomposition-Coordination Approach to Large-Scale Linear Programming Models - An Aspect of Applications of Aonuma's Decomposition Method - ", forthcoming.

TABLE 1

Dimensions of Test Problems

		Entire Problem				Subproblem		
Problem	Period	Rows	Col.'s*	%Density	L.V.**	Rows	Col.'s	%Density
Gilmore-Gomory								
G3A	3	90	162	3.0	12	30	50	9.1
G3B	3	90	162	3.1	12	30	50	9.3
G4A	4	120	218	2.2	18	30	50	9.1
G5A	5	150	274	1.8	24	30	50	8.0
G6A	6	180	330	1.3	30	30	50	8.0
G6B	6	180	330	1.5	30	30	50	9.3
Manne's Model								
MA1	6	116	266	1.8	26	19-20	37-43	9.7-11.0
Refinery Prod.								
R1A	6	60	186	2.7	30	10	26	20.0
R1B	6	Only the linking matrix is different from R1A above.						
R2A	6	90	198	2.5	30	15	28	15.0

* Includes slack variables.

** L.V. denotes the number of linking variables.

TABLE 2

Number of Cycles and CPU Computing Time

Problem	Periods	Number of Cycles	CPU Computing Time	
			min.	sec.
G3A	3	4 (0)*	5	40
G3B	3	5 (1)	6	30
G4A	4	5 (0)	10	10
G5A	5	5 (1)	15	00
G6A	6	8 (0)	24	45
G6B	6	6 (0)	24	50
MA1	6	6 (0)	20	00
R1A	6	4 (1)	7	10
R1B	6	3 (0)	5	48
R2A	6	5 (1)	12	10

* A parenthesized figure denotes the number of times returned to Step 1 in Step 6. In Step 1 the subproblem is reoptimized.

TABLE 3

Degree of Optimality and Number of Cycles

	Number of Coordination Cycle								
Problem	0	1	2	3	4	5	6	7	8
G3A	0	94.7	96.2	98.0	100%				
G3B	0	0	18.5	71.5	71.5	100%			
G4A	0	89.6	93.1	97.3	97.5	100%			
G5A	0	87.7	88.6	100^{-}	100^{-}	100%			
G6A	0	89.7	95.5	98.6	99.0	99.5	99.5	99.8	100%
G6B	0	76.6	87.6	96.6	97.3	99.3	100%		
MA1	*	0	32.6	71.9	87.2	96.8	100%		
R1A	*	*	*	0	100%				
R1B	*	0	97.1	100%					
R2A	*	*	0	69.2	84.0	100%			

Note: An asterisk denotes that a feasible solution is not found yet. 0 denotes feasibility attained for the first time. 100^{-} denotes a near 100.

TABLE 4

CPU Computing Time up to Every Cycle for MA1

Up to	Optimization	Number of Cycle					
	of Subprob.'s	1	2	3	4	5	6
	min. sec.						
Computing Time	2 37	4 49	8 30	11 27	14 17	17 07	20 00
Per Cycle	-	4 49	3 41	2 57	2 50	2 50	2 53

TABLE 5

Comparison of MULPS with Direct Simplex Method

CODE			MULPS		SEXOP		SEXOP/MULPS
PROBLEM			CYCLES	TIME	ITER-ATIONS	TIME	RATIO IN TIME
No.	Periods	Size					
MA1	6	116 x 266	6	31.6	192	134.0	4.2
G6B	6	180 x 330	6	56.6	306	215.9	3.8
MAIN STORAGE USED			294 KB		729 KB		2.5

i) The times reported are in CPU seconds on a FACOM M-160 (comparable to IBM 370/148).

ii) The FACOM M-160 has 768 KB real memory and 16 MB virtual memory, which is under OS IV/X8 (comparable to IBM OS/VS2). The FORTRAN IV HE compiler with OPTIMIZE(2) is used throughout (comparable to IBM FORTX compiler with OPT = 2).

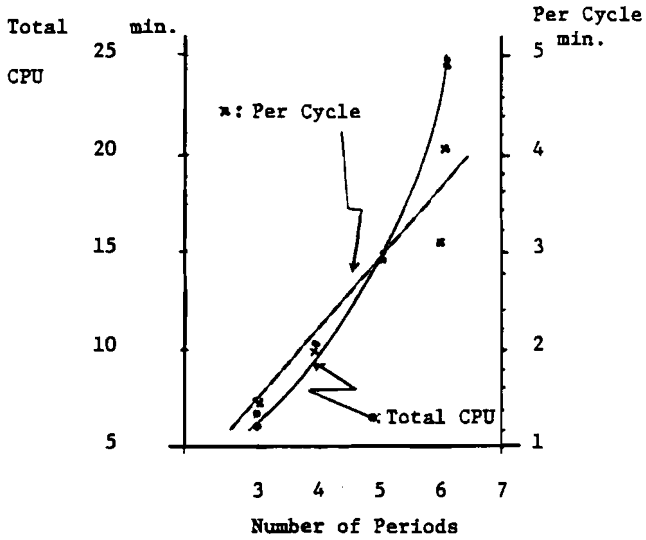


Fig.1 CPU Computing Time and Number of Periods for Problems G3A-G6B.

ASPECTS OF BASIS FACTORIZATION FOR BLOCK-ANGULAR SYSTEMS WITH COUPLING ROWS

Michael Bastian

Rheinisch–Westfälische Technische Hochschule
Aachen

In the class of decomposition and factorization algorithms characterized by Winkler [9], certain subinverses have to be updated by elementary column- *and* row-matrices. It is shown how to keep a Forrest–Tomlin representation of these subinverses in spite of the row transformations.

For the case of staircase systems – viewed as nested blockangular systems – problems of data handling are addressed.

1. INTRODUCTION

Many algorithms have been proposed over the years to take advantage of noticeable block structures in the coefficient matrices of Linear Programming problems. We are concerned here with modifications of the Simplex Method which are generally based on a factorization of the basis inverse that preserves the block structure.

The general judgement of whether special LP-algorithms are useful or not has changed during the past 25 years several times.

In 1955 DANTZIG [3] wrote:

'Now the main obstacle toward the full application of standard linear programming techniques to dynamic systems is the magnitude of the matrix for even the simplest situation. For example, a trivial 15-activity-7-item static model, would become a 180-activity by 84-item system, which is considered a large problem for application of the standard simplex method.···It is clear that dynamic models must be treated with special tools if any progress is to be made toward solutions of these systems'.

With every next generation of computers and improvements of general LP-systems people tended to disregard block-structures. On the other hand, the size of the problems that had to be solved also increased enormously, and special methods were reconsidered.

Right now, it seems to me that we are in a period where a lot of attention is paid to the efficient solution of block-structured Linear Programs, one of the main reasons probably being the development of really huge multi-period multi-area energy models at many places in the world.

But there are other reasonsto apply special algorithms also to block-structured problems of medium size:

- In many situations one knows in advance that for certain computations only a small number of 'parts' of the factorized inverse as well as of the original data is used.
  By an efficient buffering system one might be able to have most of the relevant data in core during these computations.

- In the near future it may become quite standard to use programming languages that allow for the implementation of parallel algorithms. There are more possibilities to make use of parallel computations if block-structures are maintained.

If that is so, why don't people use factorization methods for solving block-structured models today? The main reason, I think, is that so far most special LP-algorithms were developed in an academic environment, where implementations - if there were any - served as a standalone test vehicle for the factorizational and decompositional part of the problem. What should be done is to make sure that the highly efficient procedures developed for general large scale LP remain part of the system whereever this is possible: Factorization for structured LP should be an option not a separate system. To be more specific: A system should have several options for different block-structures which share as many routines as possible and extensively use standard LP-procedures.

After giving a brief summary on block-structures and factorization methods, I shall show how subinverses of a basis-factorization may be updated by the Forrest-Tomlin Method, even though some of the updating transformations are elementary row matrices.
This situation occurs for example in the class of algorithms derivable from Carlos Winkler's unified theory of partitioning and decomposition (WINKLER [9]).

One of the block-structures which are most often encountered in practical applications and hard to solve are staircase-structures. Problems of data handling when using Winkler's nested factorization approach for solving staircase structured LP's are addressed.

2. Block-Structured Systems

Let A be an $m \times n$-matrix with real coefficients,
M the set of nonempty subsets of $\{1,2,...,m\}$,
N the set of nonempty subsets of $\{1,2,...,n\}$,
$1 < k < m$ and $K := \{1,2,...,k\}$

Def.:

A (row-oriented) block-structure of A is a set $BS(A) := \{(\alpha_i, \gamma_i) | i \in K\}$ of pairs $(\alpha_i, \gamma_i) \in M \times N$ such that

(1) $\{\alpha_i | i \in k\}$ is a partition of $\{1,..,m\}$;

(2) every nonzero element $A_{h_j} \neq 0$ of A is contained in one of the sub-matrices
A_{α_i, γ_i} $(i \in K)$.

The matrices A_{α_i, γ_i} $(i \in K)$ are called blocks.

A block of a blockstructured matrix is thus given by a set α of rows and at least those columns that have a nonzero element in any positive number of rows in α.

The factorization methods under consideration keep representations of the basis-inverses which retain (to a large extent) the block-structure of the corresponding bases. The idea is that the FTRAN- and BTRAN-operations of the Simplex Method are simpler to perform if nonzeroes only appear in certain blocks, the position of which is apriori known.

3. Basisfactorization for Blocktriangular Matrices

Def.:

A real m × n-matrix A is called blocktriangular, if it has the following block-structure:

$BS(A):=\{(\alpha_i,\gamma_i) \in M \times N \mid i \in K,\ \gamma_i \not\subset \bigcup_{j>i} \gamma_i \quad \forall\ 1<i<k\}.$

The class of blocktriangular matrices is quite large and contains the majority of block-structured coefficient matrices of 'real world' Linear Programs. Well-known substructures are:

a) the blockangular structure (with coupling rows and coupling variables):

$BS:=\{(\alpha_i,\gamma_i) \mid i \in K,\ \gamma_1=\{1,\dots,n\},\ \gamma_i \not\subset \gamma_k\ \forall\ i \neq k,$
$\gamma_i \cap \gamma_j = \gamma_k\ \forall\ 1 \neq i \neq j \neq 1\}$;

b) the staircase structure:

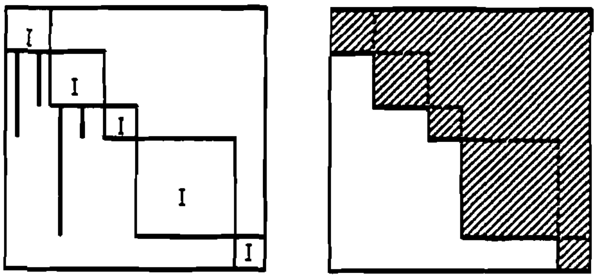
$BS:=\{(\alpha_i,\gamma_i) \mid i \in K,\ \gamma_i \cap \gamma_{i+1} \neq \emptyset\ \forall\ i \neq k,\ \gamma_i \cap \gamma_j = \emptyset\ \forall\ |i-j| > 1\}$

It ist now twenty-five years ago that George Dantzig [3] suggested to modify the simplex algorithm when applied to problems with block-triangular coefficient matrices. The main idea of that early paper, i.e. the factorization of block-triangular bases into a blocktriangular and a very sparse factor, remains the same in most of today's approaches.

Let $\bar{B}$ be a blocktriangular basis. By column exchanges it is possible to yield a matrix B from $\bar{B}$ with the following properties:

a) B is blocktriangular up to a few 'spikes'
b) B can be factorized into two invertible matrices F and L such that $B^{-1} = L^{-1} \cdot F^{-1}$, where F^{-1} is blocktriangular and L^{-1} is very sparse.
c) The submatrices on the main diagonal of B are square.

$B^{-1} = L^{-1} \cdot F^{-1} =$



This structure greatly simplifies the operations BTRAN and FTRAN of the simplex method.

BTRAN: $\pi = c \cdot B^{-1} = (c \cdot L^{-1}) \cdot F^{-1}$

FTRAN: $\bar{d} = B^{-1} \cdot A_{\bullet s} = L^{-1} \cdot (F^{-1} \cdot A_{\bullet s})$.

The multiplication by F^{-1} is simple because of the structure and the multiplication by L^{-1} is fast because of the small number of nonzeroes.

The main challenge is to provide an efficient method for maintaining this structure of the inverse during the iterations of the simplex method.
KALLIO and PORTEUS [6] published a solution to this problem in 1977. A different approach was taken by PEROLD and DANTZIG [4].

In the case of particular blocktriangular structures the matrix F^{-1} is further factorized. We shall consider here Winkler's factorization for blockangular structures with coupling rows.

4. Blockangular Systems With Coupling Rows

Def.:

A real $m \times n$-matrix A is called blockangular (with coupling rows), if it has the following block-structure:
$BS(A) := \{(\alpha_i, \gamma_i) \in M \times N \mid i \in K,\ \gamma_1 = \{1,..,n\},\ \gamma_i \cap \gamma_j = \emptyset\ \forall\ 1 \neq i \neq j \neq 1\}$

In this paragraph we shall consider coefficient matrices of the structure just defined.

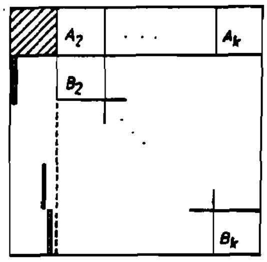
4.1 Winkler's Factorization

Let $\bar{B}$ be a basis of A and $\beta \subset \{1,2,..,n\}$, $|\beta| = m$, its set of column indices.
It follows from $\bar{B}$ being invertible that there exists a partition $\{\beta_i\}_{i \in K}$ of β such that

(a) $\beta_i \subset \gamma_i$ and $|\beta_i| = |\alpha_i|$ $(i = 1,2,..,k)$

(b) $\bar{B}_{\alpha_i, \beta_i}$ is invertible $(i = 2,..,k)$

A rearrangement of the columns of $\bar{B}$ (to the order $\beta_1,..,\beta_k$) yields:

$B =$ 

where the B_i are square and invertible and the first $|\beta_1|$ columns of B are very sparse.

Let $C_i : = -A_i \cdot B_i^{-1}$ for i = 2,3,..,k. Then there exists a decomposition of B into three invertible factors B_N, W and L such that $B^{-1} = L^{-1} \bullet W^{-1} \bullet B_N^{-1}$ has the form:

$$B^{-1} = \begin{bmatrix} I & & & \\ -V_2 & I & & \\ \vdots & & \ddots & \\ -V_k & & & I \end{bmatrix} \cdot \begin{bmatrix} B_W^{-1} & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix} \cdot \begin{bmatrix} I & C_2 & \dots & C_k \\ & B_2^{-1} & & \\ & & \ddots & \\ & & & B_k^{-1} \end{bmatrix}$$

Here

$$V := \begin{bmatrix} V_2 \\ \vdots \\ V_k \end{bmatrix}$$

is very sparse and $\begin{bmatrix} B_W \\ V \end{bmatrix} = B_N^{-1} \cdot \bar{B}_{\bullet\beta_1}$.

Notice that in order to maintain B^{-1} it is sufficient to store (in addition to the coefficient matrix A) a sparse matrix V as well as k 'subinverses' B_W^{-1}, B_2^{-1},..,B_k^{-1}.

The simplifications during FTRAN and BTRAN are tremendous, because only very few of the subinverses are needed. A similar statement is true for the actualization of the inverse-representation during the iterations of the simplex method (WRETA).

There are three different update situations depending on the pivot row p and the entries of the matrix V.

Disregarding the changes performed on elements of the V-matrix (details are explained in WINKLER [9] or BASTIAN [1]) an update consists of:

case 1: adding a column eta to the file of B_W^{-1}.
case 2: adding a column eta to one of the subinverses B_i^{-1} $(i \in \{2,..,k\})$.
case 3: adding a column eta to one of the subinverses B_i^{-1} $(i \in \{2,..,k\})$,
adding a row eta <u>and</u> a column eta (which pivot in the same row)
to B_W^{-1}.

There are pivot selection strategies that tend to reduce the number of occurrences of case 3 and completely avoid this situation during the first part of phase 1.

Our explanation of the update cases has tacitly assumed that the product-form of the inverse (PFI) is used for all subinverses. The B_i^{-1}, $(i = 2,3,..,k)$, could as well be kept in EFI using the Forrest-Tomlin method. Infact, this should be done in view of the advantages of the EFI and our aim to incorporate latest LP-technology into special routines for block-structured problems.

For B_W^{-1} the situation is more complicated, as case 3 does not correspond to a simple column-exchange in B_W. For this matrix, however, an updating procedure which reduces the growth of the eta-file would be extremely desirable, because (in contrast to the other subinverses) B_W^{-1} is involved in each BTRAN- and in each FTRAN-operation. Moreover, the columns of B_W are not contained in the coefficient matrix A and have to be computed prior to each reinversion.

In the next section it is shown that also B_w^{-1} can be stored and maintained in the Elimination Form of the Inverse. The multiplication by two elementary matrices in update-case 3 is replaced by a modified Forrest-Tomlin procedure which yields a growth of the eta-file comparable to the PFI (at most three new eta-vectors have to be stored; one row is erased in the U-file).

In case 1, however, one enjoys all benefits of the classical Forrest-Tomlin method which should yield considerable savings in total computation time.

4.2 Using the Forrest-Tomlin Method for Updating B_w^{-1}

In the update-situation under consideration (case 3) one is given an $m_1 \times m_1$-inverse B_w^{-1} and an m_1-row-vector $v \neq 0$, from which a nonzero component v_z is chosen.

Let E_z be obtained from the identitiy matrix by replacing its z-th row by the vector v and define

$$\bar{B}_w^{-1} := E_z \cdot B_w^{-1}.$$

Let $\hat{B}_w$ be an invertible matrix obtained from $\bar{B}_w$ by replacing its z-th column by an m_1-column d. Then there exists an elementary column matrix E_s such that

$$\hat{B}_w^{-1} = E_s \cdot E_z \cdot B_w^{-1} .$$

In the following sections a different representation for $\hat{B}_w^{-1}$ is derived.

4.2.1 Assumption

B_w^{-1} is given by two factors U^{-1} and L^{-1} which are stored in product form

$U^{-1} = U_1 \cdot U_2 \dots U_{m_1}$ and $L^{-1} = L_{n_1} \cdot L_{n_1-1} \dots L_1$

on different files (the U-file and the L-file) in order to allow for the insertion of new elementary matrices between U_{m_1} and L_{n_1}. There are no further assumptions on L^{-1}, but the existence of a permutation matrix P is postulated such that $P \cdot U \cdot P^{-1}$ is upper triangular. Because of this structure, the eta vector of U_i may be obtained directly from the i-th column of $U \cdot P^{-1}$ $(i = 1,2,..,m_1)$

Notice that the situation discribed is for example given right after an inversion using LU-decomposition.

4.2.2 Theorem

Let B_w^{-1} satisfy assumption 3.2.1 and $v_Z \neq 0$. There exist an elementary column matrix T, elementary row matrices R_1 and R_2, an eta column y as well as a representation

$$\hat{B}_w^{-1} = \hat{U}^{-1} \cdot \hat{L}^{-1}$$

of the matrix $\hat{B}_w^{-1} = E_S \cdot E_Z \cdot B_w^{-1}$, such that $\hat{B}_w^{-1}$ satifies assumption 3.2.1.

The product forms of $\hat{U}^{-1}$ and $\hat{L}^{-1}$ are easily derived from the representation of B_w^{-1} by the following modification of the Forrest-Tomlin method:

(1) add the eta vectors of R_1, T, R_2 to the L-File (in that order);
(2) mark the eta vector of the U-file which pivots in row z as being deleted;
(3) delete all elements of the U-File having row index z;
(4) add y to the U-file.

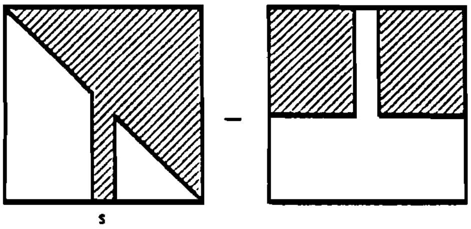
4.2.3 Outline of the Proof

As $\hat{B}_w$ and $\bar{B}_w = L \cdot U \cdot E_z^{-1}$ differ by just one column, the same is true for $\bar{U} := L^{-1} \cdot \hat{B}_w$ and $U \cdot E_z^{-1}$.
We have:

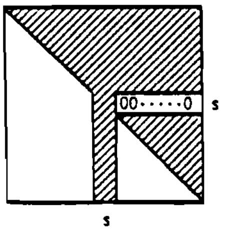
$$\bar{U}_{\bullet z} = L^{-1} \, . \, d$$

$$\bar{U}_{\bullet j} = U_{\bullet j} - v_j \cdot U_{\bullet z} \qquad \forall\, j \neq z. \qquad (1)$$

From $P \cdot U \cdot P^{-1}$ being upper triangular we conclude

$P \cdot \bar{U} \cdot P^{-1} =$ 

Our intention is now to transform $\bar{U}$ back to a permuted triangular matrix which differs from U just by one column and one row:

$P \cdot (R_2 \cdot T \cdot R_1 \cdot \bar{U}) \cdot P^{-1} =$ 

The product form of $\hat{B}_w^{-1} = \hat{U}^{-1} \cdot \hat{L}^{-1}$ is later obtained from that representation completely analogous to the Forrest-Tomlin method.

The roles of the elementary transformations R_1, T and R_2 may be described using the shape of $P \cdot \bar{U} \cdot P^{-1}$ sketched above:

- R_1 eliminatesrow s of the first term (up to the diagonal element);
- T eliminates rows 1 to s-1 of the second term;
- R_2 eliminates row s of the second term.

We shall now determine the eta vectors of R_1, T, R_2.

4.2.4 The Eta Vectors of R_1, T and R_2

R_1 differs from the unit matrix just by its z-th row w, which is supposed to have the property $w \cdot U_{\bullet j} = 0 \ \forall \ j \neq z$. Choosing

$$w := U_{zz} \cdot U^{-1}_{z\bullet} \qquad (2)$$

yields ones on the main diagonal of R_1.
(The notation $U^{-1}_{\bullet\bullet}$ is used instead of $(U^{-1})_{\bullet\bullet}$ in this section).

The eta column of T is already available in the U-file; it is the eta vector c that pivots in row z:

$$c_i = \begin{cases} 1/U_{zz} & i = z \\ -U_{iz}/U_{zz} & i \neq z \end{cases} \qquad (3)$$

The transformations already applied to $\bar{U}$ lead to the matrix

$$\bar{\bar{U}} := T \cdot R_1 \cdot \bar{U} \ ,$$

the elements of which are easily determined (using (1),(2),(3)) to be

$$\begin{array}{lll} \bar{\bar{U}}_{zj} = - v_j & & j \neq z \\ \bar{\bar{U}}_{ij} = U_{ij} & & i \neq z \neq j \\ \bar{\bar{U}}_{zz} = U^{-1}_{z\bullet} \cdot L^{-1} \cdot d & & \\ \bar{\bar{U}}_{iz} = (L^{-1} \cdot d)_i - U_{iz} \cdot U^{-1}_{z\bullet} \cdot L^{-1} \cdot d & & i \neq z \end{array}$$

The row eta $\bar{w} \neq 0$ of R_2 has to satisfy the condition $\bar{w} \cdot \bar{\bar{U}}_{\bullet j} = 0 \;\; \forall\, j \neq z.$

We define $\quad q := v \cdot U^{-1} \qquad (4)$

and choose $\quad \bar{w} := q - q_z \cdot w + I_{z\bullet} \qquad (5)$

As $\bar{w}_z = 1$, we have ones on the main diagonal fo R_2.

4.2.5 The Representation of $\hat{B}_w^{-1} = \hat{U}^{-1} \cdot \hat{L}^{-1}$

$R_2 \cdot \bar{\bar{U}}$ is now a permuted upper triangular matrix, which can be factorized (as in the Forrest-Tomlin method) into

$$R_2 \cdot \bar{\bar{U}} = C_z \cdot \tilde{U} \,,$$

where $\tilde{U}$ is obtained from U by replacing the z-th row as well as the z-th column by unit vectors, and C_z is an elementary matrix with $x := R_2 \cdot \bar{\bar{U}}_{\bullet z}$ as its z-th column.

We have

$$\hat{B}_w^{-1} = \hat{U}^{-1} \cdot \hat{L}^{-1} = (\tilde{U}^{-1} \cdot C_z^{-1}) \cdot (R_2 \cdot T \cdot R_1 \cdot L^{-1}).$$

Identifying y as the eta vector of C_z^{-1} (obtainable from x by a pivot on x_z) the claims of theorem 3.2.2 are proved.

What is really stored in the U-File is $y_z = 1/x_z$ as well as the nonzero components x_i $(i \neq z)$ of x (cf. FORREST-TOMLIN [5]).

It can be shown (cf. BASTIAN [2]) that

$$y_z = v_z/g_z \,,$$
$$x_i = h_i - f_z \cdot U_{iz} \qquad\qquad i \neq z,$$

where g_z, f_z and h_i $(i=1,..,m_1)$ are data available in Winkler's algorithmic approach.

Summarizing the computations necessary to update B_W^{-1} in case 3 of Winkler's algorithm we have

- two BTRAN-operations to compute w and q (as compared to one BTRAN-operation in case 1, if the Forrest-Tomlin method is applied);

- two multiplication of a vector by a scalar and two vector additions to compute $\bar{w}$ and x.

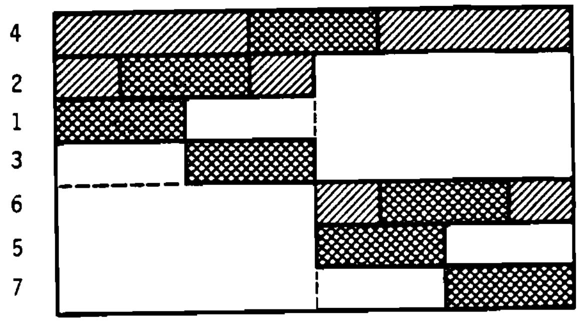
Update case 3 occurs if the sparse matrix V has nonzero elements in the pivot row p. If this row of V contains exactly one nonzero element, then the whole procedure simplifies to what is basically a standard Forrest Tomlin update: R_2 is a unit matrix and T ist not added to the L-file but rather used to modify the eta-vector of C_Z^{-1}:

$$\hat{B}_W^{-1} = (\tilde{U}^{-1} \cdot (C_Z^{-1} \cdot T)) \cdot (R_1 \cdot L^{-1})$$

Although the modification of the Forrest-Tomlin method just described was illustrated in the context of Winkler's class of algorithms, it may have other applications in situations where an inverse is frequently updated by elementary column matrices and sometimes by elementary row matrices.

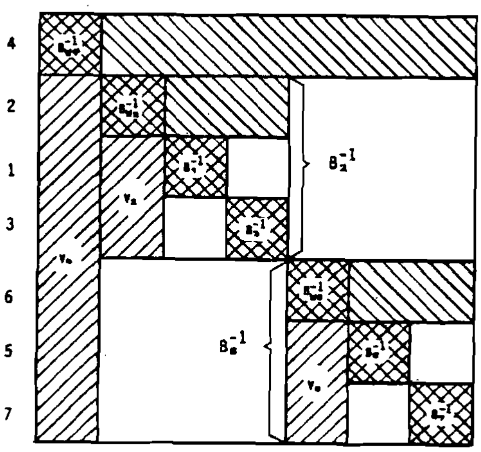
5. Staircase Systems Viewed As Nested Blockangular Systems

Any block-structured matrix may be viewed as a permuted matrix with nested blockangular structure, as we know for example from ZVIAGINA [10] and LOUTE [7].
Staircase structures are a particularly nice example. Let $k = 2^h - 1$, $h \in \mathbb{N}$; for h = 3 we have



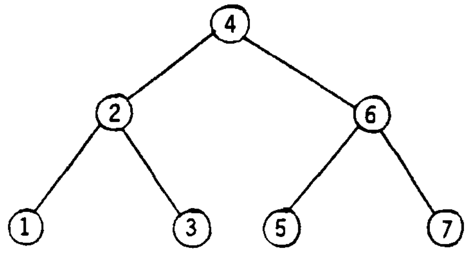
(The numbers indicate the position of a block in the original staircase structure).
Winkler showed that his factorization also extends to this nested situation, where the inverse is given by k 'subinverses' (for each i ∈ K there is one of dimension $|\alpha_i| \times |\alpha_i|$) and (k-1)/2 V-matrices.

All data that is used for a BTRAN or FTRAN operation with the basis inverse is shown in the following matrix:



Here ▩ indicates a subinverse,
▨ a V-matrix and
▧ original data

The following 'binary search tree' is the key for understanding operations with this structure:



With each leaf i we associate the inverse of a matrix B_i whose columns are drawn from block A_{α_i,γ_i} of the original staircase coefficient matrix.

Let i beanon-leaf-node having the two sons f and g. With i we associate three matrices: the inverse of a matrix B_{wi} and a V-matrix V_i which may be obtained from B_g^{-1}, B_f^{-1} and original data (as explained earlier for block-angular matrices), and a larger inverse B_i^{-1} which is given in the form of Winkler's factorization by B_{wi}^{-1} and B_g^{-1}, B_f^{-1} and V_i. Candidates for columns in B_{wi}^{-1} and V_i are original columns which have nonzeroes in at least one row $p \in \alpha_j$ where j ist a node in the subtree with root i.

Finally, $B_4^{-1} = B^{-1}$.

The main advantage of a 'divide-and conquer' - approach like this is that during FTRAN and WRETA at most $h \ll k$ of the subinverses and V-matrices (associated with a path in the tree) are needed; the same holds for the BTRAN-operation if the partial-block-pricing strategy is used. This compares to an average of $k/2$ in many other methods.

There are, however, two serious drawbacks:

- the data-handling, particularly with the V-matrices, is not simple;
- if $j \leq h$ subinverses are involved in a WRETA-operation, then $j - 1$ of them have to be updated like B_w^{-1} in case 3 discussed earlier; this amounts to a comparatively rapid growth of the eta-files.

I shall address in the next sections some of the data-handling problems.

5.1 Storing the Coefficient Matrix A

We have $K + 1$ different types of columns, type 1 having nonzeroes only in rows $p \in \alpha_1$, type i having nonzeroes in rows $p \in \alpha_{i-1} \cup \alpha_i$ $(i = 2,3,..,k)$, and type $k + 1$ having nonzeroes only in rows α_k.

The column header should contain the type of the column as well as the length of its two parts. From that information one can immediately decide whether a column has nonzeroes in a given set of rows α_j and one can read that part. In addition, one should store the starting address for the first column of each type in order to have fast access in case of partial block pricing.

(For smaller problems it may be considered to take all columns of a type into core simultaneously).

5.2 Storing the V-Matrices

As the columns that multiply a V-matrix in a BTRAN- or FTRAN-operation are expanded, there won't be any problems whether or not the V-matrix may be accessed column- or row- wise and whether the entries of an accessed vector are sorted or not.

The situation is much more complicated during WRETA. Here, the following operations may have to be performed:

(a) replace a column of V;
(b) determine whether a row p has at least one nonzero element;
(c) update all columns having a nonzero element in row p;
(d) get row p;
(e) exchange two rows .

Here (a) occurs in cases 1 and 3,
(b) occurs in cases 2 and 3,
(c),(d) and (e) occur in case 3 only.

It is very hard to decide whether column- or row-oriented access is more frequent. But as operations that may affect the length of a packed vector are confined to columns, I would suggest a column-oriented addressing scheme.

The nonzero elements of each column should be kept sorted according to their row indices. This makes operations (b), (d) and (c) considerably faster as binary search can be used. The only disadvantage would be in operation (e), where several entries of a column have to be shifted if that column has a nonzero element in exactly one of the two rows that are exchanged.

What kind of additional structure could be introduced to support row access?

The simplest one would be a bit vector whose entries correspond to the rows of V; bit i is set to 1 if row i may possibly contain a nonzero element. Whenever a nonzero is encountered in (a), (c) or (e) the corresponding bit is set to 1; it is reset to 0, if no nonzeroes have been found in that row during a(b)-operation.

Another possibility is a bit matrix which contains a 1 in position (i,j) if $V_{ij} \neq 0$.

This would yield direct access to the columns relevant during (c), (d) and (e) at the cost of more complicated update-operations (a), (c), (e) to maintain the bit matrix.

One of these approaches I would consider to be appropriate. One could of course store a column-oriented and a row-oriented representation of the V-matrix, but that would be extremely costly to maintain during operations (a) and (c).

In this context it should be pointed out that searching for a particular row index does (on the average) only have to be applied to half the number of columns of a V-matrix:
If B_i^{-1} is an inverse given by B_{wi}^{-1}, V_i, B_f^{-1} and B_g^{-1}, then no column of V_i ever has nonzeroes in rows in α_f and in α_g. Which block applies can be seen from the type (index) of the column.

6. Conclusions

It is shown that for updating an inverse with elementary column and row transformations the Forrest-Tomlin method can be used. This seems to be advantageous to do if row transformations are not likely to occur too frequently.

The class of Winkler's factorization algorithms for blockangular systems is considered to be an area of application.

When Winkler's approach is applied (in a nested way) to staircase structures, the situation is more complicated:

- The 'unpleasant' update-cases occur more frequently which makes the standard product form more competitive for about half the number of subinverses.

- Instead of one there are several sparse 'V-matrices' involved, for which row and column access is necessary.
  Different ways of storage have been discussed.

References

[1] BASTIAN, M.: "Basisfaktorisierung bei dünn besetzten strukturierten Koeffizientenmatrizen", in: Henn, R. et al. (eds.):'Operations Research Verfahren Vol. XXVIII'. Meisenheim 1978.

[2] BASTIAN, M.: "Lineare Optimierung großer Systeme: Compact-Inverse Verfahren und Basisfaktorisierungen". Mathematical Systems in Economics Vol. 55. Meisenheim 1980.

[3] DANTZIG, G.B.: "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming". Econometrica 23, 1955, 174-183.

[4] DANTZIG, G.B. and A.F. PEROLD: " A Basis Factorization Method for Block Triangular Linear Programs", Tech. Rep. SOL 78-7. Operations Research Department. Stanford University. Stanford 1978.

[5] FORREST, J.J.H. and J.A. TOMLIN: "Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method". Mathematical Programming 2, 1972, 263-278.

[6] KALLIO, M. and E.L. PORTEUS: "Triangular Factorization and Generalized Upper Bounding Techniques". Operations Research 25, 1977, 89-99.

[7] LOUTE, E.: "A Revised Simplex Method for Block Structured Linear Programs". Dissertation. Universitê Catholic de Louvain. Facultê des Sciences Appliquês. Louvain 1976.

[8] TOMLIN, J.A.: "Modifying Triangular Factors of the Basis in the Simplex Method", in: ROSE, D.J. and R.A. WILLOUGHBY (eds.): 'Sparse Matrices and their Applications'. N.Y. 1972.

[9] WINKLER, C.: "Basis Factorization for Block-Angular Linear Programs: Unified Theory of Partitioning and Decomposition Using the Simplex Method". Tech.Rep. SOL 74-9. Operations Research Department. Stanford University. Stanford 1974.

[10] ZVIAGINA, R.A.: "Prinzip Mnogostupentschatoi Dekompozitsii w Linjeinom Programmirowanii. Mathematische Operationsforschung und Statistik 4, 1973, 427-443.

SOLVING STAIRCASE LINEAR PROGRAMS BY THE SIMPLEX METHOD: 1. INVERSION*

Robert Fourer**

Systems Optimization Laboratory
Department of Operations Research
Stanford University

Proposals for improving the general simplex method have been relatively successful. As a result the simplex method has become an amalgam of fairly sophisticated algorithms. Many of these algorithms are objects of study in their own right, and are not normally thought of in connection with linear programming. The simplex method has consequently become more and more a specialist's domain.

It is therefore not surprising that study of staircase LPs has tended to diverge from study of the simplex method. Staircase linear programming has become a search for methods to replace the old simplex method; in the meantime a new, better simplex method has emerged for general linear programming but has not been applied to special structures such as staircases.

This and a companion paper [16] seek to reverse the trend: they are concerned with adapting the modern simplex method to solve staircase LPs more efficiently. Each paper looks at a set of algorithms within the simplex method: this one deals with "inversion" of the basis — more accurately, solution of linear systems by Gaussian elimination — and the succeeding one considers partial pricing.

Both papers describe extensive, although preliminary, computational experience. The results are quite promising: a staircase-adapted simplex method sometimes performs considerably better than the general method, yet on a range of problems it is never significantly worse. Moreover, further improvement appears possible in a number of respects.

*Research for this paper was partially supported by the Department of Energy Contract DE-AS03-76-SF00326 PA# DE-AT-03-76ER72018, the Office of Naval Research Contract N00014-75-C-0267, and the National Science Foundation Grants MCS76-20019 A01 and ENG77-06761. Reproduction in whole or in part is permitted for any purposes of the U.S. Government.

**Currently at:

Department of Industrial Engineering and Management Sciences
Northwestern University

INTRODUCTION

Staircase-structured linear programs (LPs) have been studied about as long as linear programming itself. Staircase LPs arose naturally from models of economic planning over time: activities were run in a series of periods, and constraints linked activities in adjacent periods. The resulting LPs, in their simplest form, had a structure like this:

$$\begin{array}{lllll}
\text{maximize} & c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_{t-1}x_{t-1} + c_tx_t & & & \\
\text{subject to} & A_{11}x_1 & & & = b_1 \\
 & A_{21}x_1 + A_{22}x_2 & & & = b_2 \\
 & \quad A_{32}x_2 + A_{33}x_3 & & & = b_3 \\
 & \qquad \ldots\ldots\ldots\ldots & & & \\
 & & & A_{t,t-1}x_{t-1} + A_{tt}x_t & = b_t
\end{array}$$

In the infancy of computers this sort of structured problem was attractive because it seemed to offer a hope of solving practical LPs in a reasonable amount of time. Thus in 1949 Dantzig observed [5] that

> ...while the general mathematical problem is concerned with maximization of a linear form of nonnegative variables subject to a system of linear equalities, in the linear programming case one finds by observing the above [staircase] system that the grand matrix of coefficients is composed mostly of blocks of zeros except for submatrices along and just off the "diagonal". Thus any good computational technique for solving programs would probably take advantage of this fact.

The simplex method was as yet impossibly slow for large general problems, but there was reason to think that a much faster version could be devised for staircase LPs.

Staircase linear programs are of no less interest today. Along with economic planning, they have found applications in production scheduling, inventory, transportation, control, and design of multistage structures [32]. Yet a recent survey [18] observes that

> the "staircase" model, in which similar sets of variables and constraints are replicated many times, seems no more tractable today then when its importance was recognized over 20 years ago. Typical of many "time-phased" economic problems, it is the standard model for numerically solving problems of optimal control. Today we know only how to solve it as we would any linear programming problem; but this type of problem requires more work to solve than does the average problem of the same size. However, there should be some way to take advantage of their simple structure.

Thus the situation has been reversed. The general simplex method is now impressively fast rather than impossibly slow, while staircase LPs are a troublesomely hard case rather than a promisingly easy one.

Proposed methods for staircase LPs

There has certainly been no shortage of attempts to solve staircase LPs more efficiently. Although the simplex method has usually been involved in some guise, individual proposals have varied considerably. The essential ideas of these proposals may be classified in four broad areas:

Compact basis methods employ a special representation of the basis or basis inverse in conjunction with a more or less standard simplex method. This approach was first suggested by Dantzig [6,8], and early variations were employed by Heesterman and Sandee [23] and Saigal [46]. More recent compact-basis schemes have been worked out by Dantzig [9], Wollmer [51], Marsten and Shepardson [35], Perold and Dantzig [42], and Propoi and Krivonozhko [43].

Nested decomposition methods apply the Dantzig-Wolfe decomposition principle to generate a series of sub-problems at each period. This approach was suggested by Dantzig and Wolfe in their original paper on decomposition [10], and has been extended or modified by Cobb and Cord [4], Glassey [19,20] and Ho and Manne [29]. (Ho has reported favorable computational results in two special cases [26,27].)

Transformation methods start with a simpler LP that can be solved easily, and work toward a solution of the original staircase LP. Varied proposals in this class are from Grinold [22], Aonuma [1], and Marsten and Shepardson [35].

Continuous methods deal with a multi-period LP in continuous rather than discrete time. Fundamentals of a simplex method for continuous-time LPs have been proposed by Perold [41].

Computational experience with most of these proposals is negligible. At present no method has proved as effective as the general simplex method in handling a wide variety of staircase problems.

Adaptation of the simplex method for staircase LPs

Proposals for improving the general simplex method itself have been, by contrast, much more successful. As a result the simplex method has become an amalgam of fairly sophisticated algorithms. Many of these algorithms are objects of study in their own right, and are not normally thought of in connection with linear programming. The simplex method has consequently become more and more a specialist's domain.

It is therefore not surprising that study of staircase LPs has tended to diverge from study of the simplex method. Staircase linear programming, typified by the above-listed papers, has become a search for methods to replace the old simplex method; in the mean time a new, better simplex method has emerged for general linear programming but has not been applied to special structures such as staircases.

This and a companion paper [16] seek to reverse the trend: they are concerned with adapting the modern simplex method to solve staircase LPs more efficiently. Each paper looks at a set of algorithms within the simplex method: this one deals with "inversion" of the basis--more accurately, solution of linear systems by Gaussian elimination--and the succeeding one considers partial pricing.

Both papers describe extensive, although preliminary, computational experience. The results are quite promising: a staircase-adapted simplex method sometimes performs considerably better than the general method, yet on a range of problems it is never significantly worse. Moreover, further improvement appears possible in a number of respects.

1. STAIRCASE LINEAR PROGRAMS

Staircase linear programs share two simple characteristics: their variables fall into some sequence of disjoint groups; and their constraints relate only variables within adjacent groups. Usually the sequence of groups corresponds to a sequence of times, so that variables in a group represent activities during one time _period_. Constraints then indicate how activities in one period are related to activities in the next. Staircase LPs thus arise especially often from many kinds of economic planning models.

A constraint is said to be in period ℓ if it contains variables of period ℓ but not of later periods. Typically some constraints involve only variables of period ℓ, while others relate variables of periods ℓ and ℓ-1; the latter are _linking_ constraints, whereas the former are _non-linking_. Analogously, linking variables appear in constraints of periods ℓ and ℓ+1, while non-linking variables appear only in constraints of period ℓ.

A staircase LP is also naturally viewed as a kind of linear discrete-time optimal control model. Typically such a model minimizes a linear function of nonnegative state vectors x_ℓ and control vectors u_ℓ, subject to dynamic equations,

$$C_\ell x_{\ell+1} = A_\ell^{(1)} x_\ell + D_\ell^{(1)} u_\ell + b_\ell^{(1)} , \qquad \ell = 1,\ldots , t$$

and control contraints,

$$0 = A_\ell^{(2)} x_\ell + D_\ell^{(2)} u_\ell + b_\ell^{(2)} , \qquad \ell = 1,\ldots, t+1$$

This is readily seen to be a staircase linear program. The state vectors are the linking variables, and the control vectors are the non-linking variables; the dynamic equations are the linking constraints, while the control contraints are non-linking.

Staircase LPs of higher orders

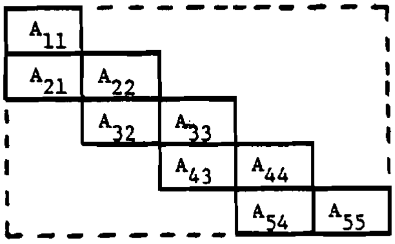
A more general approach says that a staircase linear program is of order r if its constraints relate variables that are at most r periods apart. The preceding subsection thus characterized staircase LPs of order one. Higher-order staircase LPs are not uncommon in complex applications (for example, modeling energy systems [40]). They are analogous to linear control models that have rth-order dynamic equations.

This paper is predominantly concerned with first-order staircase LPs: these have the most specialized structure and, consequently, are most amenable to special techniques. Still, many techniques are essentially applicable to higher-order staircases as well, with appropriate adaptations that will be pointed out as the exposition proceeds. For brevity, however, the adjective "first-order" will usually be dropped.

Higher-order staircase LPs can also be made into first-order ones, in either of two ways. First, rth-order equations can be transformed to equivalent first-order ones by adding certain variables and constraints. This yields a larger first-order LP that has the same number of periods. Second, every r periods of the rth-order LP may simply be aggregated as one period. The result is a first-order staircase LP of the same size but having only about t/r periods. The first method is most practical when the LP is nearly first-order to begin with, while the second may be feasible when the number of periods is large relative to r.

Staircase matrices

The matrix of constraint coefficients of a staircase linear program is a staircase matrix. Its nonzero elements are confined to certain submatrices centered roughly on and just off the diagonal--as, for example,



Formally, one partitions the rows of an m × n matrix A into t disjoint subsets, and the columns into t disjoint subsets, so that the matrix is partitioned into t^2 submatrices, or "blocks":

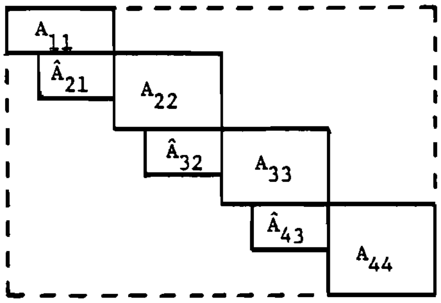
$$A_{ij}, \qquad i = 1,\ldots, t; \;\; j = 1,\ldots, t$$

A is lower staircase (as above) if $A_{ij} = 0$ expcet for $i = j$ and $i = j+1$. A is upper staircase if $A_{ij} = 0$ except for $i = j$ and $i = j-1$.

By analogy with staircase models, rows in the ith partition of a staircase matrix A are called period-i rows, and columns in the jth partition are called period-j columns. If a period-i row has nonzero elements in blocks $A_{i,i-1}$ and A_{ii}, it is a linking row; if it has nonzeroes only in A_{ii} it is a non-linking row. Similarly, a period-j column that has nonzeroes in A_{jj} and $A_{j+1,j}$ is a linking column, while one that has nonzeroes in A_{jj} only is a non-linking column.

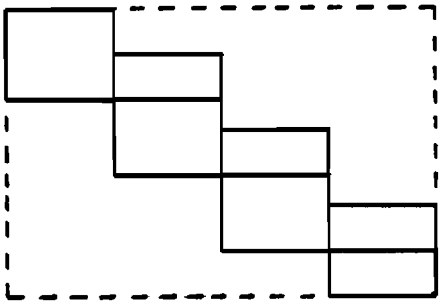
Any upper-staircase matrix may be permuted to lower-staircase form by reversing the order of the periods [15]. Moreover, if a period-i row is entirely zero within A_{ii} that row may be moved back to period i-1 without disrupting the staircase structure; analogously, a period-j column that is all-zero within A_{jj} may be moved to period j+1. Nothing is lost, therefore, in assuming that A is lower staircase and that its diagonal blocks $A_{\ell\ell}$ have no all-zero rows or columns; A is then said to be in standard staircase form. Henceforth it will be assumed that all staircase LPs have a constraint matrix A in this standard form. (The trivial case in which A has an all-zero row or column is thus ruled out.)

Following [15], the period-i rows may be permuted to put the linking rows first, and the period-j columns may be permuted to put the linking columns last. Then A has the following reduced form:



The reduced block $\hat{A}_{k,k-1}$ is just the intersection of the period-k linking rows and the period-(k-1) linking columns.

If the linking rows of every period i are switched to period i-1, then A gains an alternative row-upper-staircase form:



Switching the linking columns of period j to period j+1 gives a different, column-upper-staircase form. Thus a staircase A in reduced standard form embodies three staircases--lower, row-upper, and column-upper--each corresponding to a different choice of where the periods begin and end.

Staircase bases

Any basis B of a staircase linear program necessarily inherits a staircase structure from the constraint matrix A; B's staircase blocks, $B_{\ell,\ell-1}$ and $B_{\ell\ell}$, may be taken to be the sub-blocks of $A_{\ell,\ell-1}$ and $A_{\ell\ell}$ that contain only the basic columns. If A has a reduced form, $\hat{B}_{\ell,\ell-1}$ may likewise be taken as the basic part of $\hat{A}_{\ell,\ell-1}$.

The inherited staircase of B need not be in standard or reduced form, even though A is. Specifically, either $B_{\ell\ell}$ or $\hat{B}_{\ell,\ell-1}$ may be zero along some linking row i--if it happens that, in $A_{\ell\ell}$ or $\hat{A}_{\ell,\ell-1}$, all the nonzeroes along row i are in non-basic columns. In this event, B may be returned to reduced standard form by reassigning certain rows and columns. Any linking row that is zero in $B_{\ell\ell}$ becomes a non-linking row in period ℓ-1; in the process, some linking columns of period ℓ-1 may become non-linking. Any linking row that is zero in $B_{\ell,\ell-1}$ becomes a non-linking row.

It is generally more convenient to deal with B in its inherited staircase form, whether standard, reduced or otherwise. However, better results are often achieved by using B's reduced standard form instead, especially as it has fewer linking rows and columns and hence a tighter structure. This issue is considered further subsequently.

Henceforth $B_{\ell\ell}$ and $B_{\ell,\ell-1}$ (or $\hat{B}_{\ell,\ell-1}$) will represent the blocks of B's chosen staircase form, whether inherited or reduced standard. The number of rows in period i will be denoted m_i, and the number of columns in period j will be n_j; the respective numbers of linking rows and columns will be $\hat{m}_i$ and $\hat{n}_j$. For the row-upper-staircase form, the number of rows in period i will be m^i, and for the column-upper-staircase form the number of columns in period j will be n^j. Necessarily $\sum m_i = \sum m^i = \sum n_j = \sum n^j = m$, and $\hat{m}_i \leq m_i$, $\hat{n}_j \leq n_j$.

Balance constraints and square sub-staircases

If the staircase LP has a special dynamic Leontief structure [7] then in each period the number of basic columns must exactly equal the number of rows: $n_\ell = m_\ell$ for all ℓ, and all blocks $B_{\ell\ell}$ are square.

This is not the case in general, however. A basis B of an arbitrary staircase LP may have $n_\ell > m_\ell$ for some periods ℓ and $n_\ell < m_\ell$ for others.

Since the basis is nonsingular, however, it must obey the "balance constraints" developed in [15]. In summary, these restrict the excess of basic columns over rows in each period, individually and cumulatively, as follows:

$$0 \le \sum_1^\ell (n_i - m_i) \le \min(\hat{m}_{\ell+1}, \hat{n}_\ell), \quad \ell = 1,\ldots, t-1$$

$$-\min(\hat{m}_k, \hat{n}_{k-1}) \le \sum_k^\ell (n_i - m_i) \le \min(\hat{m}_{\ell+1}, \hat{n}_\ell), \quad k,\ell = 2,\ldots, t-1$$

$$-\min(\hat{m}_k, \hat{n}_{k-1}) \le \sum_k^t (n_i - m_i) \le 0 \qquad k = 2,\ldots, t$$

In words, the cumulative imbalance between rows and basic columns in periods k through ℓ is bounded by the smaller dimension of $\hat{B}_{k,k-1}$ and the smaller dimension of $\hat{B}_{\ell+1,\ell}$. Hence these constraints are quite strict when there are relatively few linking rows or columns.

The first constraint above may also be written as the following three inequalities:

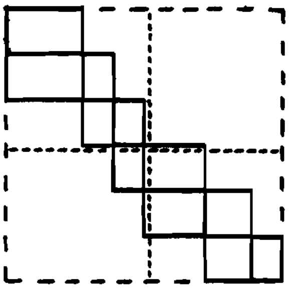
$$\sum_1^\ell n_i \ge \sum_1^\ell m_i$$

$$\sum_1^\ell n_i \le \sum_1^\ell m^i$$

$$\sum_1^\ell n^i \le \sum_1^\ell m_i$$

These say that the first ℓ periods of the lower staircase cannot have more rows than columns, while the first ℓ periods of the associated row-upper or column-upper staircase cannot have more columns than rows.

All three of these relations are equalities when $\ell = t$, since B is square. It can also happen that equality is achieved for some $\ell < t$. For example, if $\sum_1^\ell m_i = \sum_1^\ell n_i$, B must look something like this:



$$\sum_1^3 m_i = \sum_1^3 n_i$$

The rows and columns of periods 1 through ℓ form a square sub-staircase, as do the rows and columns of periods $\ell+1$ through t; they are linked only by nonzero elements in the off-diagonal block $B_{\ell+1,\ell}$. In a similar way an equality $\sum_1^\ell n_i = \sum_1^\ell m^i$ implies a pair of square sub-staircases within the row-upper staircase form, and $\sum_1^\ell n^i = \sum_1^\ell m_i$ implies the same for the column-upper form.

Generally B may exhibit any or all of these three kinds of equalities, and each may hold for several values of $\ell < t$. If p different such equalities hold, then B breaks into p+1 disjoint square sub-staircases of various kinds. The presence or absence of sub-staircases will be of importance to several of the techniques described further on in this paper.

2. SOLVING LINEAR SYSTEMS IN THE SIMPLEX METHOD

In solving linear programs by the simplex method, a great deal of computational effort is devoted to "inverting the basis". More precisely, at each iteration the simplex method solves two linear systems:

$$By = a$$

$$B^T\pi = z$$

B is the basis, an $m \times m$ matrix of basic columns of the constraint matrix A; a is a non-basic column of A; and z is an appropriately chosen "pricing form".*

There are many ways to solve such systems, but not all are suitable to practical linear programming. Typically m is in the range of several hundred to several thousand, and the simplex method generates roughly $2m$ <u>different</u> bases B. Hence only very efficient solution techniques are useful. Further, B has two very special properties:

- Successive bases are similar. Only one column of B is changed at each iteration.

- Bases are sparse. For a typical large application, less than 1% of the elements of an average B are nonzero.

The best techniques can use these properties to advantage in various ways that are outlined in this section.

*It is general practice to incorporate the linear objective function as a row of A. Then, when the basis is feasible, the pricing form z is a unit vector; when the basis is infeasible, z has one nonzero element--either +1 or -1--corresponding to each infeasible basic variable. The exact choice of z depends on details of the implementation, as explained in [39,50].

Permutation of the basis

The variables and equations of a linear system $By = a$ or $B^T\pi = z$ can be written in any order. Each ordering of the variables corresponds to some permutation of the columns of B, while each ordering of the equations corresponds to some permutation of the rows of B.

Any permutation of the rows and columns of B may be written PBQ^T, where P and Q^T are suitably chosen permutation matrices. The system $By = a$ is thus equivalent to the permuted system $(PBQ^T)(Qy) = (Pa)$. $B^T\pi = z$ is likewise equivalent to $(QB^TP^T)(P\pi) = (Qz)$.

LU factorization

At the heart of recent simplex implementations is a technique based on Gaussian elimination. The basis B is factored as the product of a lower-triangular matrix L, and an upper-triangular matrix U. Once $B = LU$ is known, the linear systems of importance reduce to

$$L(Uy) = a$$
$$U^T(L^T\pi) = z$$

Then y or π may be found through solving two triangular systems by back-substitution.

In practice Gaussian elimination is applied to a chosen permutation PBQ^T. Choice of P and Q^T is a crucial matter, as can be seen by considering the computation involved in elimination. Its essential operations are defined by the following recursion:

$$\beta^{(1)} = PBQ^T$$

$$\beta_{ij}^{(k+1)} = \beta_{ij}^{(k)} - \beta_{ik}^{(k)}\beta_{kj}^{(k)}/\beta_{kk}^{(k)}, \qquad i, j > k; \quad k = 1,\ldots, m-1$$

of which L and U are a by-product:

$$L_{ij} = \beta_{ij}^{(j)}/\beta_{jj}^{(j)}, \qquad i \geq j$$

$$U_{ij} = \beta_{ij}^{(i)}, \qquad i \leq j$$

The critical values are the "pivots" $\beta_{kk}^{(k)}$: an LU factorization exists if and only if all pivots are nonzero. Moreover, elimination is numerically stable only if all picots are sufficiently large in magnitude, both absolutely and relative to other elements of $\beta^{(k)}$.

As a consequence, practical Gaussian elimination looks for permutations P and Q^T such that PBQ^T has an acceptably large series of pivots. Choosing P and Q^T is thus commonly called "pivot selection".

Once L and U are computed, solving the resulting triangular systems presents no difficulty. Back-substitution in these systems is an inherently fast and stable process.

The jargon of LP computer codes refers to solution of a lower-triangular system as an FTRAN ("forward transformation"); solution of an upper-triangular system is a BTRAN ("backward transformation"). Solving $L(Uy) = a$ thus requires first an FTRANL and then a BTRANU, while solving $U^T(L^T\pi) = z$ requires an FTRANU and a BTRANL.

Updating the LU factorization

Just as successive bases are similar, their LU factorizations are similar. Consequently it is practical to merely update L and U at each basis change, rather than compute the factorization from scratch each time.

The idea of an LU update is as follows. Suppose the initial basis, B_0, has been factored as $P_0B_0Q_0^T = L_0U_0$. Thus $B_0 = (P_0^TL_0)(U_0Q_0)$: B_0 is the product of a permuted lower-triangular matrix and a permuted upper-triangular matrix. Equivalently, $(P_0^TL_0)^{-1}B_0 = U_0Q_0$.

Now update B_0 to a new basis B_1, and consider

$$(P_0^TL_0)^{-1}B_1 = \tilde{U}_0Q_0 \qquad (1)$$

$\tilde{U}_0$ need not be upper-triangular; however, it does have an LU factorization, $\tilde{U}_0Q_0 = (P_1^TL_1)(U_1Q_1)$. Substituting into (1) and rearranging shows that

$$B_1 = (P_0^TL_0)(P_1^TL_1)(U_1Q_1) \qquad (2)$$

Thus B_1 is factored as the product of two permuted lower-triangular matrices and a permuted upper-triangular matrix. Linear systems involving B_1 are then readily solved as before, but with the addition of some back-substitutions in L_1.

Similar updates can be applied at subsequent basis changes. After k iterations, the basis B_k is factored at

$$B_k = (P_0^TL_0)(P_1^TL_1)\cdot\cdots\cdot(P_k^TL_k)(U_kQ_k) \qquad (3)$$

FTRANL and BRRANL perform back-substitutions with L_0 through L_k, while FTRANU and BTRANU use U_k.

LU updating in this way is practical because B_1 differs from B_0 in only one column. Hence $\tilde{U}_0$ is nearly upper-triangular--it differs from U_0 in only one column--and, as a result, U_1 is much the same as U_0, while L_1 is not much different from the identity. The factorization (2) is.thus fairly easy to find and record, and the subsequent back-substitutions are only marginally more expensive than for B_0. Further updates are equally economical, and may continue until the cost of back-substitution in (3) begins to rise appreciably--typically after 50 to 100 iterations. A fresh LU factorization of the basis is then computed, and updating begins anew.

Specific algorithms for LU updates differ primarily in their choice of permutations P_1 and Q_1 for the factorization $U_0Q_0 = (P_1^TL_1)(U_1Q_1)$. The original algorithm of Bartels and Golub [2,3] was designed to ensure numerical stability. Subsequent variations have given more weight to storage arrangement [14,47] or sparsity [17,44].*

*Another technique, proposed by McBride [36], promises an especially sparse update. Essentially, it usesas B_1 acarefully updated and permuted B_0, with the result that the product $(P_0^TL_0)(P_1^TL_1)$ may be collapsed to a single lower-triangular factor; in effect this technique updates the lower-triangular factor at each iteration, whereas the other techniques merely augment it. McBride avoids Gaussian elimination in his implementation, however, preferring to keep the inverse of one small matrix explicitly.

Storing the LU factorization

To benefit from sparsity, an LP code must store only the nonzero elements in matrices such as A, L and U. The total storage required by a sparse problem is thereby drastically curtailed; indeed, large-scale linear programming would be impossible if all zeroes had to be stored. Moreover, sparse storage makes possible efficient pricing and pivoting routines that automatically skip multiplying and adding zeroes.

Because bases are subsets of the columns of A, it is universal practice to store A by column. Typically one array lists the nonzero elements of A in column order, a parallel array lists the row index for each element, and a shorter third array indicates where each column begins in the first two arrays. A basis is represented by just a list of the basic columns.

To factorize a basis B stored in this way, it may be efficient to rearrange the operations of Gaussian elimination so that only one column, b_j, is processed at a time. An LU factorization of PBQ^T is then computed by essentially the following algorithm:

```
1: SET $L = U = I$
2: REPEAT for each column $b_j$ of $BQ^T$:
    2.1: SOLVE $Lx = Pb_j$ for $x$
    2.2: SET $U_{ij} = x_i$ for $i = 1,\dots, j$
    2.3: SET $L_{ij} = x_i/x_j$ for $i = j+1,\dots, m$
```

L and U are produced one column at a time, and so may be stored like A as columnwise lists of nonzeroes. FTRAN operations read forward through these lists, whereas BTRAN operations start at the end of a list and read backward to the beginning. (Hence the terms FTRAN and BTRAN.)

In practice the storage arrangement of L and U is closely tied to the updating technique. Any of the previously-mentioned techniques may store L columnwise, since it is just augmented (by $P_k^T L_k$) at each iteration. Only the Forrest-Tomlin technique, however, can be adequately implemented with U stored columnwise. Saunders' technique requires row-wise access as well to a (hopefully small) part of U, while Reid's technique is only practical with row-wise access to all of U. Thus these latter techniques have been implemented with various alternative storage schemes for U: Saunders has stored part of U explicitly [47], while Reid has experimented both with linked lists and with a combination of row-wise and column-wise arrays [45].

There are important advantages to storing L and U by column only. Column-wise storage is simple and compact; the associated FTRAN and BTRAN routines are also simple and L and U may be held on any sequential storage device. In a virtual-machine environment, sequential storage also minimizes the danger of "thrashing"—excessive overhead cost that results from trying to access too many widely-separated parts of storage in a short interval of time. On the other hand, if storage is at a premium one may take further advantage of "triangle" columns--those that are zero above the diagonal of PBQ^T; a triangle column is essentially trivial in U and unchanged in L, and so may be represented in L by just a pointer into A.

Access to U by column only does have its disadvantages, however. It restricts updating to the Forrest-Tomlin technique which, while usually adequate, is inferior to other techniques in numerical stability and sparsity. In addition, it suffers from certain inefficiencies in applying FTRAN and BTRAN to sparse vectors, as explained further below.

Sparse LU factorization

It is well-known [11,12,13] that when B is sparse, some of its permutations have much sparser L and U factors than others. Consequently all LP codes implement some form of sparse Gaussian elimination in which pivots are chosen to promote sparsity of L and U as well as numerical stability.

There are principally two techniques of sparse Gaussian elimination employed in linear programming. Bump-and-spike techniques look for a block-triangular permutation of B that has many small blocks ("bumps") and few columns ("spikes") that extend above the diagonal. Local-minimization techniques choose each pivot to minimize the estimated number of nonzeroes added to L and U by that pivot alone. These ideas are described and compared in Section 1 of [15].

Each technique of sparse elimination is best suited to certain updating techniques. Saunders' update relies on there being relatively few spikes in U, and so it has been implemented with bump-and-spike elimination. Reid's update, by contrast, benefits when nonzeroes fall more heavily in U than in L, and is well-suited to elimination by local minimization.

As noted previously, update techniques can also be designed to promote sparsity in the updated factors L_k and U_k. Reid's update in particular is intended to preserve sparsity, and Gay has also incorporated Reid's ideas in Saunders' technique.

Sparse right-hand side vectors

The linear systems of the simplex method, $By = a$ and $B^T\pi = z$, usually have not only a sparse matrix but a very sparse right-hand side: a is a column of the sparse matrix A, and the pricing form z has one nonzero when the basis is feasible and k nonzeroes when there are k infeasibilities. FTRAN and BTRAN routines can take advantage of this additional sparsity to a certain extent, depending on how they access L and U.

For purposes of illustration, consider first a simple lower-triangular system $Lx = d$. If the nonzero elements of L are available sequentially by column, back-substitution is carried out as follows:

```
FTRANL:
REPEAT FOR $j$ FROM 1 TO $m$:
    SET $x_j = d_j/L_{jj}$
    REPEAT FOR $L_{ij} \neq 0$, $i$ FROM $j+1$ TO $m$:
        SET $d_i = d_i - L_{ij}x_j$
```

At the jth pass through the main loop, if $d_j = 0$ then also $x_j = 0$ and the inner loop merely adds zero to various elements of d. Hence the jth pass is superfluous when $d_j = 0$. Moreover, if it happens that $d_1, \ldots, d_k$ are all zero, then the main loop does no work until pass $k+1$. A more efficient algorithm is thus as follows:

FTRANL:

```
1: SET $k = \min\{j : d_j \neq 0\}$; SET $x_j = 0$ for $j = 1,\ldots, k$
2: REPEAT FOR $j$ FROM $k+1$ TO $m$:
       IF $d_j = 0$: SET $x_j = 0$
       ELSE: SET $x_j = d_j/L_{jj}$
             REPEAT FOR $L_{ij} \neq 0$, $i$ from $j+1$ TO $m$:
                 SET $d_i = d_i - L_{ij}x_j$.
```

Step 1 is especially valuable when $d_1,\ldots, d_k$ are known beforehand to be zero. In step 2, d tends to fill in with nonzeroes in each pass of the loop; but if L and d are both sparse then d should not fill in too quickly.

The situation is quite different if instead one must solve the upper-triangular system $L^T x = d$. If the nonzeroes of L are only available sequentially by column, then L^T is effectively available only by row, and back-substitution must be carried out as follows:

BTRANL:

```
REPEAT FOR $j$ FROM $m$ TO 1:
    REPEAT FOR $L_{ij} \neq 0$, $i$ FROM $m$ to $j+1$:
        SET $d_j = d_j - L_{ij}x_i$
    SET $x_j = d_j/L_{jj}$
```

Here there is no advantage to knowing $d_j = 0$, since d_j is continually modified within the inner loop and x_j is not set until after the inner loop. The most one can say is that, if $d_m,\ldots, d_k$ are all zero, then $x_m,\ldots, x_k$ are also all zero and the main loop may be started with $j = k-1$.

For sparse elimination with updating the situation is somewhat more complex, involving not one L but a series of permuted L's. The conclusions are the same, however: if the lower-triangular factors of the basis are stored by column only—as they commonly are—then FTRANL can benefit from sparsity in the right-hand side to a much greater extent than BTRANL. Moreover, the same reasoning can be applied to U: if all or part of the upper-triangular factor is stored by column only, then BTRANU can exploit right-hand side sparsity much more than FTRANU.

In practice these differences have various consequences. At a typical iteration, the FTRAN and BTRAN operations are carried out once each, to solve systems that look like these:

TO SOLVE $By = a$:

FTRANL: $(P_0^T L_0)(P_1^T L_1)\cdot \cdots \cdot(P_k^T L_k)\ y^{(1)} = a$

BTRANU: $(U_k Q_k)\ y = y^{(1)}$

TO SOLVE $B^T\pi = z$:

FTRANU: $(Q_k^T U_k^T)\pi^{(1)} = z$

BTRANL: $(L_k^T P_k)\cdot \cdots \cdot(L_1^T P_1)(L_0^T P_0)\pi = \pi^{(1)}$

Hence sparsity of the right-hand side can be exploited in the following ways:

FTRANL can fully exploit the sparsity of a. A small additional advantage can be had if it is known that $(P_0 a)_1, \ldots, (P_0 a)_i$ are all zero for some i; this knowledge is not readily available in the general case, but it is often available from staircase methods to be described.

BTRANU can fully exploit any sparsity in $y^{(1)}$. Since $y^{(1)}$ is the solution vector from a sparse FTRANL, it may well be sparse itself.

FTRANU can exploit the considerable sparsity in z only if either U_k is available by row, or $(Q_k z)_1, \ldots , (Q_k z)_i$ are all zero from some i. In many cases it is possible to arrange that i is quite close to m [21]. Indeed, with some updating methods it can be guaranteed--provided the basis is feasible--that $(Q_k z)_1, \ldots , (Q_k z)_{m-1}$ are all zero, so that FTRANU may effectively be skipped.

BTRANL generally cannot benefit from sparsity in $\pi^{(1)}$. However, the update factors $L_1, \ldots , L_k$ are generally so simple in form that BTRANL handles them as efficiently as FTRANL. The significant extra work lies entirely in processing L_0^T.

Partial solutions

It is evident from the preceding analysis that the solution to $By = a$ or $B^T\pi = z$ is ultimately computed one element at at time, regardless of how L and U are stored. The vector y is produced by BTRANU in the order $(Q_k y)_m, \ldots , (Q_k y)_1$; likewise, the vector π is computed by BTRANL in the order $(P_0\pi)_m, \ldots , (P_0\pi)_1$.

BTRANL or BTRANU may therefore be terminated prematurely if only part of y or π needs to be computed. Such a partial solution has two potential uses in linear programming: when the rest of y is known to be zero, and when only a portion of π is required for pricing in the current iteration.

Nevertheless, in the general case there is little to be gained from trying to compute partial solutions, owing to the presence of permutations P_0 and Q_k: there is no efficient way to tell whether all remaining elements of $Q_k y$ are zero, or to predict which elements of $P_0 \pi$ will be needed. Section 4 will show, however, that partial solutions can offer an economy in solving staircase LPs, provided P_0 and Q_k are chosen to reflect the staircase structure.

3. SPARSE ELIMINATION OF STAIRCASE BASES

Two techniques for sparse elimination of staircase matrices were proposed in [15]: one adapts the bump-and-spike approach, while the other is a kind of local minimization. Either of these techniques may be applied to the staircase bases that arise from staircase LPs in the simplex method.

This section summarizes the direct effects--on speed, storage, and sparsity--of substituting staircase elimination techniques for standard ones in a simplex LP code. Section 4 then shows how these staircase techniques make possible additional efficiencies in the FTRAN and BTRAN routines.

Bump-and-spike techniques

The standard bump-and-spike technique [24,25] is a two-step procedure. First it determines the block-traiangular reduction of the basis B, an essentially unique permutation that puts B in block-triangular form with as many diagonal blocks ("bumps") as possible. Second, each diagonal block larger than 2 × 2 is further permuted by the Preassigned Pivot Procedure (P3), a heuristic that tries to make each block lower triangular except for a small number of "spike" columns that extend above the diagonal. Permuted in this way, B has a good structure for sparse Gaussian elimination: fill-in (creation of new nonzeroes during elimination) is confined to the spike columns, and pivots within a given bump cannot give rise to fill-in within other bumps.

A proposed staircase bump-and-spike technique [15] dispenses with block-triangular reduction, and uses instead the staircase form of the basis. The heuristic P3, adapted to handle blocks that are non-square or rank-deficient, is applied in turn to each of the diagonal blocks $(B_{\ell\ell})$ of the staircase. Thus the rows of period 1 are assigned to pivot first, followed by the rows of period 2, period 3, and so forth through period t. The columns are also generally pivoted in period order, but "interperiod spikes" from certain periods are pivoted in later periods in order to square off the oblong staircase blocks. Thus fill-in is confined to two kinds of spikes--intraperiod spikes found by P3, and interperiod spikes assigned to square off diagonal blocks--and pivots within a given period can only give rise to fill-in within spikes of the same period or within interperiod spikes of preceding periods. The balance constraints of Section 1 guarantee that this is a workable arrangement: the number of interperiod spikes need not be very large, and there are always enough interperiod spikes to square off every staircase block.

Computational experience [15] has shown that the standard and staircase bump-and-spike techniques are roughly comparable. They usually produce about the same number of spikes, and both yield a sparse factorization: the fill-in due to either technique is seldom more than twice the fill-in due to the other. However, each technique does appear to be superior in certain situations.

Standard bump-and-spike seems invariably better when all bumps are small and most are 1 × 1. P3 is then applied cheaply to a few blocks, whereas the staircase technique must still apply P3 to every diagonal block of the staircase. The interperiod spikes of the staircase technique also tend to be larger than the spikes of the standard technique, and so the former fill in more: fill-in within L tends to be about the same, but the standard technique produces a notably sparser U. In addition, the standard technique is less prone to producing spikes that have unacceptable pivot elements, and so less time is wasted in "spike-swapping" during the elimination.

Staircase bump-and-spike has the advantage when there are one or two very large bumps that comprise half or more of the rows and columns of B. P3 becomes highly inefficient in processing these large bumps. Fill-in within U is comparable, while the staircase technique yields a sparser L. Moreover, the staircase technique produces substantially fewer spikes that have unacceptable pivots.

Storage requirements vary somewhat with the size of the largest block that must be processed, but are moderate in any case. Since a pivot order is fully chosen prior to elimination, storage required by the bump-and-spike heuristics may later be used to hold part of L and U.

Local-minimization techniques

Standard local-minimization techniques dynamically choose the kth pivot element from the remaining uneliminated matrix, $\beta^{(k)}$. The chosen pivot minimizes some "merit" function over all nonzero elements of $\beta^{(k)}$ that meet certain numerical tolerances. Practical merit functions are computed from two sets of values: $r_i^{(k)}$, the number of nonzeroes in row i of $\beta^{(k)}$, and $c_j^{(k)}$, the number of nonzeroes in column j of $\beta^{(k)}$.

Local minimization was first suggested by Markowitz [34], who proposed that the merit of element (i,j) be $(r_i^{(k)} - 1)(c_j^{(k)} - 1)$; no substantially better merit function has been found since.

Proposed staircase local-minimization techniques [15] differ by limiting the minimization to roughly one period of $\beta^{(k)}$ at a time. As a consequence both the rows and columns of B are pivoted in period order. It can also be shown that fill-in is limited to a small part of $\beta^{(k)}$--roughly two periods or less--while the remainder of $\beta^{(k)}$ is just the same as B.

Staircase local-minimization offers clear economies in both execution time and storage space. All of the work at the kth pivot--minimizing the merit function, updating $\beta^{(k)}$ to $\beta^{(k+1)}$, and updating $r_i^{(k)}$, $c_j^{(k)}$--is confined to the rows and columns of one or two periods, whereas the standard technique must deal with the entire $\beta^{(k)}$. Storage is required only for the part of $\beta^{(k)}$, also one or two periods, that differs from B.

For large problems of many periods, the differences in required storage may be immense. As a result, staircase local-minimization may be able to use simpler or more efficient storage strategies than standard local-minimization. During elimination by the standard technique the uneliminated $\beta^{(k)}$ shrinks while L and U grow; thus some sort of dynamic storage allocation is necessary when $\beta^{(k)}$, L and U are too large to be stored fully together. By contrast, under the staircase technique the active part of $\beta^{(k)}$ is small and fairly constant in size, and might well be kept in a fixed work area.

Standard local minimization does seem to usually produce a sparser L and U, as might be expected: it conducts its minimization over a much greater number of potential pivots. In the worst case in [15] the staircase technique produced about twice the fill-in (47% vs 22%); in some cases it did nearly as well, however, and in one it was distinctly better.

Comparison of techniques

Choice of a sparse-elimination technique cannot be separated from choice of an updating method (as explained previously), and both choices are sensitive to the nature and availability of storage. Consequently it is impossible to recommend one class of techniques--bump-and-spike or local-minimization--over the other categorically. Each may have its place in certain situations.

Indeed, the evidence of [15] suggests that every technique outlined in this section (standard and staircase bump-and-spike, standard and staircase local-minimization) offers the lowest fill-in for certain bases. Either of the staircase techniques should be acceptably fast, and all but the standard local-minimization have unproblematical storage requirements.

Staircase bump-and-spike techniques apply just as well to higher-order staircases. Staircase local-minimization might also be adapted to handle higher-order problems, but the extent of fill-in would be greater and hence the savings would be less.

4. SOLVING LINEAR SYSTEMS WITH STAIRCASE BASES

Both proposed staircase elimination techniques order their row pivots by period: all rows in period 1 are pivoted first, then all rows in period 2, and so forth. Staircase local-minimization also orders all column pivots by period, as does staircase bump-and-spike with the exception of certain columns (the interperiod spikes) that pivot after other columns of later periods.

This section describes how these staircase pivot orders can be taken advantage of to make the FTRAN and BTRAN routines more efficient. A partition of the L and U factors by period is first defined more formally, after which each solution routine—FTRANL, BRRANU, FTRANU, BTRANL—is taken up in turn.

Period partitions of the L and U factors

In the notation of Section 2, the basis B at an arbitrary iteration is factored as

$$B = (P_0^T L_0)(P_1^T L_1) \cdot \cdots \cdot (P_k^T L_k)(U_k Q_k)$$

In terms of this factorization and the staircase constraint matrix A, one may define the following indices for any period ℓ:

λ_ℓ first row of $P_0 B$ whose corresponding row of A is in period ℓ or later

μ_ℓ first column of BQ_k^T from period ℓ or later of A.

Necessarily $\lambda_\ell \leq \lambda_{\ell+1}$, $\mu_\ell \leq \mu_{\ell+1}$ for <u>any</u> factorization as above. Thus $\{\lambda_1, \ldots, \lambda_t\}$ and $\{\mu_1, \ldots, \mu_t\}$ partition the rows and columns, respectively, of $P_0BQ_k^T$ by period. Since the rows of $P_0BQ_k^T$ correspond to the rows of L_0, the λ's can also be thought of as partitioning L_0; analoguously, the μ's partition U_k.

In general these partitions are not particularly useful, as the λ's and μ's all tend to be small. In an extreme case, for example, if the first row of P_0B is a period-t row then $\lambda_1 = \cdots = \lambda_t = 1$. It is thus necessary to show that the staircase pivoting techniques yield worthwhile partitions whose λ's and μ's are more or less evenly spread out.

Consider first a factorization with no updates, $P_0BQ_0^T = L_0U_0$. Certainly the staircase techniques, applied to the staircase structure that B inherits from A, yield good partitions. Either technique yields $\lambda_\ell = \sum_1^{\ell-1} m_i + 1$. For bump-and-spike $\mu_\ell \geq \lambda_\ell$, and $\mu_\ell = \lambda_\ell$ if there are no all-zero rows in $B_{\ell\ell}$; for local minimization, $\mu_\ell = \sum_1^{\ell-1} n_i + 1$.

The situation is slightly more complicated if, as suggested in Section 1, B is put in reduced standard staircase form before the staircase pivoting techniques are applied. Some rows of B that correspond to period-ℓ rows of A may then be pivoted as if they were in period ℓ-1. As a consequence, one can say only that $\sum_1^{\ell-2} m_i + 1 \leq \lambda_\ell \leq \sum_1^{\ell-1} m_i + 1$; the λ's may be smaller, and the λ-partition less regular. Nevertheless, the λ's are still well spaced and constitute a useful partition, particularly if the periods are small and numerous.

As B changes and the factorization is updated, L_0 and the λ-partition are unchanged. U_0 is updated to U_k, however, and in the process the μ-partition is altered. Specifically, all of the common update methods have the following action: a column of BQ_{k-1}^T is deleted, and a new column is inserted at some point after the deleted column to produce BQ_k^T. The μ-partition up to the deleted column and after the inserted column is therefore unchanged; but if μ_ℓ is between the two columns then its value drops by 1. The μ-partition is thus slowly degraded. Degradation should not be severe, however, for large LPs with the usual 50-100 updates between refactorizations.

It may be concluded, then, that staircase pivot-selection techniques do yield λ's and μ's that constitute non-trivial partitions of L and U by period.

Staircase FTRANL

At each iteration FTRANL starts by solving a system like $(P_0^T L_0)x = a$, or equivalently $L_0 x = P_0 a$, where a is a column of A. If a is from period ℓ, then it is zero on rows of periods 1 through ℓ-1. Consequently,

$$(P_0 a)_i = 0 \ , \qquad\qquad i = 1,\ldots,\lambda_\ell - 1$$

and the main loop of the FTRANL routine may begin at index λ_ℓ as explained in Section 2.

In short, when FTRANL transforms a period-ℓ column it can start at the ℓth period in L_0, rather than at the beginning. The resultant savings will be small, however, since FTRANL already handles right-hand side zeroes efficiently.

Further savings might be possible if one kept track of upper-sub-staircases of B_0, as described in Section 1. The idea is as follows:

if B_0 has an upper-sub-staircase in periods 1 through ℓ, and if a lies in period ℓ or earlier, then the solution x of $(P_0^T L_0)x = a$ is zero in periods $\ell+1$ and later. Thus the main loop of FTRANL may be terminated prematurely. As a practical matter, however, the logic of such a scheme is fairly complex, and computational experiments [15] have shown only a moderate number of upper-sub-staircases; so the potential savings are probably not worth the trouble.

Staircase BTRANU

At each iteration BTRANU solves a system like $(U_k Q_k)y = x$, where x is a solution vector from FTRANL. Since FTRANL has solved with $L_0, L_1, \ldots, L_k$, there is no telling where zeroes may be in x. Hence BTRANU cannot benefit specially from a sparse right-hand side.

A small saving is possible, however, if the location of (lower) square sub-staircases in B is known. Suppose that the linear system at hand is $By = a$, that a is from period j, and that B has a sub-staircase at period $\ell < j$ (that is, $\sum_1^\ell m_i = \sum_1^\ell n_i$). Then the system can be partitioned as

$$\left[\begin{array}{c|c} B^{(11)} & 0 \\ \hline B^{(21)} & B^{(22)} \end{array}\right] \left[\begin{array}{c} y^{(1)} \\ \hline y^{(2)} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline a^{(2)} \end{array}\right]$$

where $B^{(11)}$ and $B^{(22)}$ are the square sub-staircases. Clearly the solution must have $y^{(1)} = 0$, $y^{(1)}$ being just the part of y that corresponds to the columns of B in periods 1 through ℓ.

Now if $By = a$ is written instead as $(BQ_k^T)(Q_k y) = a$, the preceding statement is equivalent to the following: an element of $Q_k y$ will be zero if it corresponds to a column of BQ_k^T in periods $1,\ldots,\ell$. That is,

$$(Q_k y)_i = 0, \qquad i = 1,\ldots,\mu_\ell - 1$$

Thus the main loop of BTRANU, which computes $(Q_k y)_i$, $i = m,\ldots, 1$, can stop after the μ_ℓ-th pass; the remainder of the solution is zero.

Staircase FTRANU

FTRANU solves at each iteration a system like $(U_k Q_k)^T x = z$, or $U_k^T x = Q_k z$, where z is a pricing form chosen in one of several ways (see Section 2). Usually most of z is zero, and often it can be determined that z is zero in all columns of the first ℓ periods of the basis; during Phase I of the simplex method, for example, this would occur if all basic variables of the first ℓ periods were feasible. It would then follow that

$$(Q_k z)_i = 0, \qquad i = 1,\ldots,\mu_\ell - 1$$

and the main loop of FTRANU could begin at μ_ℓ as explained in Section 2.

This result is analogous to the one for FTRANL above: when FTRANU transforms a z that is zero prior to period ℓ, it can start at the ℓth period in U_k rather than at the beginning. However, the potential savings are greater since--if U_k is stored only by column--FTRANU cannot normally benefit from sparsity in z. In practice the savings depend on how U is actually stored and on how z is handled.

Staircase BTRANL

BTRANL produces a vector π that is employed in "pricing" nonbasic columns of A; specifically, each iteration computes numerous inner products $\pi^T a$ with columns a. If a is from period ℓ then it is zero except on rows of periods ℓ and $\ell+1$, and so only the elements of π that correspond to these periods are needed to form $\pi^T a$. Since the simplex method seldom considers all nonbasic columns at one iteration, it can be arranged that only certain periods of π are needed. (See [16] for a more extensive explanation.)

Assume, therefore, that at the current iteration one only needs elements of π corresponding to rows of periods ℓ and later. The vector π is the solution of $B^T \pi = z$, or $(P_0 B)^T (P_0 \pi) = z$. Thus, equivalently, one needs only elements of $P_0 \pi$ that correspond to rows of $P_0 B$ in periods ℓ and later. It will suffice, therefore, to compute $(P_0 \pi)_i$, $i = \lambda_\ell, \ldots, m$.

BTRANL actually produces the elements of π by solving $(P_0^T L_0)^T \pi = x$, or $L_0^T (P_0 \pi) = x$, where x has been obtained from preceding transformations of z in FTRANU and BTRANL. Each pass through BTRANL computes another element of $P_0 \pi$, in reverse order: $(P_0 \pi)_m, \ldots , (P_0 \pi)_1$. Thus to compute the desired part of π one need only run BTRANL through the λ_ℓth pass of the main loop; the remainder may be skipped.

The potential savings in this instance are considerable. Using one of the partial-pricing schemes of [16] substantial amounts of computation may be avoided, on the average, at each iteration. This is especially important as BTRANL is one of the less efficient transformations, being unable to take advantage of right-hand side sparsity when L_0 is stored in the usual columnwise fashion.

5. COMPUTATIONAL EXPERIENCE

This section reports on initial computational experiments with some of the preceding ideas. The results indicate that staircase adaptation of the simplex method does make a significant difference: generally much less time is spent in certain routines, while more time is spent in others. Overall the staircase runs were measurably faster, and in one case the savings were quite substantial. Moreover, it appears there is still room for improvement in subsequent implementations.

For the test runs an existing LP code, MINOS [38,48], was modified to recognize staircase structure and to apply optionally the staircase techniques of Sections 3 and 4. Each test LP could then be solved twice —once with the staircase features turned off, once with them on—and the results could be meaningfully compared. Details of the test code and the experimental setup are given in Appendix B.

MINOS employs a bump-and-spike factorization with Saunders' updating technique. Consequently the staircase bump-and-spike technique was implemented in the test version, and all test results bear directly only upon bump-and-spike methods. Nevertheless, from certain results one may make quite favorable speculations about the expected performance of staircase local-minimization techniques, as described further below.

To keep the presentation compact, only short tables of results are presented in this section. Graphs of more extensive test data are collected in Appendix C.

Overall results

Seven medium-to-large-scale linear programs were used in the tests. All are from applications, and are of dissimilar structures (aside from being staircase). Their dimensions are as follows:

	PERIODS	ROWS	COLUMNS	NONZERO COEFFICIENTS	ITERATIONS TO SOLVE FROM SLACK START
SCAGR25	25	472	500	2208	1058
SCRS8	16	491	1169	4106	862
SCSD8	39	398	2750	11,349	2047
SCFXM2	8	661	914	5466	1012
SCTAP2	10	1101	1880	13,815	1174
PILOT	9	723	2789	9291	>2000
BP1	6	822	1571	11,414	>2000

For the sake of economy, PILOT and BP1 were tested on runs of 1000 and 750 iterations, respectively, starting from advanced bases. The rest were run to optimality from an all-slack start. Additional information about the test LPs is collected in Appendix A, and Appendix B explains in more detail how they were solved.

Raw results from the test runs, standardized to seconds per 1000 iterations, were as follows:

	TOTAL TIME		
	STANDARD	STAIRCASE	% CHANGE
SCAGR25	29.7	27.9	- 6%
SCRS8	33.9	31.5	- 7%
SCSD8	43.2	37.8	-13%
SCFXM2	43.4	42.2	- 3%
SCTAP2	67.2	67.1	0%
PILOT	155.7	106.4	-32%
BP1	181.8	189.7	+ 4%

Savings were substantial for PILOT, and respectable for SCSD8. For the others the gross difference between the standard and staircase techniques was small, though the latter performed worse only on BP1.

It is misleading to consider only these totals, however. When the times are broken down by function—as in the first set of graphs in Appendix C--it can be seen that gains in some areas tend to be offset by losses in others. The staircase version has an edge in simplex pricing and pivoting, while it is usually slightly behind in updating the LU factorization; it ranges from much faster to somewhat slower in pivot selection for Gaussian elimination, but is almost always slower in computing the L and U factors. Miscellaneous routines consume a good 10-20% of the time, much of which could be saved in practical (rather than test) circumstances.

Thus much more is to be learned by examining the times of individual routines and functions. The following subsections consider first the simplex-iteration routines, and then the LU-factorization ones.

Iterating routines

The simplex method spends a majority of its time in tasks that are repeated at each iteration: choosing a column to enter the basis (pricing), determining which column leaves the basis (pivoting), and revising the basis factorization accordingly (updating). The LP code's "iterating" routines carry out these tasks.

For the test problems, total time spent in the iterating routines --again, normalized to seconds per thousand iterations--was as follows:

	ITERATING TIME		
	STANDARD	STAIRCASE	% CHANGE
SCAGR25	24.6	22.2	-10%
SCRS8	28.1	23.8	-15%
SCSD8	34.2	30.5	-11%
SCFXM2	33.2	32.5	- 2%
SCTAP2	56.9	54.3	- 5%
PILOT	108.0	86.3	-20%
BP1	136.6	146.1	+ 7%

Here the results are somewhat more striking, four of the seven showing savings of 10-20%.

Again more can be learned from a further breakdown of the times, given by the second set of graphs in Appendix C. The greatest difference by far is in BTRANL, which is significantly faster with the staircase version in every instance. There is a corresponding, but smaller, efficiency in FTRANL. The figures for these two routines are as follows:

	FTRANL			BTRANL		
	STD	STAIR	% CHNG	STD	STAIR	% CHNG
SCAGR25	2.7	1.9	-29%	6.7	3.5	-48%
SCRS8	2.4	1.5	-36%	5.7	3.4	-41%
SCSD8	3.9	2.9	-25%	8.2	4.7	-42%
SCFXM2	2.6	1.9	-28%	7.8	5.4	-32%
SCTAP2	3.3	2.6	-21%	9.2	6.6	-28%
PILOT	13.0	8.0	-38%	22.9	12.7	-45%
BP1	14.8	12.6	-15%	32.5	26.9	-17%

Roughly there is a 30-50% saving in BTRANL, and a 20-40% saving in FTRANL.

There is a small but noticeable tendency of the staircase version to run slower in BTRANU and FTRANU. Most likely this behavior is a consequence of the LU factorization: the staircase bump-and-spike pivot order tends to yield a denser U.

Some of the difference in BTRAN and FTRAN timings should be due to the methods of Section 4. The efficacy of these methods cannot be told from the above data, however, since the same timings are sensitive to differences in L and U density. Consequently a separate set of runs was made, employing the staircase LU factorization but not the Section 4 enhancements. The differences were as follows:

	TIME SAVED BY EFFICIENCIES IN FTRAN, BTRAN (SECTION 4)	% OF TOTAL TIME
SCAGR25	4.9	15%
SCRS8	4.1	12%
SCSD8	5.2	12%
SCFXM2	4.4	9%
SCTAP2	4.4	6%
PILOT	13.4	11%
BP1	3.5	2%

Thus the efficiencies in FTRAN and BTRAN cut total running times 9-15% in most cases; the savings would be more pronounced as a percentage of iterating time only. Predictably, LPs of many periods tended to show the greatest differences.

Comparable savings should be realized if staircase bump-and-spike pivot selection is replaced by staircase local minimization, since the methods of Section 4 apply equally well to either. Hence local minimization may well be superior for LPs such as SCAGR25 and SCFXM2 whose staircase factorizations--as reported in [15]--are notably denser under bump-and-spike.

The one sour note in the three tables above is BP1, on which the staircase iterating routines seem to perform rather poorly. On closer examination, however, this is not entirely surprising, as BP1 differs significantly from the other LPs. Whereas the others are first-order staircases (or, in the case of PILOT, very nearly first-order), BP1 has a large number of nonzeroes below the staircase; its form is in fact closer to dual-angular. BP1's bases consequently tend to be unbalanced. Hence the staircase technique produces considerably more spikes, and a much denser U factor. The result: much more time spent in FTRANU and BTRANU, offsetting any gains in FTRANL and BTRANL.

It thus appears that a good staircase form is essential to success of the staircase techniques. BP1's staircase arrangement was deduced from fairly scant information, and is evidently inadequate. A better staircase form may exist, but a better knowledge of the underlying model may be necessary to find it.

Factorizing routines

At intervals of typically 50-100 iterations a fresh factorization of the basis is computed by a separate set of routines. For bump-and-spike techniques, these "factorizing" routines fall into two classes: ones that select a pivot order, and ones that compute the L and U factors.

For the test problems, total time in factorizing routines—normalized to seconds per 10 refactorizations--was as follows:

	FACTORIZING TIME		
	STANDARD	STAIRCASE	% CHANGE
SCAGR25	1.4	1.6	+15%
SCRS8	1.1	1.4	+22%
SCSD8	2.7	1.6	-39%
SCFXM2	1.9	2.8	+47%
SCTAP2	1.7	3.0	+80%
PILOT	32.8	9.7	-70%
BP1	27.9	26.1	- 6%

The outcomes appear to vary wildly. However, they are the consequence of a few simple patterns which are revealed by looking at the pivot-selection routines and LU-computation routines separately, with reference to the third set of graphs in Appendix C.

Pivot selection involves a routine for the P3 heuristic, a block-triangularization routine (for the standard technique only), and main routines to call these and record the selected pivots. The staircase technique's main routine seems to run usually somewhat longer, probably because it is more complicated. The others' times are summarized below:

	STANDARD			STAIRCASE	MEDIAN SIZE,
	P3	BLK Δ	TOTAL	P3	LARGEST BUMP
SCAGR25	0.4	0.2	0.6	0.2	45
SCRS8	0.2	0.2	0.4	0.2	28
SCSD8	1.1	0.4	1.5	0.2	114
SCFXM2	0.2	0.5	0.7	0.8	36
SCTAP2	0.0	0.5	0.5	0.7	1
PILOT	20.4	1.0	21.4	2.4	533
BP1	13.1	2.0	15.1	3.8	408

The behavior of P3 is clearly critical. When bumps are small P3 is quite fast; but it begins to slow down whem bump size passes 100, and it is extremely inefficient on bumps of size 400 or 500. PILOT, the worst case here, spends 16% of its total running time in P3 alone! By extrapolation, it seems likely that P3 will be prohibitively slow for larger bumps. Thus a staircase bump-and-spike technique (or else an efficient local-minimization technique) may be essential for larger versions of models like SCSD8 and PILOT.

The main LU computation routines employ FTRANL and BTRANL as subroutines: FTRANL solves for the next column of L and U (as described in Section 2); BTRANL solves for row k of $B^{(k)}$ when a column interchange ("spike swap") is necessitated by an unacceptable pivot element. The test problems gave the following results (where SWAPS is the maximum number of swapped spikes per factorization):

	STANDARD LU				STAIRCASE LU			
	MAIN	FTRAN	BTRAN	SWAPS	MAIN	FTRAN	BTRAN	SWAPS
SCAGR25	0.2	0.0	0.0	3	0.6	0.1	0.1	20
SCRS8	0.2	0.0	0.0	1	0.4	0.1	0.1	11
SCSD8	0.4	0.1	0.0	6	0.5	0.1	0.1	11
SCFXM2	0.5	0.1	0.0	2	1.0	0.2	0.1	8
SCTAP2	0.2	0.0	0.0	0	0.7	0.1	0.4	19
PILOT	3.3	3.8	2.4	27	3.2	1.8	0.8	16
BP1	3.7	3.8	2.4	28	6.4	5.8	7.2	49

Predictably, the times are sensitive to the numbers of spike swaps; each swap requires another BTRANL and FTRANL, plus extra work in the main routine. Experience with PILOT and other LPs [15] suggests that the staircase pivot order may generally require fewer swaps when the bumps are big (as for PILOT) and the staircase is well-balanced (unlike BP1's). The other test LPs have smaller bumps and require fewer swaps with the standard pivot order.

Again the data suggest that staircase local-minimization techniques might be preferable for the small-bump staircase LPs. An efficient implementation of local minimization [12,45] incurs only a small extra cost in rejecting any unaccepatably small pivot element.

Comparison with a commercial code

The PILOT model was frequently solved--on the same computer as used for the above tests--by a commercially-marketed machine-language LP code, MPS III [37]. These runs employed the WHIZARD simplex routine of MPS III, which incorporates a bump-and-spike factorization scheme. Various system parameters were set from experience to yield fast PILOT runs.

For comparison, WHIZARD was run 1000 iterations from the same starting basis as used above with MINOS. The running times were as follows:

MINOS, standard pivot selection	155.7 sec
MPS III/WHIZARD	114.7 sec
MINOS, staircase pivot selection	106.4 sec.

MINOS did require considerably more storage, primarily because its storage scheme for the U factor could not efficiently accommodate a large number of spikes. U could probably be stored more compactly, however, without significant effect upon the MINOS timings.

Nothing very definite can be inferred from these figures, since MINOS and MPS III differ in many ways; moreover, the internal structure of the latter is largely unknown, as is the case with many commercial codes. Nevertheless, it is gratifying that MINOS—which is written in FORTRAN and intended more as a test code--can compete with a supposedly fast LP system. At the least, one may conclude that the timings throughout this section are probably quite realistic. And the superiority of staircase MINOS to MPS III for PILOT suggests that, for at least some large staircase problems, the techniques of this paper will offer significant savings.

APPENDIX A: TEST PROBLEMS

The linear programs used in the computational experiments of Section 5 are described in greater detail below. The tabular summarizes for each LP are largely self-explanatory, but a few general notes are appropriate:

All statistics except OBJ ELEMS refer only to the staircase constraint matrix, excluding the objective row and right-hand side. In each case the constraint matrix, A, has been put in reduced standard form; DIAGONAL BLOCKS refers to the staircase blocks $A_{\ell\ell}$, OFF-DIAGONAL BLOCKS to the blocks $\hat{A}_{\ell+1,\ell}$, and SUB-STAIR BLOCKS (when present) to the blocks $A_{\ell+2,\ell}, \dots, A_{t\ell}$.

Variables (columns) are implicitly constrained only to be non-negative, unless there is an indication to the contrary. BOUNDED implies implicit lower and upper bounds, FIXED implies fixture at a given value, and FREE implies no implicit constraints.

MAX ELEM and MIN ELEM are the largest and smallest magnitudes of elements in A; LARGEST COL RATIO is the greatest ratio of magnitudes of elements in the same column of A. Where values are given BEFORE SCALING and AFTER SCALING, all tests were conducted with A scaled as described in Appendix B. Otherwise NO SCALING is indicated.

SCAGR25

Test problem received from James K. Ho, Brookhaven National Laboratory, Upton, N.Y.; source not documented.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1	18	20	45	13%	8	7	17	30%	19
2-24	19	20	46	12%	8	7	17	30%	19
25	16	20	43	13%					19
			1146	12%			408	30%	475

GRAND TOTALS

ROWS	471	(300 EQUALITIES, 171 INEQUALITIES)
COLS	500	
ELEMS	1554	
DENS	0.7%	

COEFFICIENTS	NO SCALING
MAX ELEM	1.3
MIN ELEM	2.0×10^{-1}
LARGEST COL RATIO	1.9×10^{-1}

SCRS8

Derived from a model of the United States' options for a transition from oil and gas to synthetic fuels; documented in [27,33].

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1	28	37	65	6%	25	22	29	5%	18
2	28	38	69	6%	25	22	29	5%	19
3-5	31	76	181	8%	25	22	29	5%	55
6-8	32	79	192	8%	25	22	29	5%	58
9	31	79	189	8%	25	22	29	5%	58
10-12	31	80	190	8%	25	22	29	5%	59
13-15	30	80	186	8%	25	22	29	5%	59
16	31	70	177	8%					59
			2747	8%			435	5%	847

GRAND TOTALS

ROWS	490	(384 EQUALITIES, 106 INEQUALITIES)
COLS	1169	
ELEMS	3182	
DENS	0.6%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	3.9×10^2	4.0
MIN ELEM	1.0×10^{-3}	2.5×10^{-1}
LARGEST COL RATIO	4.5×10^3	1.6×10^1

SCSD8

A multi-stage structural design problem, documented in [26]. This is the only staircase test problem for this paper in which the stages do not represent periods of time.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1-38	10	70	130	19%	10	50	90	18%	70
39	17	90	224	15%					90
			5164	18%			3420	18%	2750

GRAND TOTALS

ROWS	397	(ALL EQUALITIES)
COLS	2750	
ELEMS	8584	
DENS	0.8%	

COEFFICIENTS	NO SCALING
MAX ELEM	1.0
MIN ELEM	2.4×10^{-1}
LARGEST COL RATIO	4.0

SCFXM2

Test problem received from James K. Ho, Brookhaven National Laboratory, Upton, New York; source not documented.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1	92	114	679	6%	9	57	61	12%	13
2	82	99	434	5%	9	35	35	11%	4
3	66	126	300	4%	5	33	33	20%	1
4	90	118	1047	10%	5	5	5	20%	5
5	92	114	679	6%	9	57	61	12%	13
6	82	99	434	5%	9	35	35	11%	4
7	66	126	300	4%	5	33	33	20%	1
8	90	118	1047	10%					5
			4920	7%			263	13%	46

GRAND TOTALS

ROWS	660	(374 EQUALITIES, 286 INEQUALITIES)
COLS	914	
ELEMS	5183	
DENS	0.9%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	1.3×10^{2}	1.1×10^{1}
MIN ELEM	5.0×10^{-4}	8.7×10^{-2}
LARGEST COL RATIO	1.3×10^{5}	1.3×10^{2}

SCTAP2

A dynamic traffic assignment problem, documented in [28]. The LP has 11 objective rows; the objective named OBJZZZZZ was used in all tests. Statistics below omit the other ten objectives.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS
1-9	109	188	423	2%	62	138	276	3%	141
10	109	188	423	2%					141
			4230	2%			2484	3%	1410

GRAND TOTALS

ROWS	1090	(470 EQUALITIES, 620 INEQUALITIES)
COLS	1880	
ELEMS	6714	
DENS	0.3%	

COEFFICIENTS	NO SCALING
MAX ELEM	8.0×10^1
MIN ELEM	1.0
LARGEST COL RATIO	8.0×10^1

PILOT

Derived from a welfare equilibrium model of the United States' energy supply, energy demand, and economic growth: seeks maximum aggregate consumer welfare subject to competitive market equilibrium. The LP was supplied by the PILOT modeling project, Systems Optimization Laboratory, Department of Operations Research, Stanford University; it is documented in [40].

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				SUB-STAIR BLOCKS		OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS	DENS	ELEMS
1	84	343	686	2%	31	74	105	5%	18	0%	10
2	90	345	1079	3%	34	76	111	4%	8	0%	10
3	90	343	1073	3%	34	74	109	4%	5	0%	10
4	90	343	1073	3%	34	74	109	4%	5	0%	10
5	90	343	1073	3%	34	74	109	4%	5	0%	10
6	90	343	1073	3%	34	74	109	4%	3	0%	10
7	90	343	1073	3%	32	74	107	5%	1	0%	10
8	87	341	1060	4%	4	19	19	25%			10
9	11	45	113	23%							12
			8303	3%			778	4%	45	0%	92

GRAND TOTALS

ROWS	722	(583 EQUALITIES, 139 INEQUALITIES)
COLS	2789	(80 FREE, 296 BOUNDED, 79 FIXED)
ELEMS	9126	
DENS	0.5%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	4.8×10^{4}	2.0×10^{1}
MIN ELEM	1.4×10^{-4}	4.9×10^{-2}
LARGEST COL RATIO	7.0×10^{6}	4.2×10^{2}

BP1

Developed by British Petroleum, London; supplied via the Systems Optimization Laboratory, Department of Operations Research, Stanford University.

This LP is approximately dual-angular, with 6 main diagonal blocks and about 400 coupling variables. For the experiments described in this paper it was treated as a 6-period, 5th-order staircase problem.

	DIAGONAL BLOCKS				OFF-DIAGONAL BLOCKS				SUB-STAIR BLOCKS		OBJ
PERIOD	ROWS	COLS	ELEMS	DENS	ROWS	COLS	ELEMS	DENS	ELEMS	DENS	ELEMS
1	111	227	1400	6%	3	60	3	2%	163	0%	138
2	151	353	2175	4%	62	108	112	2%	142	0%	149
3	113	321	964	3%	92	232	346	2%	494	1%	270
4	170	295	2178	4%	51	14	11	2%	4	0%	74
5	134	198	1315	5%	111	2	2	1%			40
6	142	177	1091	4%							56
			9123	4%			474	2%	803	0%	727

GRAND TOTALS

ROWS	821	(516 EQUALITIES, 305 INEQUALITIES)
COLS	1571	
ELEMS	10400	
DENS	0.8%	

COEFFICIENTS	BEFORE SCALING	AFTER SCALING
MAX ELEM	2.4×10^2	1.3×10^1
MIN ELEM	2.0×10^{-4}	7.6×10^{-2}
LARGEST COL RATIO	1.7×10^5	1.7×10^2

APPENDIX B: DETAILS OF COMPUTATIONAL TESTS

Computing environment

All computational experiments were performed on the Triplex system [49] at the Stanford Linear Accelerator Center, Stanford University. The Triplex comprises three computers linked together: one IBM 360/91, and two IBM 370/168s. Runs were submitted as batch jobs in a virtual-machine environment, under the control of IBM systems OS/VS2, OS/MVT and ASP.

Test runs employed a specially-modified set of linear-programming routines from the MINOS system [38,48]. MINOS is written in standard FORTRAN. For timed runs, MINOS was compiled with the IBM FORTRAN IV (H extended, enhanced) compiler, version 1.1.0, at optimization level 3 [30].

Timings

All running-time statistics are based on "CPU second" totals for individual job steps as reported by the operating system. To promote consistency all timed jobs were run on the Triplex computer designated "system A," and jobs whose timings would be compared were run at about the same time. Informal experiments indicated roughly a 1% variation in timings due to varying system loads.

More detailed timings employed PROGLOOK [31], which takes frequent samples of a running program to estimate the proportion of time spent in each subroutine. To determine the actual time in seconds for each subroutine, every timed job was run twice--once without PROGLOOK to measure total CPU seconds, and once with PROGLOOK to estimate each subroutine's proportion of the total. PROGLOOK estimates were based on at least 2300 samples per job.

MINOS linear-programming environment

MINOS was set up for test runs according to the defaults indicated in [38], with the exception of the items listed below.

Scaling. Problems noted as "scaled"in Appendix A were subjected to the following geometric-mean scaling (where A denotes the matrix of constraint coefficients, not including the objective or right-hand side):

1: Compute $\rho_0 = \max|A_{i_1j}/A_{i_2j}|$, $A_{i_2j} \neq 0$.

2: Divide each row i of A, and its corresponding right-hand side value, by $[(\min_j|A_{ij}|)(\max_j|A_{ij}|)]^{1/2}$, taking the minimum over all $A_{ij} \neq 0$.

3: Divide each column j of A, and its corresponding coefficient in the objective, by $[(\min_i|A_{ij}|)(\max_i|A_{ij}|)]^{1/2}$, taking the minimum over all $A_{ij} \neq 0$.

4: Compute $\rho = \max|A_{i_1j}/A_{i_2j}|$, $A_{i_2j} \neq 0$.

This procedure was repeated as many times as possible until, at step 4, ρ was at least 90% of ρ_0. (In other words, scaling continued as long as it reduced ρ, the greatest ratio of two elements in the same column, by more than 10%.)

Starting basis. All LPs except PILOT and BP1 were solved with crash option 0 of MINOS: the initial basis was composed entirely of unit vectors, and all nonbasic variables were placed at zero. PILOT and BP1 were run from initial bases that had been reached and saved in previous MINOS runs.

Termination. All LPs except PILOT and BP1 were run until an optimal solution was found. PILOT and BP1 were run for 1000 and 750 iterations, respectively.

Pricing. Except for SCTAP2, the partial-pricing scheme of MINOS was employed--with one important change: the arbitrary partitioning of the columns normally defined by MINOS for partial pricing was replaced by the natural staircase partition. Thus the periods of the staircase were priced one at a time in a cyclic fashion.

Pricing for SCTAP2 was similar except that the incoming column was chosen from the latest possible period. (This choice was known to produce a relatively small number of iterations from an all-unit-vector start.)

Refactorization frequency. MINOS was instructed to refactorize the basis (by performing a fresh Gaussian elimination) every 50 iterations, except for BP1 (every 75) and PILOT (every 90).

Tolerances. The "LU ROW TOL" for MINOS was set to 10^{-4}. All other tolerances were left at their default values.

Modifications to MINOS

All runs described in this paper were made with a special test version of MINOS. This version retained MINOS' routines for standard bump-and-spike elimination, and added new routines to implement a version of staircase bump-and-spike elimination. Routines for solving linear systems were also modified to take advantage of the staircase pivot order. Control routines were adjusted appropriately.

New subroutines in the test version are described briefly as follows:

SP3--an adaptation of the P3 heuristic to find a bump-and-spike structure in non-square or rank-deficient blocks, as proposed in [15]. This routine is a modification of the MINOS subroutine P3.

SP4--main routine for the staircase bump-and-spike pivot-selection technique of [15]; sorts the staircase basis into reduced form, and calls SP3 once for each diagonal block.

DSPSPK--spike-display routine; prints a graphical summary of the basis bump-and-spike structure found by P4 (for the standard technique) or SP4 (for the staircase technique).

STAIR--a staircase analyzer. Given an initial partition of the rows by period, this routine permutes the constraint matrix to a reduced standard staircase form and stores the staircase partitions in arrays that are read by subsequent routines. STAIR is called once at the beginning of every run.

SCALE--implementation of the geometric-mean scaling scheme described above; called optionally at the beginning of a run.

UPDBAL--updating routine for cumulative-balance counts: after each iteration, revises an array that records the cumulative excess of columns over rows at each period of the staircase basis. (This array is used to find square sub-staircases.)

In addition the test version incorporates the following substantial modifications to MINOS subroutines:

FACTOR efficiently handles a pivot order from either the standard or staircase technique, and finds the partitions λ_ℓ and μ_ℓ (defined in Section 4) for the staircase technique.

FTRANL, BTRANL, FTRANU and BTRANU incorporate the ideas of Section 4 in a uniform way. FTRANL and FTRANU can begin at a specified L or U transformation, and BTRANL and BTRANU can stop at a specified transformation. BTRANL can also be restarted at a point where it previously stopped.

LPITN determines a starting point for FTRANL and a stopping point for BTRANU when the staircase technique is used.

SETPI, for the staircase technique, determines a starting point for FTRANU and a stopping point for BTRANL when it is first called at an iteration. When subsequently called at the same iteration it determines restarting and stopping points for BTRANL.

PRICE incorporates the staircase-oriented partial-pricing methods described in the preceding subsection of this appendix. When these methods are used with the staircase factorization technique, PRICE also keeps track of how much of the price vector it requires, and calls SETPI accordingly.

SPECS2 determines whether the standard or staircase technique will be used in a particular run, according to instructions in the SPECS input file.

Other subroutines were modified as necessary to accommodate these changes.

MPS III linear programming environment

For purposes of comparison the PILOT test problem was also run on the MPS III system [37], as explained in Section 5.

The MPS III run employed the WHIZARD linear-programming routines of version 8915 of MPS III. The run used the same starting basis as the MINOS runs for PILOT, and was terminated after 1000 iterations like the MINOS runs. Exact CPU timings were 0.56 seconds in the compiler step and 114.18 seconds in the executor step.

The control program for the MPS III run was as follows:

```
          PROGRAM
          INITIALZ
          XPROC = XPROC + 6000
          XCLOCKSW = 0
          XINVERT = 1
          XFREQINV = 90
          XFREQLGO = 1
          XFREQ1 = 1000
          MVADR (XDOFREQ1, TIME)
          MOVE (XDATA, 'PILOT.WE')
          CONVERT ('FILE','INPUT')
          SETUP ('BOUND', 'BOUND', 'MAX', 'SCALE')
          MOVE (XOBJ, 'OBJ')
          MOVE (XRHS,'RHSIDE')
          INSERT ('FILE','PUNCH1')
WHIZFREQ  DC (250)
WHIZSCAL  DC (4)
          WHIZARD('FREQ', WHIZFREQ, 'SCALE', WHIZSCAL)
TIME      PUNCH ('FILE', 'PUNCH1')
          EXIT
          PEND
```

APPENDIX C: TIMINGS

The bar graphs below summarize timings of the MINOS test runs for this paper. Details of the test runs and timing procedures are in Appendix B; individual MINOS subroutines are documented in Appendix B and in [48].

Graphs are presented in three groups. The first group shows time in all routines, the second shows time in iterating routines only, and the third shows time in factorizing routines only. Within each group the format is the same: the first graph compares totals for all seven test problems, and seven succeeding graphs--one for each test problem--break the times down into various subtotals.

All graphs show a pair of bars for each total or subtotal. The top bar is for the run that used standard bump-and-spike elimination on the basis; the bottom bar is for the run that used staircase bump-and-spike elimination and the related techniques described in this paper.

Total time

The FORTRAN subroutines of MINOS are classified below as follows:

PRICE routines choose a nonbasic variable to enter the basis; they include FORMC, PRICE, SETPI and FTRANU, and BTRANL when called from SETPI.

PIVOT routines choose a variable to leave the basis; they include LPITN and CHUZR, and FTRANL, BTRANU and UNPACK when called from LPITN.

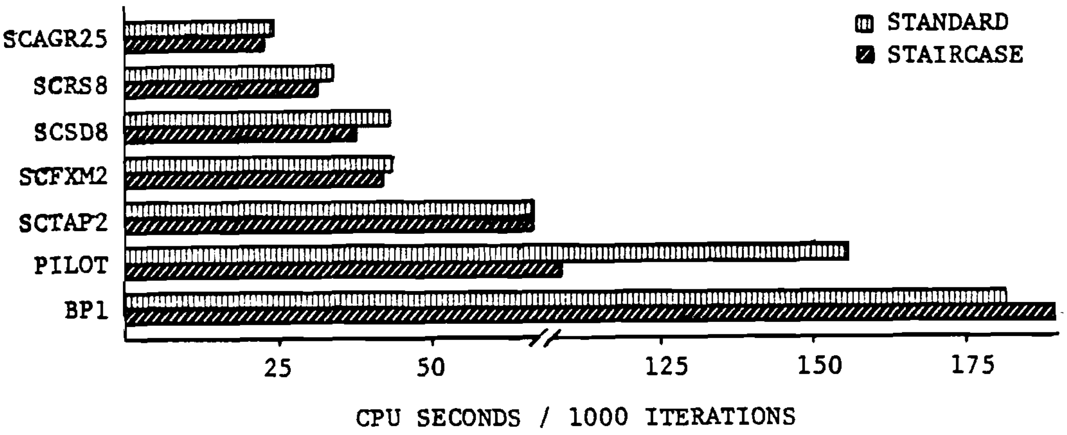
UPDATE refers to the subroutine MODLU, which updates the LU factorization of the basis at the end of each iteration.

PERM routines permute the basis of a bump-and-spike structure. For the standard method they include P4, P3, TRANSVL, BUMPS and MKLIST; for the staircase method they are SP4, SP3 and MKLIST.

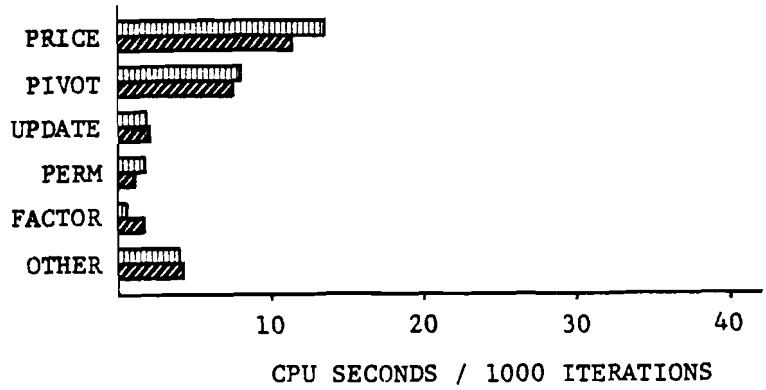
FACTOR routines compute an LU factorization of the basis; they include FACTOR and PACKLU, and FTRANL, BTRANL and UNPACK when called from FACTOR.

OTHER routines include all other MINOS subroutines, and utility routines inserted by the FORTRAN compiler. Other MINOS routines comprise DRIVER and routines it uses (BTRANU, FTRANL, ITEROP, SETX, STATE, UNPACK, UPDBAL), INVERT and routines it uses (BTRANU, DSPSPK, FTRANL, SETX), and various routines called once only at the beginning or end of the run (CRASH, GO, HASH, INITLZ, LOADB, MINOS, MOVE, MPS, MPSIN, NMSRCH, SAVEB, SCALE, SOLN, SOLPRT, SPECS, SPECS2, STAIRS). FORTRAN routines for input and output registered significantly (3-10% of total) in the timings; the volume of input was very small, so these routines probably did most of their work in producing printed output for the runs. A FORTRAN square-root subroutine, called from SCALE and SETPI, used an insignificant amount of time.

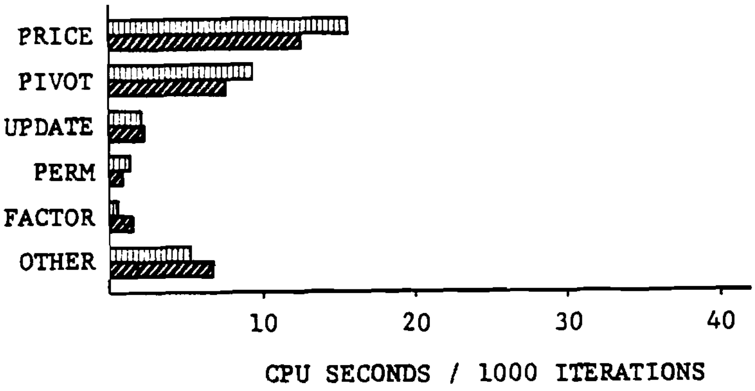
TOTAL TIME



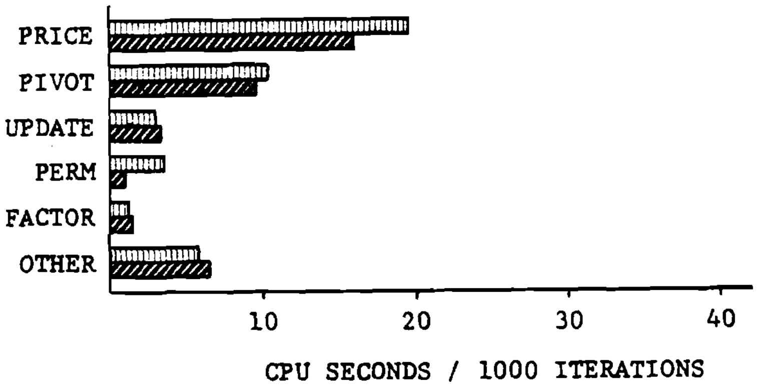
SCAGR25



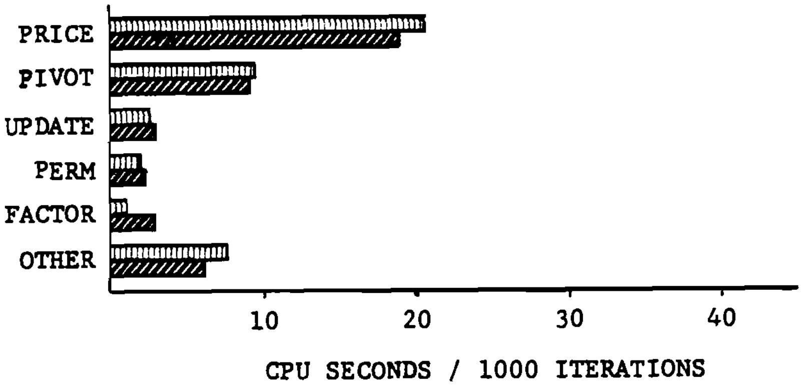
SCRS8


PRICE
PIVOT
UPDATE
PERM
FACTOR
OTHER
10
20
30
40
CPU SECONDS / 1000 ITERATIONS

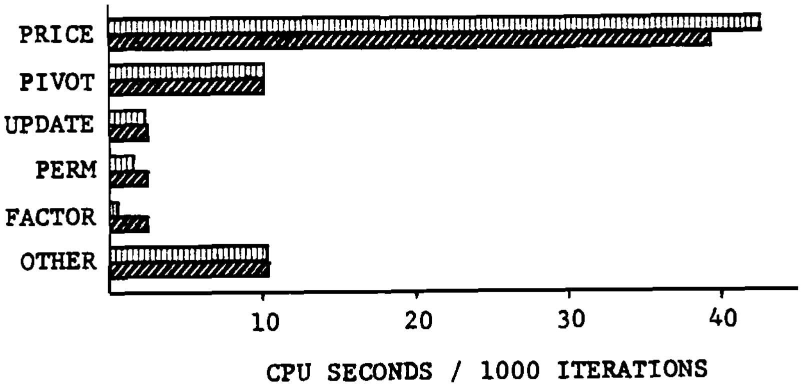
SCSD8

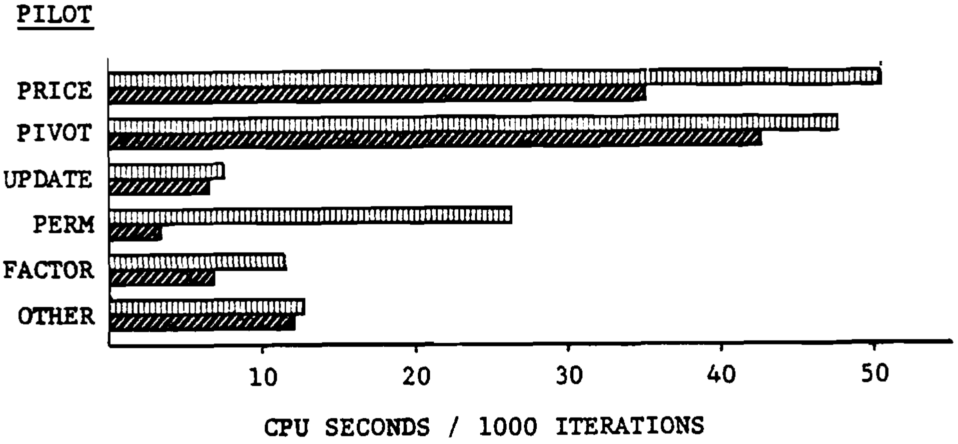

PRICE
PIVOT
UPDATE
PERM
FACTOR
OTHER
10
20
30
40
CPU SECONDS / 1000 ITERATIONS

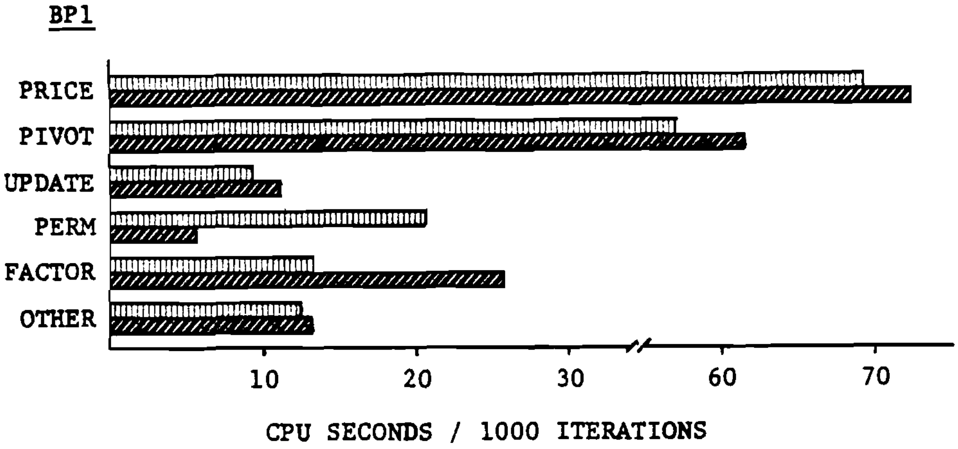
SCFXM2



SCTAP2




PILOT
PRICE
PIVOT
UPDATE
PERM
FACTOR
OTHER
10
20
30
40
50
CPU SECONDS / 1000 ITERATIONS


BP1
PRICE
PIVOT
UPDATE
PERM
FACTOR
OTHER
10
20
30
60
70
CPU SECONDS / 1000 ITERATIONS

Iterating time

Iterating routines are those invoked at each iteration. They are classified as follows:

MAIN includes DRIVER and miscellaneous routines invoked from it: ITEROP, SETX, STATE, UNPACK and UPDBAL, and FTRANL and BTRANU when called from SETX.

PRICE refers to subroutines FORMC, PRICE and SETPI.

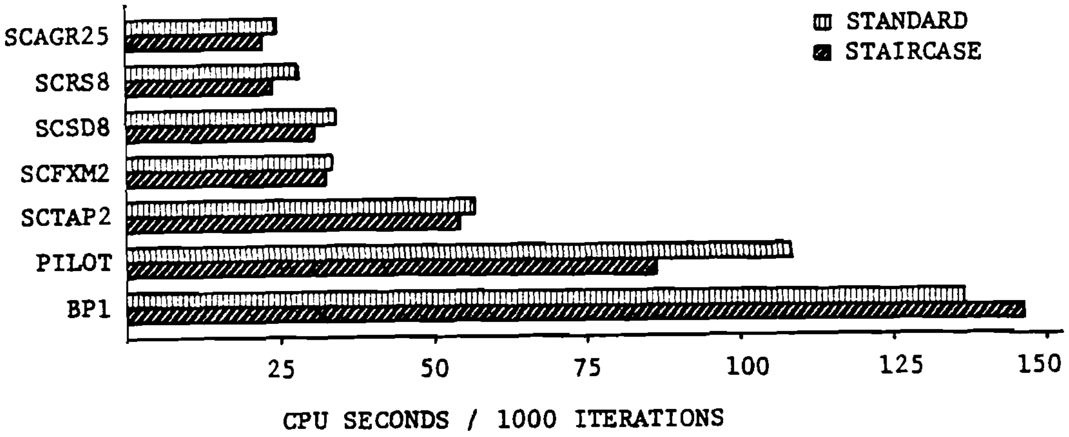
FTRANU and BTRANL refer to the like-named subroutines when called from SETPI.

PIVOT refers to subroutines LPITN and CHUZR, and UNPACK when called from LPITN.

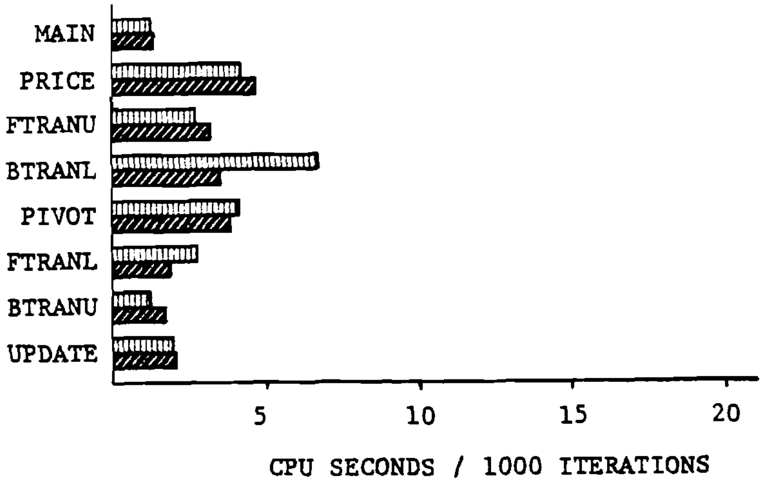
FTRANL and BTRANU refer to the like-named subroutines when called from LPITN.

UPDATE refers to subroutine MODLU.

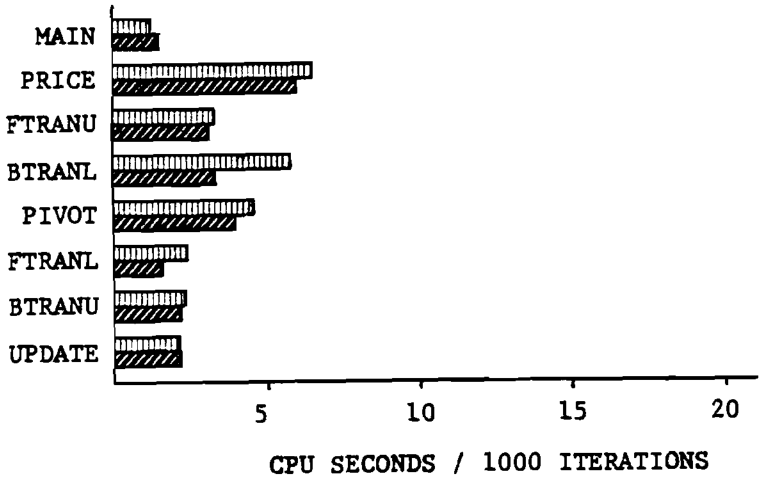
TOTAL ITERATING TIME



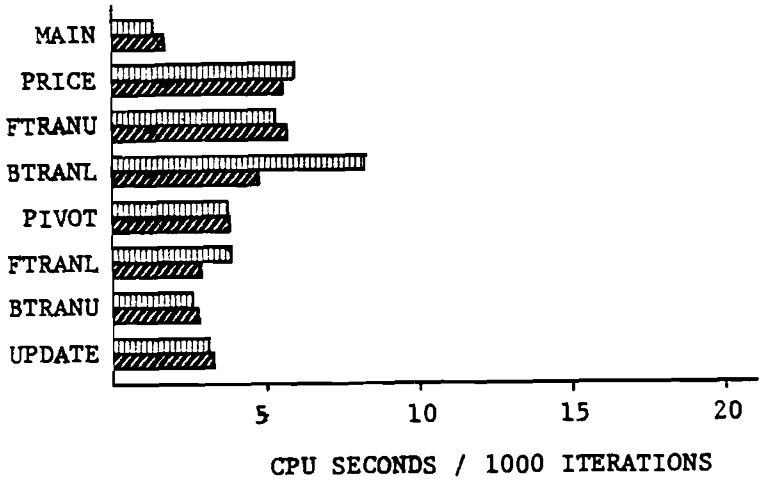
SCAGR25



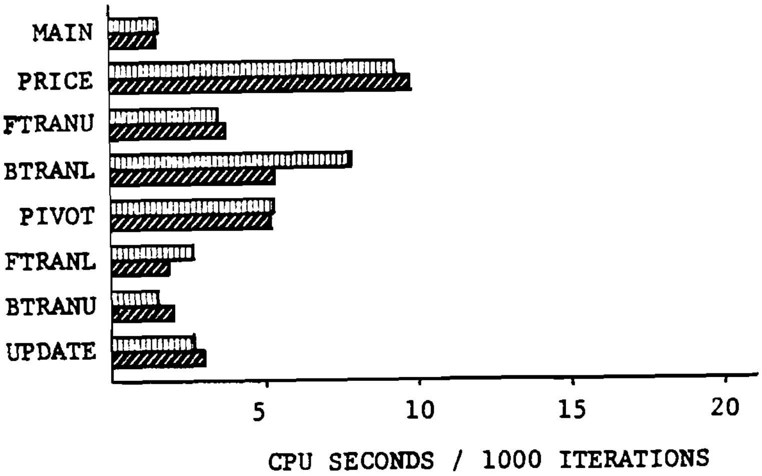
SCRS8


MAIN
PRICE
FTRANU
BTRANL
PIVOT
FTRANL
BTRANU
UPDATE
5
10
15
20
CPU SECONDS / 1000 ITERATIONS

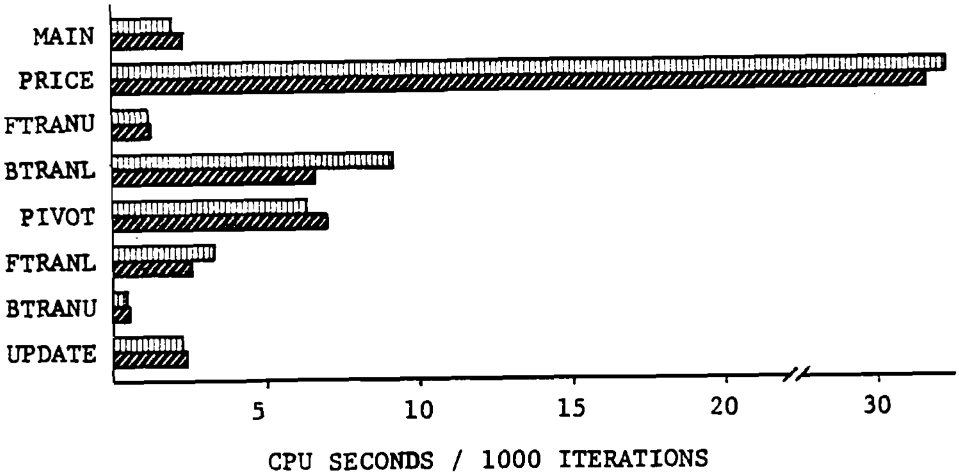
SCSD8

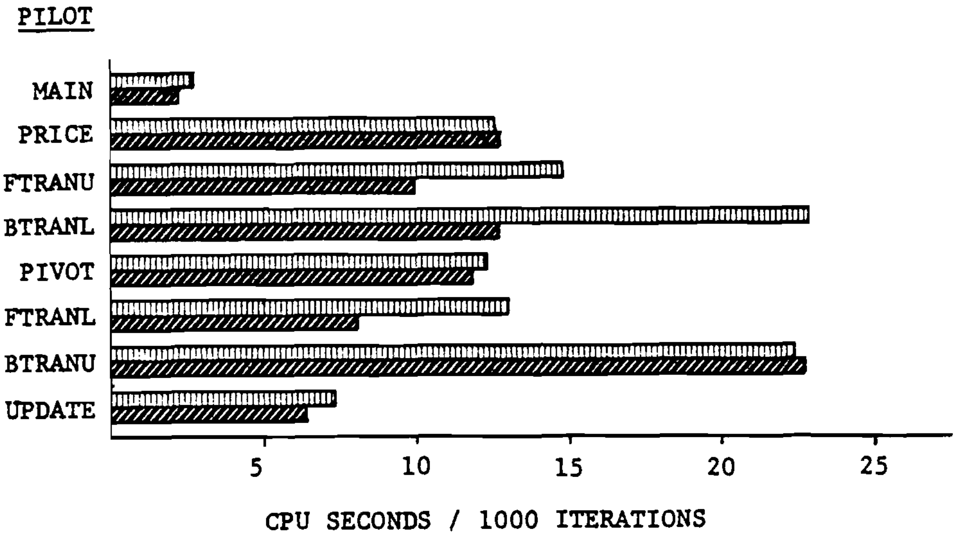

MAIN
PRICE
FTRANU
BTRANL
PIVOT
FTRANL
BTRANU
UPDATE
5
10
15
20
CPU SECONDS / 1000 ITERATIONS

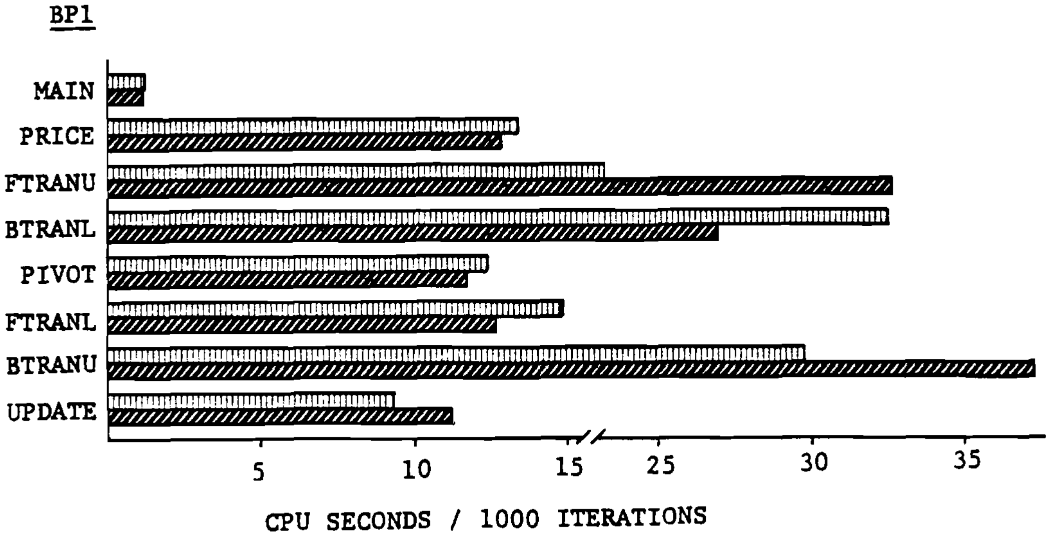
SCFXM2



SCTAP2




PILOT
MAIN
PRICE
FTRANU
BTRANL
PIVOT
FTRANL
BTRANU
UPDATE
5
10
15
20
25
CPU SECONDS / 1000 ITERATIONS


BP1
MAIN
PRICE
FTRANU
BTRANL
PIVOT
FTRANL
BTRANU
UPDATE
5
10
15
25
30
35
CPU SECONDS / 1000 ITERATIONS

Factorizing time

Factorizing routines are those invoked at each refactorization of the basis. They are classified as follows:

MAIN includes INVERT and miscellaneous routines invoked from it: DSPSPK and SETX, and FTRANL and BTRANU when called from SETX.

PERMUTE includes the driving routine for bump-and-spike permutation--P4 with the standard method, SP4 with the staircase method--and the utility routine MKLIST.

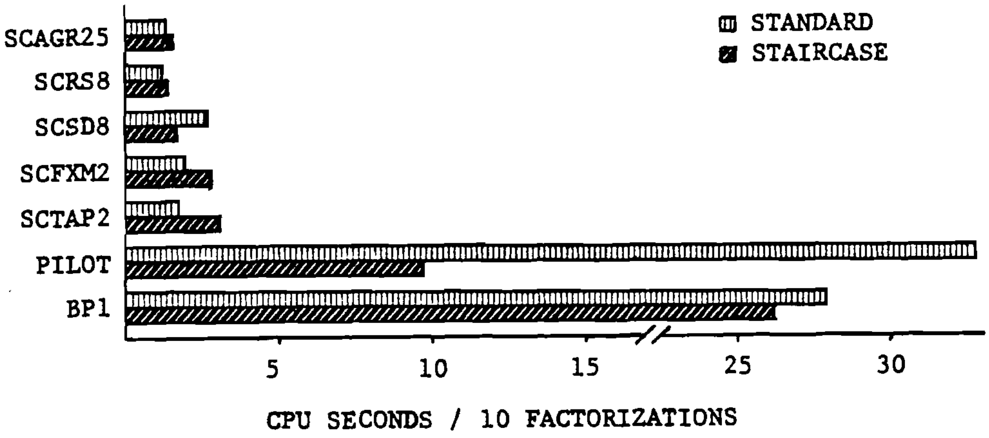
P3 refers to the subroutine that implements the spike-finding heuristic: P3 for the standard method, or SP3 for the staircase method.

BLK Δ refers to subroutines TRNSVL and BUMPS, which find a block-triangular reduction of the basis (in the standard method only).

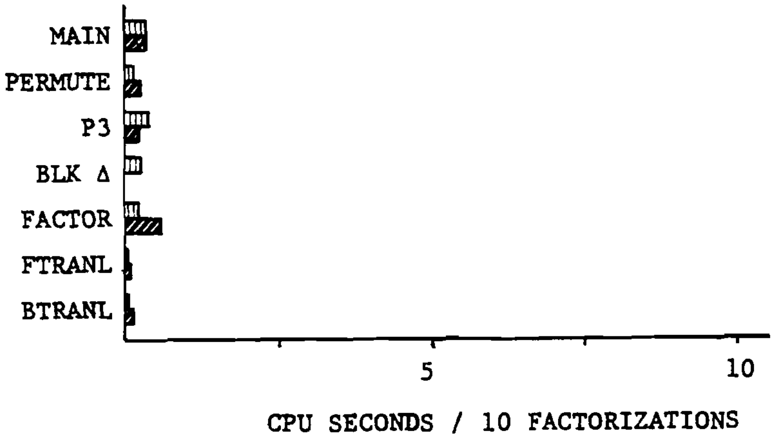
FACTOR includes subroutine FACTOR, the driving routine for LU factorization of the basis, plus routines PACKLU and UNPACK invoked from FACTOR.

FTRANL and BTRANL refer to the like-named subroutines when called from FACTOR.

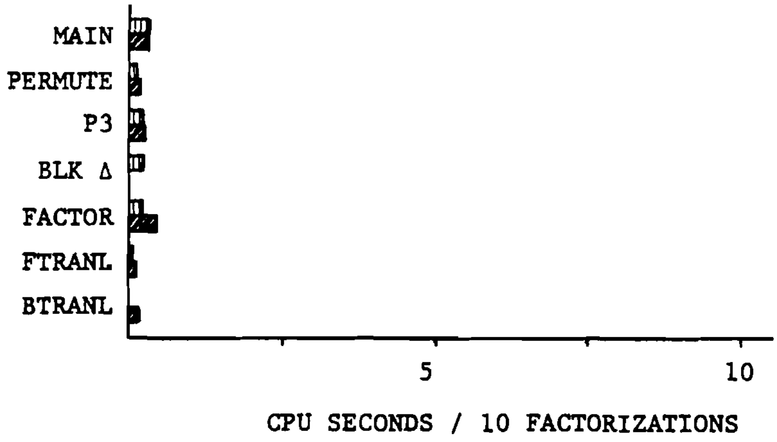
TOTAL FACTORIZING TIME



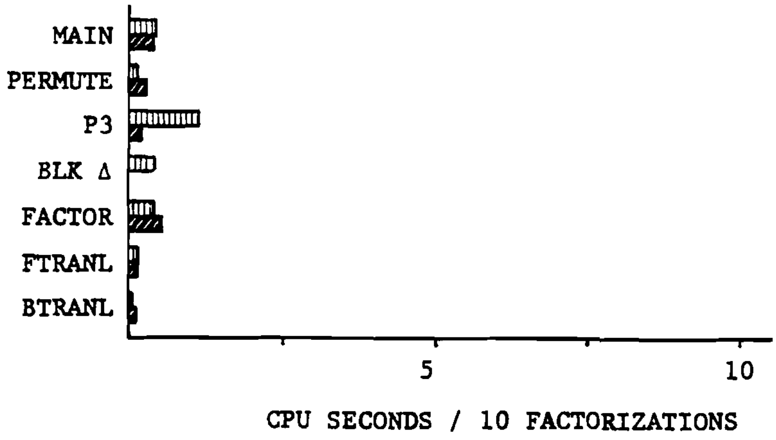
SCAGR25



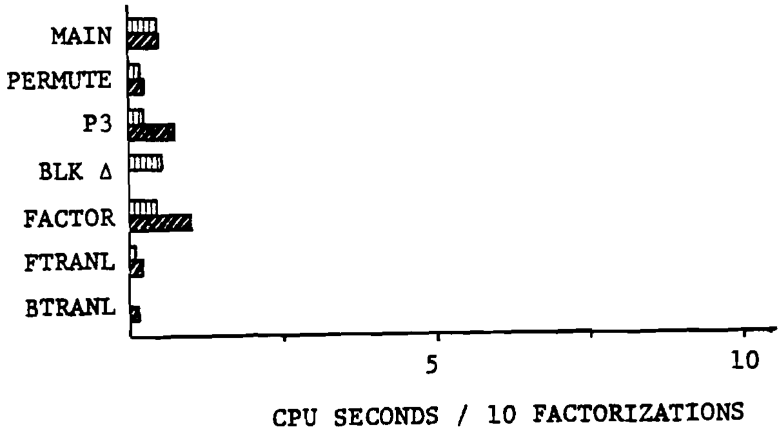
SCRS8


MAIN
PERMUTE
P3
BLK Δ
FACTOR
FTRANL
BTRANL
5
10
CPU SECONDS / 10 FACTORIZATIONS

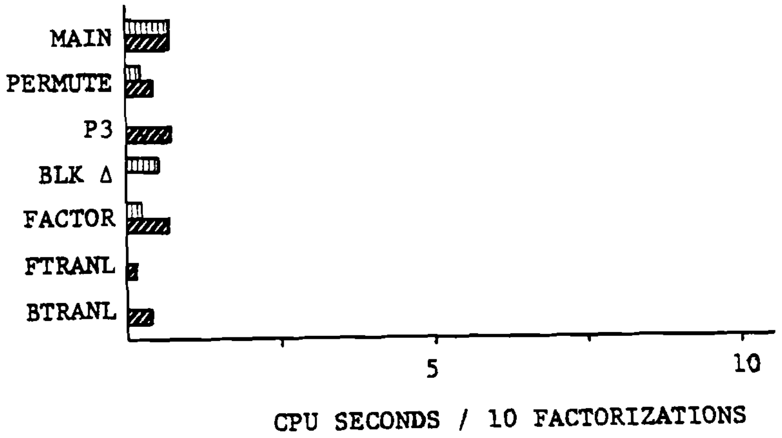
SCSD8


MAIN
PERMUTE
P3
BLK Δ
FACTOR
FTRANL
BTRANL
5
10
CPU SECONDS / 10 FACTORIZATIONS

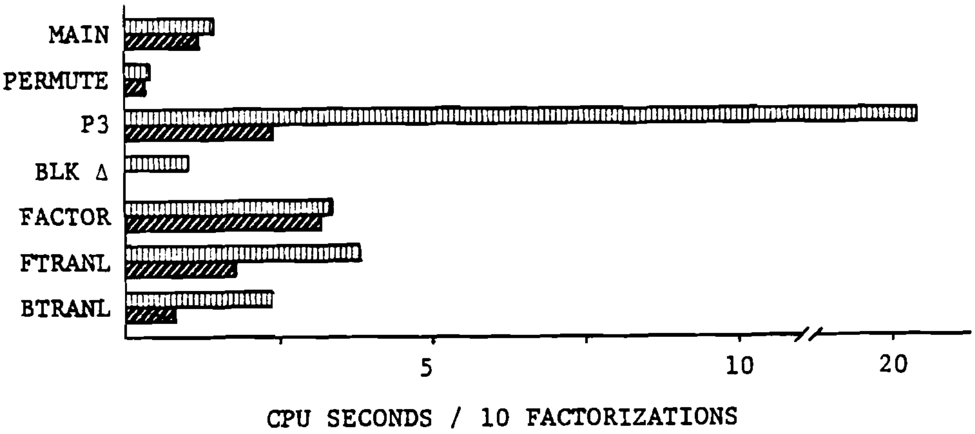
SCFXM2



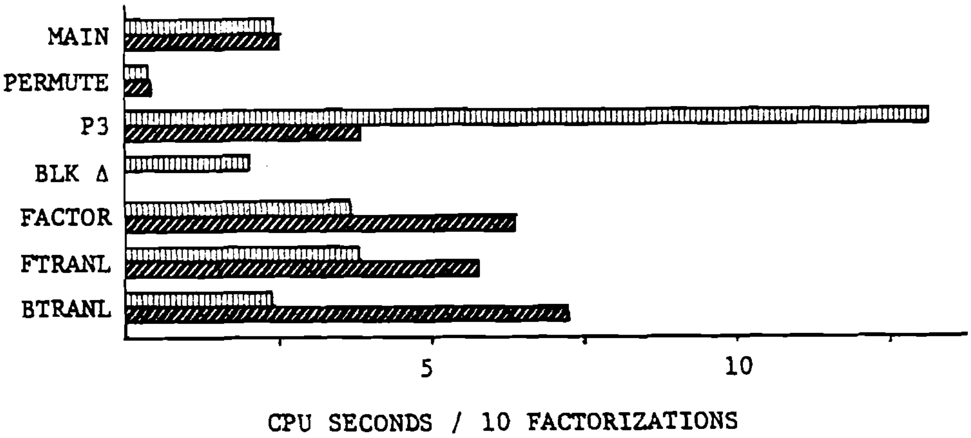
SCTAP2



PILOT


MAIN
PERMUTE
P3
BLK Δ
FACTOR
FTRANL
BTRANL
5
10
20
CPU SECONDS / 10 FACTORIZATIONS

BP1


MAIN
PERMUTE
P3
BLK Δ
FACTOR
FTRANL
BTRANL
5
10
CPU SECONDS / 10 FACTORIZATIONS

REFERENCES

[1] Aonuma, T., "A Two-Level Algorithm for Two-Stage Linear Programs." Journal of the Operations Research Society of Japan 21 (1978), 171-187.

[2] Bartels, Richard H., "A Stabilization of the Simplex Method." Numerische Mathematik 16 (1971), 414-434.

[3] _____ and Gene H. Golub, "The Simplex Method of Linear Programming Using LU Decomposition." Communications of the ACM 12 (1969), 266-268.

[4] Cobb, R. H. and J. Cord, "Decomposition Approaches for Solving Linked Problems." Proceedings of the Princeton Symposium on Mathematical Programming, Harold W. Kuhn, ed. (Princeton University Press, 1970).

[5] Dantzig, George B., "Programming of Interdependent Activities II: Mathematical Model." Econometrica 17 (1949), 200-211.

[6] _____, "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming." Econometrica 23 (1955), 174-183.

[7] _____, "Optimal Solution of a Dynamic Leontief Model with Substitution." Econometrica 23 (1955), 295-302.

[8] _____, "Compact Basis Triangularization for the Simplex Method." Recent Advances in Mathematical Programming, R. L. Graves and Philip Wolfe, eds. (New York: McGraw-Hill Book Co., 1963), 125-132.

[9] _____, "Solving Staircase Linear Programs by a Nested Block-Angular Method." Technical Report 73-1, Dept. of Operations Research, Stanford University (1973).

[10] _____ and Philip Wolfe, "Decomposition Principle for Linear Programs." Operations Research 8 (1960), 101-111.

[11] Duff, Iain S., "On the Number of Nonzeroes Added when Gaussian Elimination is Performed on Sparse Random Matrices." Mathematics of Computation 28 (1974), 219-230.

[12] _____, "Practical Comparisons of Codes for the Solution of Sparse Linear Systems." Sparse Matrix Proceedings--1978, Iain S. Duff and G. W. Stewart, eds. (Society for Industrial and Applied Mathematics, 1979).

[13] _____ and J. K. Reid, "A Comparison of Sparsity Orderings for Obtaining a Pivotal Sequence in Gaussian Elimination." Journal of the Institute of Mathematics and Its Applications 14 (1974), 281-291.

[14] Forrest, J. J. H. and J. A. Tomlin, "Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method." Mathematical Programming 2 (1972), 263-278.

[15] Fourer, Robert, "Sparse Gaussian Elimination of Staircase Systems." Technical Report SOL 79-17, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1979).

[16] _____, "Solving Staircase Linear Programs by the Simplex Method, 2: Pricing." Technical Report SOL 79-19, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1979).

[17]. Gay, David M., "On Combining the Schemes of Reid and Saunders for Sparse LP Bases." Sparse Matrix Proceedings-1978, Iain S. Duff and G. W. Stewart, eds. (Society for Industrial and Applied Mathematics, 1979).

[18] Gear, C. W. et al., "Numerical Computation: Its Nature and Research Directions." SIGNUM Newsletter, Association for Computing Machinery (1979).

[19] Glassey, C. Roger, "Dynamic Linear Programs for Production Scheduling." Operations Research 19 (1971), 45-56.

[20] _____, "Nested Decomposition and Multi-Stage Linear Programs." Management Science 20 (1973), 282-292.

[21] Goldfarb, D., "On the Bartels-Golub Decomposition for Linear Programming Bases." Mathematical Programming 13 (1977), 272-279.

[22] Grinold, Richard C., "Steepest Ascent for Large-Scale Linear Programs." SIAM Review 14 (1972), 447-464.

[23] Heesterman, A. R. G. and J. Sandee, "Special Simplex Algorithm for Linked Problems." Management Science 11 (1965), 420-428.

[24] Hellerman, Eli and Dennis Rarick, "Reinversion with the Preassigned Pivot Procedure." Mathematical Programming 1 (1971), 195-216.

[25] _____, "The Partitioned Preassigned Pivot Procedure (P^4)." Sparse Matrices and Their Applications, Donald J. Rose and Ralph A. Willoughby, eds. (New York: Plenum Press, 1972), 67-76.

[26] Ho, James K., "Optimal Design of Multi-Stage Structures: A Nested Decomposition Approach." Computers and Structures 5 (1975), 249-255.

[27] _____, "Nested Decomposition of a Dynamic Energy Model." Management Science 23 (1977), 1022-1026.

[28] _____, "A Successive Linear Optimization Approach to the Dynamic Traffic Assignment Problem." Report BNL-24713, Brookhaven National Laboratory, Upton, New York (1978).

[29] _____ and Alan S. Manne, "Nested Decomposition for Dynamic Models." Mathematical Programming 6 (1974), 121-140.

[30] IBM OS FORTRAN IV (H Extended) Compiler Programmer's Guide. No.SC28-6852, International Business Machines Corp. (1974).

[31] Johnson, R. and T. Johnston, "PROGLOOK User's Guide." User Note 33, SLAC Computing Services, Stanford Linear Accelerator Center (1976).

[32] Madsen, Oli B. G., "Solution of LP-Problems with Staircase Structure." Research Report 26, The Institute of Mathematical Statistics, Lyngby, Denmark (1977).

[33] Manne, A. S., "U.S. Options for a Transition from Oil and Gas to Synthetic Fuels." Discussion Paper 26D, Public Policy Program, Kennedy School of Government, Harvard University (1975).

[34] Markowitz, Harry M., "The Elimination Form of the Inverse and Its Application to Linear Programming." Management Science 3 (1957), 255-269.

[35] Marsten, Roy E. and Fred Shepardson, "A Double Basis Simplex Method for Linear Programs with Complicating Variables." Technical Report 531, Dept. of Management Information Systems, University of Arizona (1978).

[36] McBride, Richard D., "A Spike Collective Dynamic Factorization Algorithm for the Simplex Method." Management Science 24 (1978), 1031-1042.

[37] MPS III Mathematical Programming System: User Manual. Ketron, Inc., Arlington, VA (1975).

[38] Murtagh, Bruce A. and Michael A. Saunders, "MINOS: A Large-Scale Nonlinear Programming System (For Problems with Linear Constraints): User's Guide." Technical Report SOL 77-9, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1977).

[39] Orchard-Hays, William, Advanced Linear-Programming Computing Techniques (New York: McGraw-Hill Book Co., 1968).

[40] Parikh, S. C., "A Welfare Equilibrium Model (WEM) of Energy Supply, Energy Demand, and Economic Growth." Technical Report SOL 79-3, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1979).

[41] Perold, Andre, "Fundamentals of a Continuous Time Simplex Method." Technical Report SOL 78-26, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1978).

[42] _____ and George B. Dantzig, "A Basis Factorization Method for Block Triangular Linear Programs." Technical Report SOL 78-7, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1978).

[43] Propoi, A. and V. Krivonozhko, "The Simplex Method for Dynamic Linear Programs." Report RR-78-14, International Institute for Applied Systems Analysis, Laxenburg, Austria (1978).

[44] Reid, J. K., "A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases." Report CSS 20, Computer Science and Systems Division, A.E.R.E. Harwell, England (1975).

[45] _____, "Fortran Subroutines for Handling Sparse Linear Programming Bases." Report AERE-R8269, Computer Science and Systems Division, A.E.R.E. Harwell, England (1976).

[46] Saigal, Romesh, "Block-Triangularization of Multi-Stage Linear Programs," Report ORC 66-9, Operations Research Center, University of California, Berkeley (1966).

[47] Saunders, Michael A., "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating." Sparse Matrix Computations, James R. Bunch and Donald J. Rose, eds. (New York: Academic Press, 1976), 213-226.

[48] _____, "MINOS System Manual." Technical Report SOL 77-31, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University (1977).

[49] Vinson, Ilse, "Triplex User's Guide." User Note 99, SLAC Computing Services, Stanford Linear Accelerator Center (1978).

[50] Wolfe, Philip, "The Composite Simplex Algorithm." SIAM Review 7 (1965), 42-54.

[51] Wollmer, Richard D., "A Substitute Inverse for the Basis of a Staircase Structure Linear Program." Mathematics of Operations Research 2 (1977), 230-239.

A BASIS FACTORIZATION TECHNIQUE FOR STAIRCASE LINEAR PROGRAMS

Philippe Gille and Etienne Loute*

CORE
Université Catholique de Louvain
Louvain-la-Neuve

Basis matrices of staircase linear programs can be rearranged in a block tridiagonal matrix with the property that it can be decomposed into a lower (L) and an upper (U) block triangular matrix. The U matrix has block diagonal submatrices consisting of identity matrices. The basic data and any representation of the inverses of the block diagonal submatrices of L form a substitute for the basis inverse.

We present an algorithm which allows updating of this basis inverse representation for any basic change. Our work is related to the papers of Heesterman and Sandee (1965), Saigal (1966), and Wollmer (1977). Our contribution is threefold: we prove it is always possible to maintain the basis factorization for any basis change. We obtain better bounds for the worst case computational complexity of the updating algorithm. Moreover we present a practical method of controlling the accuracy of the basis inverse representation when it is updated.

*Research supported by the Belgian Department of Science Policy under contract E/I/3.

1. THE STAIRCASE STRUCTURED LINEAR PROGRAMMING PROBLEM

A linear programming problem is said to have a staircase structure of to be a staircase LP problem if the nonzero coefficients of the constraint matrix are confined to certain submatrices on or just below the block diagonal as in figure 1. A partitioning of the row indices, $R_1, \ldots, R_N$ can be associated to the staircase structure. N will be referred to as the *number of periods* and the sets R_i as *periods*. The set R_i contains m_i indices and $m = \Sigma_i m_i$. A column of the matrix will be called a *type i column* if its nonzero elements are confined to rows in R_i and R_{i+1} with at least one nonzero element in R_i.

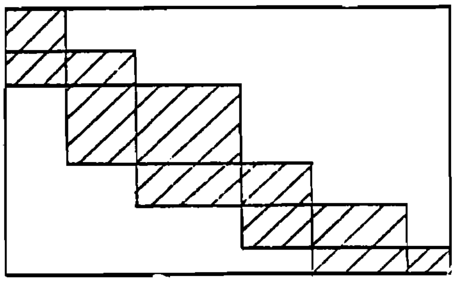


Fig. 1 : Staircase LP problem

2. STAIRCASE BASES

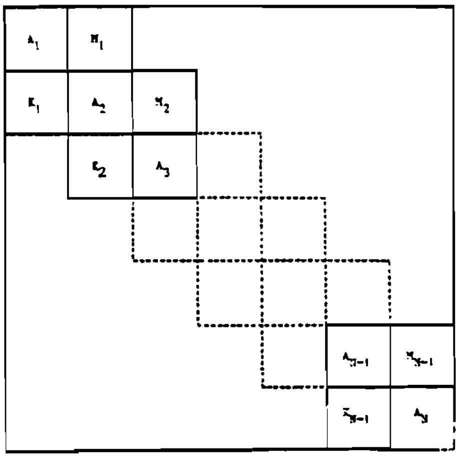


Fig. 2 : Staircase basis

A basis matrix of a staircase LP problem inherits the staircase structure of figure 1. This can be formalized as follows : the nonzero coefficients of the matrix are contained in the submatrices A_i, M_i and K_i of figure 2, these submatrices being respectively of dimension $m_i \times m_i$, $m_i \times m_{i+1}$ and $m_{i+1} \times m_i$. If we denote by

e_k the k^{th} column of the identity matrix of appropriate dimension, we have the following :

For $i \in \{2, \ldots, N-1\}$ and $k \in R_i$

$$M_{i-1} e_k \neq 0 \text{ implies } K_i \, e_k = 0. \qquad (2.1)$$

We denote by BS a basis of a staircase LP problem with the structure of figure 2 and which satisfies (2.1). Note that the k^{th} column of A_i, $A_i e_k$, corresponds necessarily to a basic column of type i-1 or i. If $M_{i-1} e_k \neq 0$, the column is said to be of type $(i-1)^+$. If $K_i e_k \neq 0$, it is said to be of type i^-.

It is known (see e.g. WOLLMER [7] that through a suitable column permutation of BS between adjacent periods, the following matrices, henceforth named block pivot or BP, exist and are non singular :

$$\tilde{A}_1 = A_1 \qquad \tilde{A}_i = \tilde{A}_i - K_{i-1} \, \tilde{A}_{i-1}^{-1} \, M_{i-1} \qquad i = 2, \ldots, N. \qquad (2.2)$$

The matrix BS is then said under a feasible form and is denoted by FBS. Such a matrix can be factorized in two matrices L and U, the first one being

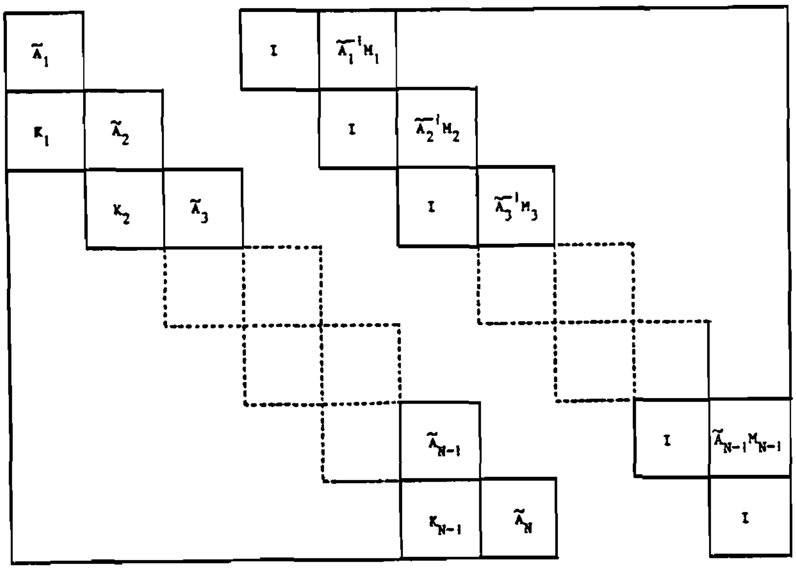


Fig. 3 : Block LU decomposition of FBS

lower block-triangular and the second one upper block-triangular with identity matrices on the principal diagonal (see figure 3). This basis inverse substitute and the related operations of the revised simplex algorithm are presented in WOLLMER [7]. This factorization technique enjoys several advantages : the associated data structure is easy to handle and simpler than in related works where "spikes" in the U matrix can extend beyond the second block diagonal (see e.g. PROPOI and KRIVONOSHKO [5], LOUTE [4]). Any operation of the revised simplex algorithm can be efficiently performed with the original data and the block pivot inverses (BPI) only. Updating the basis inverse reduces to updating the BPI's. This can be done efficiently by means of dyad corrections defined as follows :

$$I + \frac{1}{\lambda} hg' \qquad (2.3)$$

where λ is a nonzero scalar, I the identity matrix, h and g column vectors of same dimension (g' denotes the transposes of g). We restrict ourselves to the use of such multiplicative corrections because they lead to product form substitute for the BPI's.

3. THE PARTIAL UPDATES

Let us denote by v the entering column (see figure 4.a) and by e the column vector with zero elements except the one of index corresponding to the leaving column which is equal to one : this index is supposed to belong to R_q. Let us denote it $\ell_q \in R_q$. In fact, when the updating begins, the partial updates of these columns are available, i.e. the vectors $h = L^{-1}v$ and $g' = e'U^{-1}$ (see figures 4 b,c). Their subvectors are given by

$$g'_q = e'_{\ell_q}, \qquad g'_{p+1} = -g'_p \widetilde{A}_p^{-1} M_p \quad \text{for } p \geqslant q \qquad (3.1)$$

$$h_{i-1} = \widetilde{A}_{i-1}^{-1} d, \quad h_i = \widetilde{A}_i^{-1}(b-K_{i-1}h_{i-1}), \quad h_{p+1} = -\widetilde{A}_{p+1}^{-1} K_p h_p \text{ for } p \geqslant i. \qquad (3.2)$$

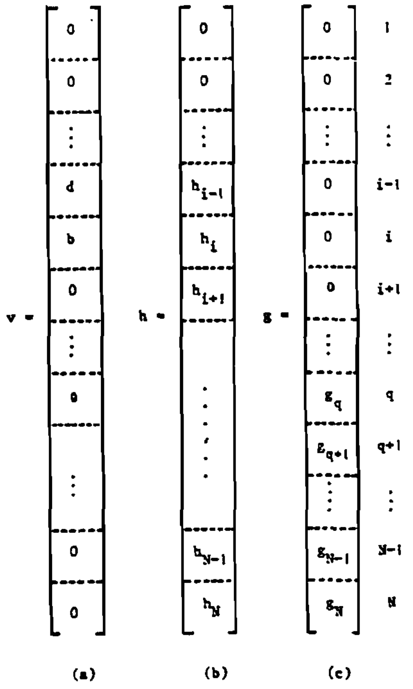


Fig. 4 : The entering column and the partial updates

We shall refer to the pivot element (supposedly nonzero) $\xi = g'h$ as the exchange value . A scalar α is said "almost zero", noted $\alpha \simeq 0$, if $\left|\frac{\alpha}{\xi}\right| \leqslant \eta$, where η is a small positive number chosen in order to satisfy the following properties :

1) If $\alpha \simeq 0$ $\quad \xi - \alpha \not\simeq 0$ and $\xi + \alpha \not\simeq 0$ (3.3)

2) Let s and t be vectors; if $s't \not\simeq 0$ then there exists an index k such that the components s_k and t_k satisfy simultaneously $s_k \not\simeq 0$ and $t_k \not\simeq 0$. (3.4)

Remark on the notation : At any stage of the algorithm, the BPI's and the partial updates, i.e. the sequences $\widetilde{A}_j^{-1}$, h_j, g_j, $j = 1, \ldots, N$, are at hand and sometimes modified by dyad corrections. To limit the number of symbols used, we shall not introduce a new notation after a correction. We use the symbol ← which means "replaced by", and which allows a dynamic use of the notation.

For example the following sequence of transformations

$$\underline{\widetilde{A}}_j^{-1} = (I + st')\widetilde{A}_j^{-1}$$
$$\underline{h}_j = (I + st')h_j$$
$$\underline{\underline{\widetilde{A}}}_j^{-1} = (I + s\underline{h}'_j)\underline{\widetilde{A}}_j^{-1}$$

where s and t are column vectors, will be written

$$\widetilde{A}_j^{-1} \leftarrow (I + st')\widetilde{A}_j^{-1} \; ; \; h_j \leftarrow (I + st')h_j \; ; \; \widetilde{A}_j^{-1} \leftarrow (I + sh'_j)\widetilde{A}_j^{-1} ;$$

to be read from left to right.

4. *UPDATING : COLUMN PERMUTATION*

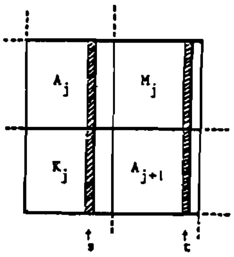


Fig. 5 : Column Permutation

In the next pages we often use the permutation of two columns in two subsequent BP's. Let us consider as in figure 5, the columns s of type j^- and t of type j^+. The permutation is feasible if and only if

$$e_s' \tilde{A}_j^{-1} M_j e_t \neq 0. \qquad (4.1)$$

Then the sequence of computations is the following :

Step P(j) : Compute $P_1 = I - \frac{1}{e_s'\tilde{A}_j^{-1}M_je_t} (\tilde{A}_j^{-1}M_je_t - e_s)e_s'$

$$P_2 = I - e_t(e_t' + e_s' \tilde{A}_j^{-1}M_j)$$

$$P_3 = I - \frac{1}{e_s'\tilde{A}_j^{-1}M_je_t} e_t(e_s' \tilde{A}_j^{-1}M_j+e_t').$$

Then, $\tilde{A}_j^{-1} \leftarrow P_1\tilde{A}_j^{-1}$; $\tilde{A}_{j+1}^{-1} \leftarrow P_2\tilde{A}_{j+1}^{-1}$; $h_{j+1} \leftarrow P_2 h_{j+1} + e_te_s'h_j$;

next $h_j \leftarrow P_1h_j$; $g_{j+1}' \leftarrow g_{j+1}' P_3$.

Modify the definition of A_j, M_j, K_j, A_{j+1} (see figure 5).

By "modify the definition" we mean to permute the correspondinp values of the pointers which are used to pick up the original data.

The other elements (except for A_j and A_{j+1}) remain unchanged.

5. *UPDATING : CENTRAL CASES*

As mentioned before, the algorithmic procedures presented in the paper produce the corrections for the BPI's successively, in the natural order : 1, 2, ..., N. For the BPI's of indices between i and q, we distinguish several cases according to the types of the entering and leaving columns (†). This is the object of Section 6. However, for each case, there exists a period say p, where the algorithmic procedure can be embedded in a general framework : *the central cases* where singularity may occur. These central cases are linked as follows : the *principal central* case generally occurs at the period equal to max(i,q). It is followed by the *auxiliary central case* in the next period. This last case then occurs repeatedly up to period N.

5.1. *Auxiliary central case at period p (p > max(i,q))*

At period p, the BP has to be modify in the following way :

$$\tilde{A}^*_p = \tilde{A}_p\left(I + \frac{1}{\Delta_{p-1}} h_p g'_p\right) \tag{5.1}$$

where $\Delta_{p-1} \neq 0$ and $|\xi| = |\Delta_{p-1} + \sum_{\ell=p}^{N} g'_\ell \, h_\ell| \neq 0.$ (5.2)

The input at this stage of the algorithm consists of p and Δ_{p-1}.

We define the value $\Delta_p = \Delta_{p-1} + g'_p \, h_p$. The non singularity of $\tilde{A}^*_p$ depends on the value of Δ_p.

5.1.1. $\Delta_p \neq 0$ *(No singularity)*

We simply perform the following step :

$$\boxed{\text{Step AC1:} \quad \tilde{A}_p^{-1} \leftarrow \left(I - \frac{1}{\Delta_p} h_p g'_p\right) \tilde{A}_p^{-1}} \tag{5.3}$$

and we find that the following BP becomes

$$\tilde{A}^*_{p+1} = \tilde{A}_{p+1}\left(I + \frac{1}{\Delta_p} h_{p+1} \, g'_{p+1}\right). \tag{5.4}$$

Therefore, after the incrementation of p, we are led back to formula (5.1).

(†) In sections 5 and 6, the type of the entering column will be noted i-1 and the type of the leaving one q.

5.1.2. $\Delta_p \simeq 0$ (*Singularity*)

From (5.2), (3.1) and (3.2) we find :

$$g'_p \tilde{A}_p^{-1} M_p \left(I + (\tilde{A}_{p+1}^{-1} M_{p+1})\ (\tilde{A}_{p+2}^{-1} K_{p+1}) + \dots\right) \tilde{A}_{p+1}^{-1} K_p h_p \not\simeq 0. \tag{5.5}$$

Consequently, there must exist two indices s and t such that

$$g'_p \tilde{A}_p^{-1} M_p e_t \not\simeq 0 \quad K_p e_s \not\simeq 0 \quad e'_s h_p \not\simeq 0. \tag{5.6}$$

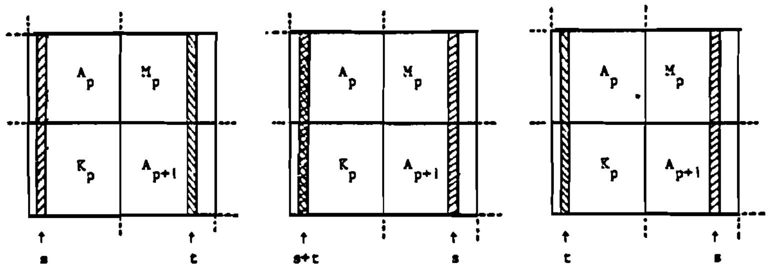
As $M_p e_t \not\simeq 0$ and $K_p e_s \not\simeq 0$, we have $K_{p+1} e_t = 0$, $M_{p-1} e_s = 0$ and, as a consequence, it is possible at least from a structural point of view to permute the columns of indices s and t, as in figure 5 with j = p. However the condition (4.1) is required, so let us distinguish both possibilities.

$e'_s \tilde{A}_p^{-1} M_p e_t \not\simeq 0$ (weak singularity)

The permutation is performed by means of the step P(p). The new value of $\Delta_p = \Delta_{p-1} + g'_p h_p$ is no longer almost zero and the algorithm is unlocked. The step AC1 is performed and we are led back to the situation above.

$e'_s \tilde{A}_p^{-1} M_p e_t \simeq 0$ (strong singularity)

In this situation, it is no longer possible to permute the columns s and t; this operation must be splitted as illustrated in figure 6.



Initial positions Prior substitution Posterior substitution

Fig. 6. : Column permutation in case of strong singularity

The prior substitution induces the following operations

<u>Step ACO</u> : Compute $S_1 = \left(I - \frac{1}{\Delta_1}\widetilde{A}_p^{-1}M_p e_t e_s'\right)$ where $\Delta_1 = 1 + e_s'\,\widetilde{A}_p^{-1}M_p e_t \not\approx 0$ (5.7)

$$S_2 = I - e_t(e_s'\widetilde{A}_p^{-1}M_p + 2e_t') \tag{5.8}$$

$$S_3 = I - \frac{1}{\Delta_1}e_t(e_s'\widetilde{A}_p^{-1}M_p + 2e_t'). \tag{5.9}$$

$$\text{Then } \widetilde{A}_p^{-1} \leftarrow S_1\widetilde{A}_p^{-1};\ h_p \leftarrow S_1 h_p\ ;\ \widetilde{A}_{p+1}^{-1} \leftarrow S_2\,\widetilde{A}_{p+1}^{-1}; \tag{5.10}$$

$$h_{p+1} \leftarrow S_2 h_{p+1} + e_s'h_p e_t\ ;\ g_{p+1}' \leftarrow g_{p+1}'S_3. \tag{5.11}$$

After the preliminary corrections, the new value of $\Delta_p = \Delta_{p-1} + g_p'h_p$ is non almost zero, and the step AC1 can be performed.

Finally, the posterior substitution has to be done, the resulting sequence is divided in two parts.

<u>Step AC2</u> : Update the data A_p, M_p, K_p, A_{p+1} as in figure 6.

$$\widetilde{A}_p^{-1} \leftarrow \left(I + \frac{1}{\Delta_2}\widetilde{A}_p^{-1}M_p e_t e_s'\right)\widetilde{A}_p^{-1} \text{ where } \Delta_2 = 1 - e_s'\widetilde{A}_p^{-1}M_p e_t = 1 - \frac{\Delta_p^*}{\Delta_p\Delta_1} \not\approx 0 \tag{5.12}$$

(Δ_p^* is the old value of Δ_p which was almost zero).

Compute and store $S_4 = I - e_t e_s'\widetilde{A}_p^{-1}M_p$. (5.13)

<u>Step AC3</u> : $\widetilde{A}_{p+1}^{-1} \leftarrow S_4\,\widetilde{A}_{p+1}^{-1}$. (5.14)

Note that the step AC3 is performed only after the auxiliary central case is initiated for p+1 (i.e. after the step AC1 for p+1) because the logic of this correction is the following. The prior substitution induces the modification of the BPI's p and p+1. The correction (5.3) of AC1 affects the BPI p and all the following ones, including p+1. Finally the posterior substitution is carried out and modifies the BPI's p and p+1. Hence, the correction of the BPI p+1 resulting from the posterior correction must be stored and performed later.

The flow chart of the auxiliary central case is described in figure 7. We have introduced an array of logical variables DEG; the value of DEG(p) is YES if strong singularity occurs in period p, i.e. if the step AC3 is to be performed at period p+1.

5.2. *Principal central case*

The BP corresponding to period p has to be modified as follows.:

$$\tilde{A}^*_p = \tilde{A}_p\left[I + \frac{1}{g'_p e_{\ell_p}} e_{\ell_p} (u'_p - g'_p)\right] \left\{I + \frac{1}{\alpha}\left[(I - e_{\ell_p} u'_p)h_p + (g'_p h_p - (\alpha+\varepsilon))e_{\ell_p}\right]u'_p\right\} \quad (5.15)$$

where $g'_p e_{\ell_p} \not\simeq 0, \quad \varepsilon \simeq 0, \quad u'_p e_{\ell_p} = 1, \quad \alpha \not\simeq 0$ (5.16)

and for each index t if $M_{p-1} e_t = 0$ then $u'_p e_t = g'_p e_t = e'_{\ell_p} e_t$ (5.17)

and $|\xi| = |\varepsilon - \sum_{\ell=p}^{N} g'_\ell h_\ell|$. (5.18)

The input for this procedure consists of p, ℓ_p, ε, α, u'_p. We define $\Delta_p = g'_p h_p - \varepsilon$; and the situations to be studied are the same as in the previous section.

5.2.1. $\Delta_p \not\simeq 0$ *(No singularity)*

The BPI is updated in two steps.

<u>Step PC1.1</u> : Compute $E_1 = I - e_{\ell_p} (u'_p - g'_p)$ (5.19)

$\tilde{A}^{-1}_p \leftarrow E_1 \tilde{A}^{-1}_p \; ; \; h_p \leftarrow E_1 h_p$. (5.20)

<u>Step PC1.2</u> : Compute $E_2 = I - \frac{1}{\Delta_p} (h_p - (\alpha + \varepsilon) e_{\ell_p}) u'_p$ (5.21)

$\tilde{A}^{-1}_p \leftarrow E_2 \tilde{A}^{-1}_p$.

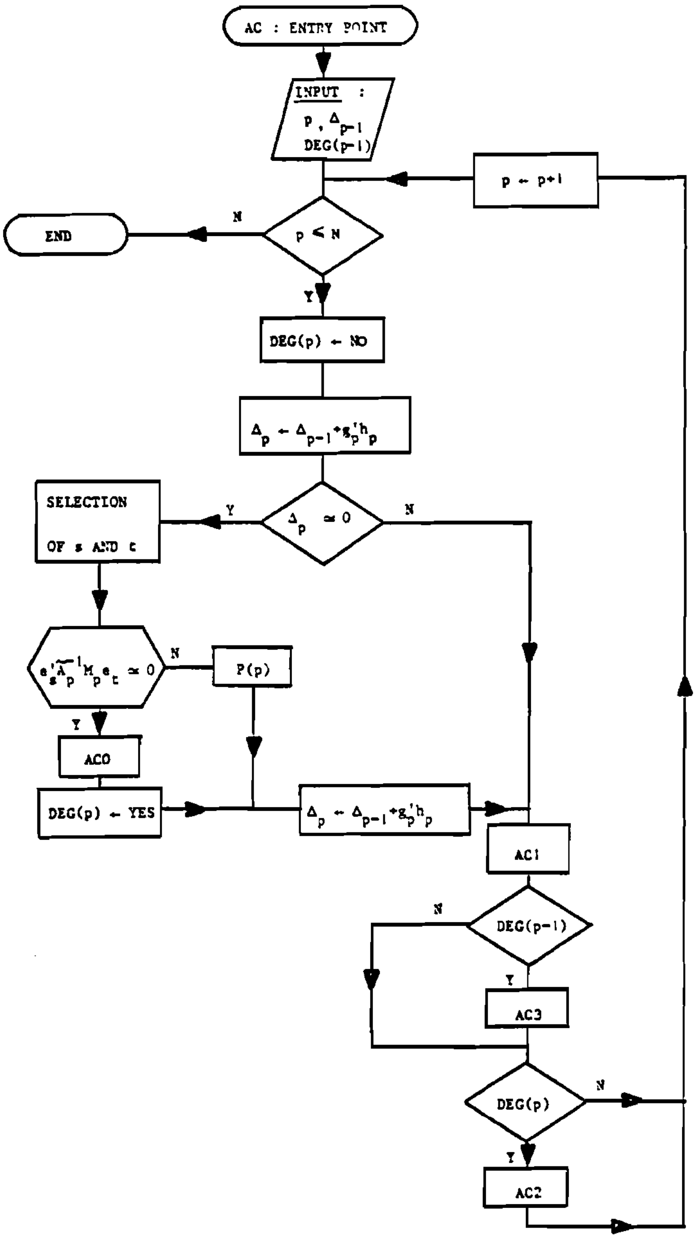


Fig. 7 : Flowchart of algorithm AC

Remark : Y stands for YES and N for NO.

It is easy to see that the next BPI has to be corrected as in formula (5.3). We then branch to the auxiliary central case with DEG(p) = NO.

5.2.2. $\Delta_p \simeq 0$ *(Singularity)*

As in 5.1.2. we select two indices s and t and we try to permute the corresponding columns.

$e'_s\tilde{A}_p^{-1}M_p e_t \not\simeq 0$ (weak singularity)

The permutation is feasible and can be performed. Note that the steps P(p) and PC1.1 commute, i.e. they can be applied in any order.

$e'_s\tilde{A}_p^{-1}M_p e_t \simeq 0$ (strong singularity)

The permutation is splitted in prior and posterior substitution. The prior substitution is performed by the step PCO which is the same as ACO, and which commutes with PC1.1. Δ_p is then recomputed and PC1.2 is performed. The step PC2 is identical to AC2; AC3 does not occur here.

The flow chart of the principal central case (figure 8) is very similar to the one of the auxiliary central case. In fact several steps are common to both of them.

6. UPDATING : EXCHANGE ALGORITHM

The present section is devoted to the presentation of the algorithms handling the BPI's of indices between i and q. Obviously, in a block LU factorization the BP's of indices less than min(i,q) remain unchanged.

We successively consider three subseotions according to the relative values of i and q; each subsection being subdivisided to consider the type of the leaving column (q^- or $(q-1)^+$).

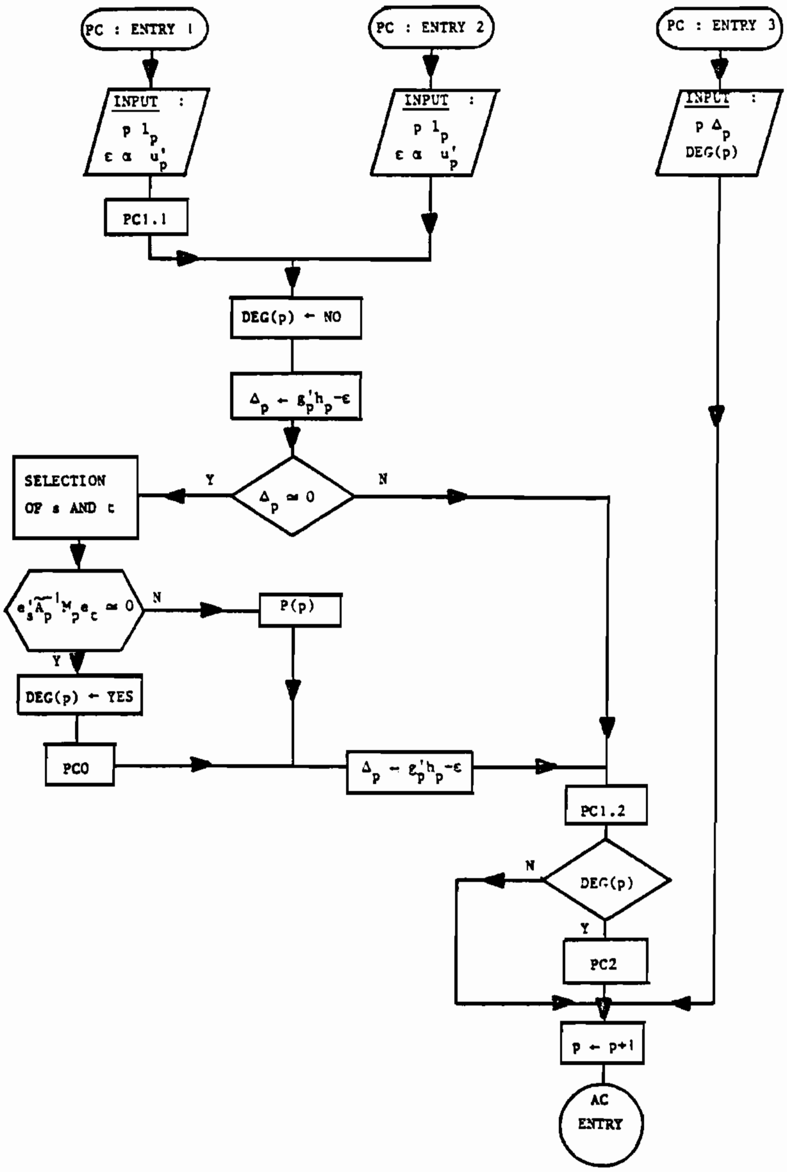


Fig. 8 : Flowchart of algorithm PC

Remark : The meaning of the different entries will be explained in the next section

6.1. $i = q$

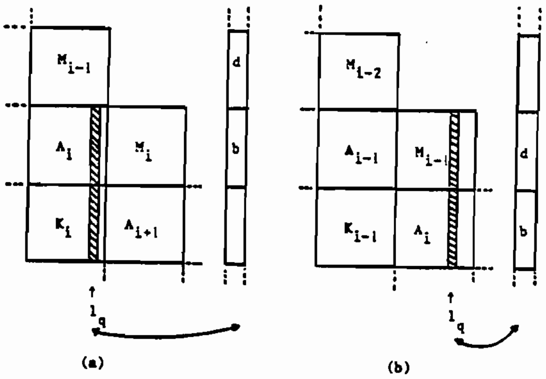


Fig. 9 : Case where i = q

6.1.1. *The type of the leaving column is q^-* (*see figure 9(a)*)

The basis data is modified as follows

$$M^*_{i-1} = M_{i-1} + de'_{\ell_q} \;,\; A^*_i = A_i + (b - A_i e_{\ell_q})e'_{\ell_q} \;,\; K^*_i = K_i - K_i e_{\ell_q} e'_{\ell_q} \tag{6.1}$$

This leads to
$$\widetilde{A}^*_i = \widetilde{A}_i(I + (h_i - e_{\ell_q})e'_{\ell_q}) . \tag{6.2}$$

This allows us to branch to the principal central case with the following steps.

Step SA0 : Set $g'_p = u'_p = e'_{\ell_q}$, $\alpha = 1$, $\varepsilon = 0$, $p = q$.

Step SA1 : Modify the definitions of M_{i-1}, A_i, K_i (see figure 9(a)).

6.1.2. *The type of the leaving column is $(q-1)^+$* (*see figure 9(b)*)

The modifications leads to the same expression as (6.2) and we just replace SA1 by

Step SA2 : Modify the definitions of M_{i-1}, A_i (see figure 9(b)).

6.2. $i < q$

6.2.1. *The type of the leaving column is q-*

If $h_q \simeq 0$, then from (3.2) and $\xi = g'h$ we get $\xi \simeq 0$ which is absurd. Thus

$$h_{k+1} = -\tilde{A}_{k+1}^{-1} K_k h_k \not\simeq 0 \text{ for } k = i+1, \ldots, q \qquad (6.3)$$

and there exists a sequence of indices ℓ_k, $k = i, \ldots, q-1$, such that simultaneously $\Delta_k = e'_{\ell_k} h_k \not\simeq 0$ and $K_k e_{\ell_k} \not\simeq 0$ for $k = i, \ldots, q-1$. This suggests the modifications described in figure 10.

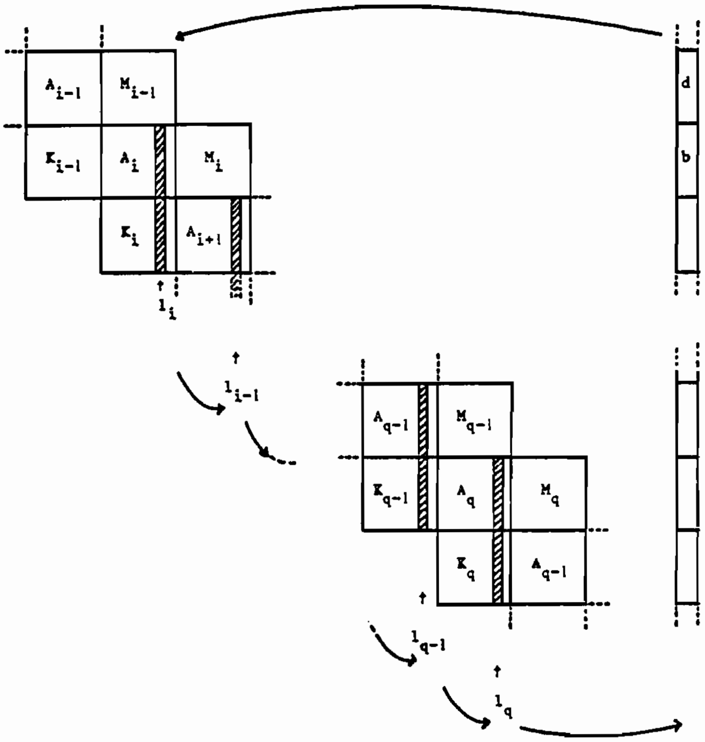


Fig. 10 : Case $i < q$

The exchange algorithm for this case makes use of the following property.

Property 6.1

If the following transformations occur in period k

$$M_k^* = M_k + A_k e_{\ell_k} e'_{\ell_{k+1}} \text{ while } M_k\, e_{\ell_{k+1}} = 0 \text{ and } K_k^* = K_k - K_k e_{\ell_k} e'_{\ell_k}$$

and if $\tilde{A}_k^* = \tilde{A}_k\left(I - e_{\ell_k} e'_{\ell_k} + \frac{1}{\Delta_{k-1}} h_k u'_k\right)$ with $u'_k e_{\ell_k} = 1$, then the BPI is modified as follows.

Step SBO(k) : Compute $D = I - e_{\ell_k}(u'_k - e'_{\ell_k})$

$$\tilde{A}_k^{-1} \leftarrow D\, \tilde{A}_k^{-1} \;;\; h_k \leftarrow D\, h_k \tag{6.5}$$

$$\tilde{A}_k^{-1} \leftarrow \left[I - \frac{1}{\Delta_k}(h_k - \Delta_{k-1} e_{\ell_k}) e'_{\ell_k}\right] \tilde{A}_k^{-1}. \tag{6.6}$$

Moreover if $A_{k+1}^* = A_{k+1} + K_{k+1} e_{\ell_k} e'_{\ell_{k+1}} - A_{k+1}\, e_{\ell_{k+1}} e'_{\ell_{k+1}}$

then
$$\tilde{A}_{k+1}^* = \tilde{A}_{k+1}\left(I - e_{\ell_{k+1}} e'_{\ell_{k+1}} + \frac{1}{\Delta_k} h_{k+1} u'_{k+1}\right) \tag{6.7}$$

$$\text{where } u'_{k+1} = \frac{\Delta_k}{\Delta_{k-1}} u'_k\, \tilde{A}_k^{-1} M_k + e'_{\ell_{k+1}} \tag{6.8}$$

with $\tilde{A}_k^{-1}$ resulting from the step SBO(k).

This property is proved by applying the Sherman-Morrison formula twice in succession. Now it is easy to show that

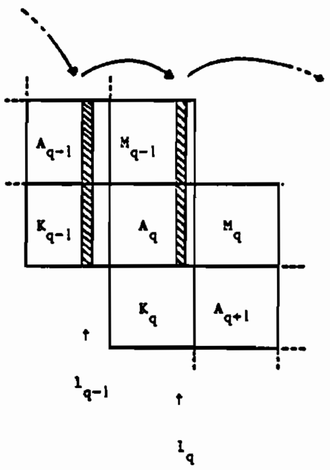
$$\tilde{A}_i^* = \tilde{A}_i(I + h_i e'_{\ell_i} - e_{\ell_i} e'_{\ell_i}) \tag{6.9}$$

and if we state $u'_i = e'_{\ell_i}$ and $\Delta_{i-1} = 1$ then the property 6.1 can be used recursively for k = i, ..., q-1. After each step SBO(k) we have to perform the following step :

Step SB1(k) : Compute u'_q as in (6.8).

Modify the definition of M_{k-1}, A_k and K_k

6.2.2 . *The type of the leaving column is* $(q-1)^+$



The only difference with figure 10 is given in figure 11.

The property 6.1. is used in the same way as in the preceding case for k = i, ..., q-1 and the only modification is the formulation of u'_q which is now

$$u'_q = \frac{\Delta_{q-1}}{\Delta_{q-1}} u'_{q-1} \tilde{A}^{-1}_{q-1} M_{q-1} (I - e_{\ell_q} e'_{\ell_q}) + e'_{\ell_q} \quad (6.10)$$

The step SB1(q) is replaced by :

Fig. 11 : The leaving column is of type $(q-1)^+$

Step SB2 : Compute u'_q as in (6.10)
Modify the definition of M_{q-1} and A_q

Now it is easy to see that (5.15) reduces to (6.7) with k+1 = q, if we take p=q, ε = 0, $\alpha = \Delta_{p-1}$ and $g'_p = e'_{\ell_p}$ in (5.15). The conditions (5.16), (5.17) and (5.18) are satisfied and we can branch to ENTRY 1 of PC.

6.3. $i > q$

6.3.1. *The type of the leaving column is* $(q-1)^+$

From (3.1) and the fact that $\xi \neq 0$, we deduce that $g'_{i-1} \neq 0$ and that it is possible to select a sequence of indices ℓ_k, k = q+1, ..., i-1, such that $\Gamma_{k-1} = g'_k e_{\ell_k} \neq 0$ for k = q, ..., i-1 (ℓ_q is already defined) and $K_k e_{\ell_k} = 0$ for k = q, ..., i-1. Moreover if $\Delta_{i-1} = g'_{i-1} h_{i-1} \simeq 0$ then $\Gamma_{i-1} \neq 0$ and $K_i e_{\ell_i} = 0$.

The figure 12 illustrates the modifications (the problem of the index ℓ_i is explained below).

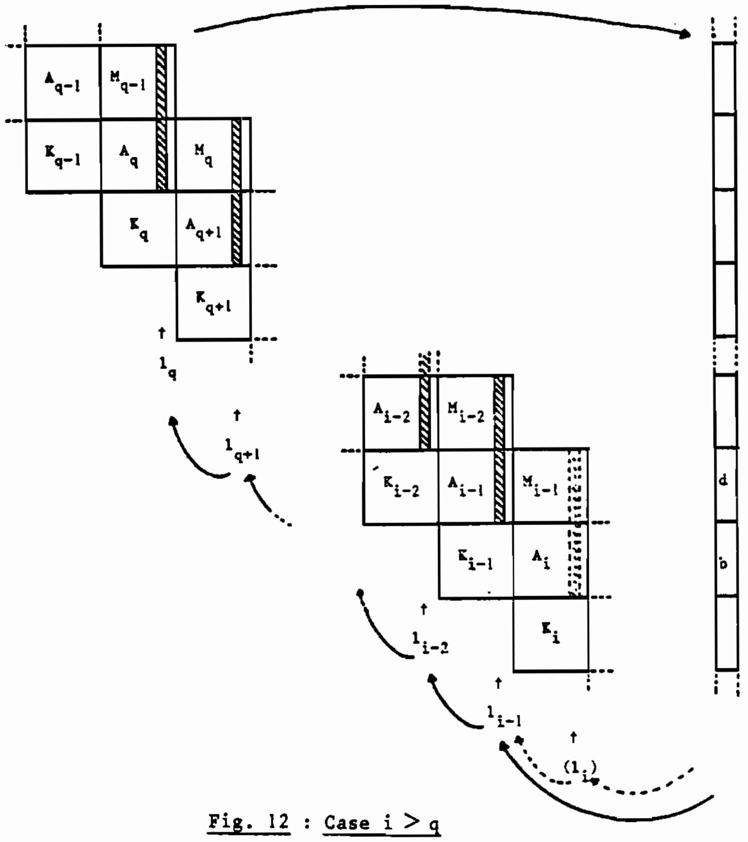


Fig. 12 : Case i > q

As in subsection 6.2, we introduce a property on which the algorithmic process is based.

Property 6.2

If the following transformations occur in period k

$$M_k^* = M_k - M_k e_{\ell_{k+1}} e'_{\ell_{k+1}}, \quad K_k^* = K_k + A_{k+1} e_{\ell_{k+1}} e'_{\ell_k}$$

and $$\tilde{A}^*_k = \tilde{A}_k + M_k e_{\ell_{k+1}} e'_{\ell_k} - \frac{1}{\Gamma_{k-1}} \tilde{A}_k e_{\ell_k} g'_k \qquad (6.11)$$

Then the BPI is transformed as follows :

Step SCO(k) : $\tilde{A}_k^{-1} \leftarrow \left(I - e_{\ell_k}(e'_{\ell_k} - g'_k)\right)\tilde{A}_k^{-1}$,

$$\tilde{A}_k^{-1} \leftarrow \left[I + \frac{1}{\Gamma_k}(\tilde{A}_k^{-1} M_k e_{\ell_{k+1}} - e_{\ell_k})e'_{\ell_k}\right]\tilde{A}_k^{-1}.$$

Moreover if $A^*_{k+1} = A_{k+1} + (M_{k+1} e_{\ell_{k+2}} - A_{k+1} e_{\ell_{k+1}})e'_{\ell_{k+1}}$, we find for $\tilde{A}_{k+1}$ the same expression as (6.11) after incrementation of k.

At period q the new BP is

$$\tilde{A}^*_q = \tilde{A}_q + M_q e_{\ell_{q+1}} e'_{\ell_q} - \frac{1}{\Gamma_{q-1}} \tilde{A}_q e_{\ell_q} g'_q$$ (Note that $\Gamma_{q-1} = 1$, because $g'_q = e'_{\ell_q}$)..

This initiates the algorithm which is applied up to i-1 or i depending on the value of Δ_{i-1}. The step SCO(k) is performed and completed by

Step SC1(k) : Modify the definitions of M_{k-1}, A_k and K_k.

$\Delta_{i-1} \neq 0$

From (6.11) with k = i-1, we can compute the BPI by the following step :

Step SC2 : Compute $E = I - e_{\ell_{i-1}}(e'_{\ell_{i-1}} - g'_{i-1})$,

$$\tilde{A}_{i-1}^{-1} \leftarrow E\,\tilde{A}_{i-1}^{-1} \quad , \quad h_{i-1} \leftarrow E\, h_{i-1} ,$$

$$\tilde{A}_{i-1}^{-1} \leftarrow \left(I - \frac{1}{\Delta_{i-1}}(h_{i-1} - e_{\ell_{i-1}})e'_{\ell_{i-1}}\right)\tilde{A}_{i-1}^{-1} .$$

Modify the definitions of M_{i-1}, A_i and K_i.

And we find $\tilde{A}^*_i = \tilde{A}_i \left(I + \frac{1}{\Delta_{i-1}} h_i g'_i\right)$. Hence we can branch to ENTRY 3 of PC with p = i-1 and DEG(p) = NO.

$\Delta'_{i-1} \simeq 0$

We use the index ℓ_i to perform a supplementary iteration ($k = i-1$) with property 6.2. Taking the following substitutions into account :

$$M^*_{i-1} = M_{i-1} - (M_{i-1}e_{\ell_i} - d)e'_{\ell_i} \quad , \quad A^*_i = A_i - (A_i e_{\ell_i} - b)e'_{\ell_i}$$

leads to the formula

$$\tilde{A}^*_i = \tilde{A}_i\left[I + \frac{1}{e'_{\ell_i} g_i} e_{\ell_i}(e'_{\ell_i} - g'_i)\right] \left\{I + \left[(I - e_{\ell_i}e'_{\ell_i})h_i + (g'_i h_i - (1 + \Delta_{i-1}))e_{\ell_i}\right]e'_{\ell_i}\right\}$$

Consequently, we branch to ENTRY 1 of PC with

$$\varepsilon = \Delta_{i-1}\ ,\ p = i\ ,\ u'_p = e'_{\ell_p}\ ,\ \alpha = 1.$$

6.3.2. *The type of the leaving column is q^-*

$i > q+1$

As before we see that $g'_{i-1} \not\simeq 0$. Thus there exists an index ℓ_{q+1} such that $e'_{\ell_q} \tilde{A}^{-1}_q M_q e_{\ell_{q+1}} \not\simeq 0$ and we apply the permutation $P(q)$ with $s = \ell_q$ and $t = \ell_{q+1}$; q is incremented and we branch to the preceding case.

$i = q+1$

As before we define $\Delta_{i-1} = g'_{i-1}h_{i-1}$. If $\Delta_{i-1} \simeq 0$, from a similar argument we deduce the existence of an index ℓ_i such that $e'_{\ell_{i-1}} \tilde{A}^{-1}_{i-1} M_{i-1} e_{\ell_i} \not\simeq 0$. The permutation is performed, q is incremented and we are in the case $i = q$ and the type of the leaving column is $(q-1)^+$. If $\Delta_{i-1} \not\simeq 0$, it is possible to perform the exchange (see figure 13). This is the object of the step SC3.

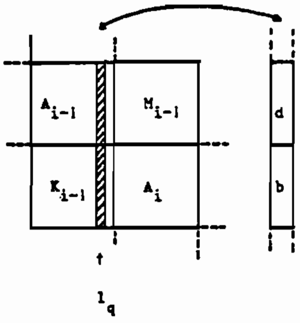


Fig. 13 : Case where $i = q+1$

<u>Step SC3</u> : $\widetilde{A}^{-1}_{i-1} \leftarrow \left(I - \frac{1}{\Delta_{i-1}} (h_{i-1} - e_{\ell_{i-1}}) e'_{\ell_{i-1}}\right) A^{-1}_{i-1}$.

Modify the definition of A_{i-1}, K_{i-1}.

It is easy to prove that

$$\widetilde{A}^*_i = \widetilde{A}_i \quad (I + \frac{1}{\Delta_{i-1}} h_i g'_i).$$

Consequently we branch to ENTRY 3 of PC with $p = i-1$ and DEG(p) = NO. All the computations and branching of section 6 are schematized in the flow-chart of figure 14.

A set of examples corresponding to all cases arising in the exchange algorithm is presented in GILLE [3].

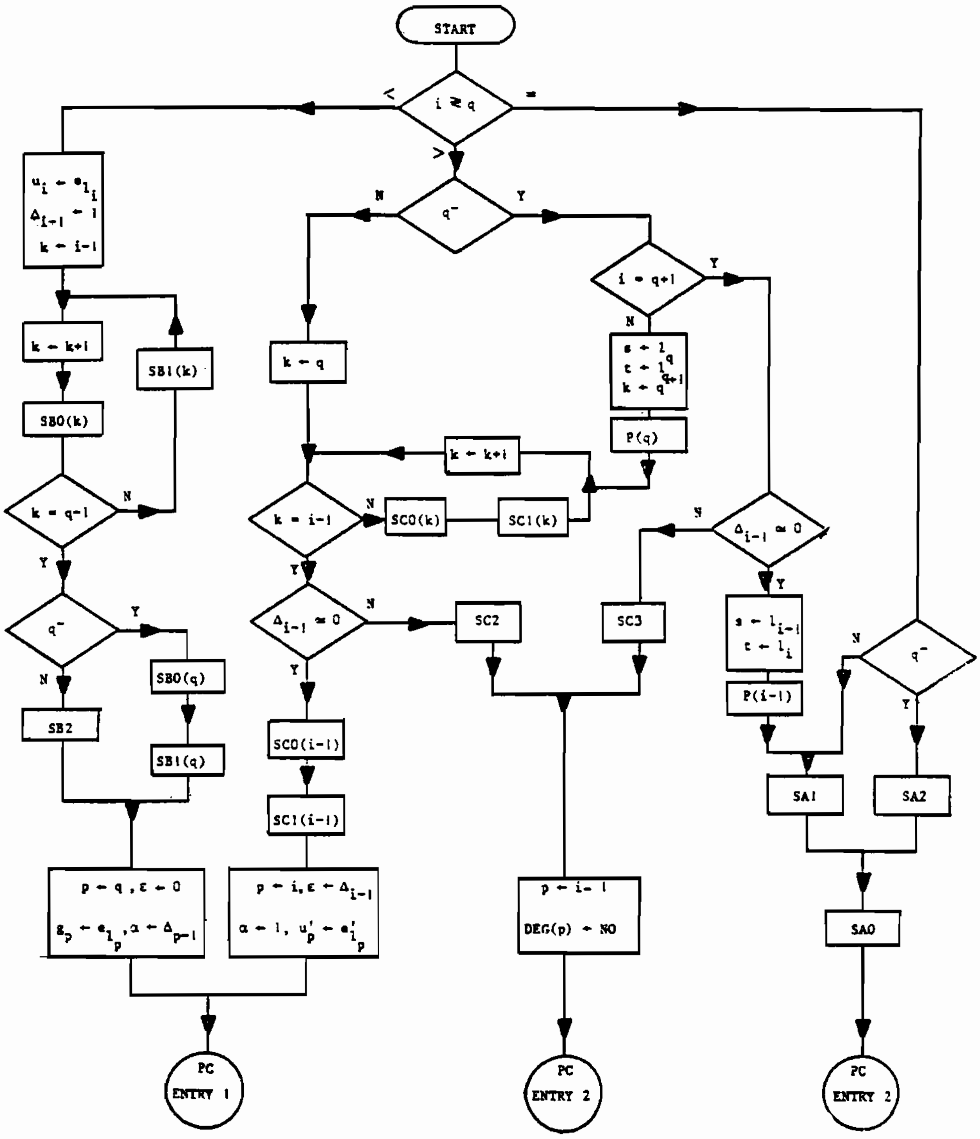


Fig. 14 : Flowchart of the exchange algorithm

Remark : ◇ q^- means : is q^- the type of the leaving variable ?

7. CONCLUSIONS

We have presented a basis inverse substitute for staircase linear programs. The computations of the revised simplex algorithm (simplex multipliers, updating of a column, etc ...) are straightforward and have been omitted in the paper (e.g. WOLLMER [7]).

We rather concentrated on the updating of the basis inverse substitute. The resulting algorithm reduces to updating submatrices of the size of the periods by means of dyad corrections. It is guaranteed that the determinant of the dyad corrections will not be small as compared to the pivot element. The number of corrections required for a basis change is smaller than in other related works. According to WOLLMER [7] the number of dyad corrections to be applied to the block pivots (to their inverses) between the periods i-1 and q (respectively of the entering and leaving column) is 1, 2, 3, ..., $|q-i| + 1$. The next block pivots are all updated by $|q-i| + 1$ dyad corrections. With our algorithm, the number of dyad corrections is 1, 2, ..., 2 between the period i and q. The next block pivots, are updated in general by one dyad correction or by at most 5 dyad corrections if weak or strong singularity occurs.

Experience with other basis factorization techniques making use of dyad corrections (LOUTE [4], HO and LOUTE [2]) shows that singularity of corrections is not frequent. However we must be able to handle such cases.

The complexity of our updating algorithm is linear in terms of the number of periods. This constrats with algorithms proposed by ZVJAGINA [8], WINKLER [6] and LOUTE [4] where the complexity of the updating is in $\log_2$ of the number of periods. However, this result is obtained at the expense of the simplicity of the data structure and at the expense of the sparsity of the basis inverse substitute.

From a sparsity preserving point of view the block LU factorization is inferior to the pure LU factorization. In fact it is possible to adapt our work to the standard LU factorization. The basis inverse substitute would consist of the original data and the LU factorization of the block pivots. The updating algorithm would require to update the L and U factors of the block pivots modified by rank one corrections. This will be the object of another paper. This basis inverse substitute would be more compact than the standard LU factorization applied to the whole basis : only a part of L and U is kept explicitely. Moreover it would be possible to make use of recent results of FOURER [1] on gaussian elimination of staircase systems.

REFERENCES

[1] FOURER R., "Sparse Gaussian Elimination of Staircase Systems", Technical Report SOL 79-17, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, 1979.

[2] HO J. and E. LOUTE, "A Comparative Study of Two Methods for Staircase Linear Programs", *Transactions on Mathematical Software*, ACM, 6, 1980, pp. 17-30.

[3] GILLE Ph., "Contributions to Dynamic Linear Programming", Ph D Thesis, Faculté des Sciences, Facultés Universitaires Notre-Dame de la Paix, Namur, 1980.

[4] LOUTE E., "A Revised Simplex Method for Block Structured Linear Programs", Ph D Thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, Louvain-La-Neuve, 1976.

[5] PROPOI A. and V. KRIVONOZHKO, "The Simplex Method for Dynamic Linear Programs", IIASA RR-78-14, Laxenburg Austria, 1978.

[6] WINKLER C., "Basis Factorization for Block-Angular Linear Programs : Unified Theory of Partionning and Decomposition Using the Simplex Method", IIASA RR-74-22, Laxenburg Austria, 1974.

[7] WOLLMER R.D., "A Substitute Inverse for the Basis of a Staircase Structured Linear Program", *Mathematics of Operations Research*, 2, 1977, pp. 230-239.

[8] ZVJAGINA R.A., "Multilevel Factorization in Linear Programming" (in Russian), *Mathematische Operations Forschung und Statistik*, 6, 1973, pp. 427-443.

COMPUTATIONAL METHODS FOR SOLVING TWO-STAGE STOCHASTIC LINEAR PROGRAMMING PROBLEMS*

Peter Kall**

Institut für Operations Research und Mathematische Methoden
der Wirtschaftswissenschaften
Universität Zürich

Approximating a given continuous probability distribution of the data of a linear program by a discrete one yields solution methods for the stochastic linear programming problem with complete fixed recourse. For a basis decomposition procedure along the lines of Strazicky, the reduction of the computational work relative to the usual revised simplex method is analyzed. Furthermore, an alternative method is proposed where the optimal value is approximated by refining particular discrete distributions.

*Reprinted with permission from *J. Appl. Math. Phys. (ZAMP)* 30(1979), 261–271. Copyright © 1979 Birkhäuser Verlag, Basel.

**The author wishes to express his professional appreciation of Herr Prof. Dr. Eduard Stiefel.

I. Introduction

One standard model in stochastic linear programming is the stochastic program with complete fixed recourse

$$\min\, [c'x + EQ(x, \omega)]^{1)}$$
$$\text{s.t. } Dx = d, \quad x \in \mathbb{R}^n_+, \tag{1}$$

where D is a real $(m \times n)$-matrix, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}^m$, and

$$Q(x, \omega) = \min\, \{q'y \mid Wy = b(\omega) - A(\omega)x, y \in \mathbb{R}^\nu_+\}, \tag{2}$$

where $q \in \mathbb{R}^\nu$ and the real $(\mu \times \nu)$-matrix W are fixed and the $(\mu \times n)$-matrix $A(\omega)$ and $b(\omega) \in \mathbb{R}^\mu$ are random on the given probability space $(\Omega, \mathscr{F}, P)$. To assure complete recourse and the existence of the expectation $EQ(x, \omega)$, we assume

A1) $\{y \mid Wy = z, y \geq 0\} \neq \varnothing \quad \forall z \in \mathbb{R}^\mu$,

A2) $\{u \mid W'u \leq q\} \neq \varnothing$,

A3) the existence of $EA(\omega)$ and $Eb(\omega)$.

For simplicity we make a further assumption, which is not very restrictive in applications:

A4) $\bar{X} = \{x \mid Dx = d, x \geq 0\} \neq \varnothing$ and bounded.

The practical meaning of problems with complete recourse is this: taking a decision $x \in \bar{X}$ before knowing the realization $\omega \in \Omega$ may yield a violation of the constraints $b(\omega) - A(\omega)x = 0$; ω being observed, this violation can always (according to A1)) be compensated by the second stage program (2) with finite penalty costs $Q(x, \omega)$ (according to A2)). The objective in (1) is to take $x \in \bar{X}$ in such a way that the total expected costs (which exist according to A3)) be minimized. Examples of such problems appear in energy, production, transportation planning, and others.

[1] The prime at vectors and matrices means transposition.

where often part of the productivities, available resources, demands etc. are to be treated as random instead of deterministic, as they are in the usual linear programming approach.

Under our assumptions the following statements are well known [4]:

Proposition 1: $\varphi(x) = EQ(x, \omega)$ is convex and Lipschitz continuous. Furthermore, if the probability distribution of $(b(\omega), A(\omega))$ is given by a density function, $\varphi(x)$ has a continuous gradient.

Proposition 2: For a finite discrete probability distribution $P(\omega_i) = p_i > 0$, $i = 1, \ldots, r$, $\sum_{i=1}^{r} p_i = 1$, solving (1) coincides with solving the linear program min $[c'x + \sum_{i=1}^{r} p_i q' y^{(i)}]$

$$\begin{aligned} &\text{s.t. } Dx = d \\ &A(\omega_i)x + Wy^{(i)} = b(\omega_i), \quad i = 1, \ldots, r \\ &x \geq 0, \qquad y^{(i)} \geq 0, \quad i = 1, \ldots, r. \end{aligned} \tag{3}$$

Proposition 1 seems to suggest the solution of (1) by means of convex optimization techniques. However, this attempt fails, except for very special problems, since the evaluation of $\varphi(x)$, and moreover of its gradient, is in general a very complicated task. Proposition 2 suggests to approximate a given continuous probability distribution by an appropriate discrete one and to solve the corresponding linear program (3) instead of the nonlinear program (1). If we construct the discrete distribution by defining a finite disjoint partition $\{\mathfrak{A}_i\}$ of Ω, $\mathfrak{A}_i \in \mathcal{F}$, $i = 1, \ldots, r$, choosing $\omega_i \in \mathfrak{A}_i$, $p_i = P(\mathfrak{A}_i)$ and letting $(\tilde{A}, \tilde{b})$ be constant on $\mathfrak{A}_i$, we get the expected penalty costs $\tilde{\varphi}(x)$ instead of $\varphi(x)$, and we know from [5]

Proposition 3: There is a constant γ such that

$$|\varphi(x) - \tilde{\varphi}(x)| \leq \gamma[\|b - \tilde{b}\|_1 + \|x\| \cdot \|A - \tilde{A}\|_1], \tag{4}$$

where $\|\cdot\cdot\|$ is the Euclidean norm and $\|\cdot\cdot\|_1$ means the L_1-norm for vector valued functions on Ω.

Hence the above-mentioned suggestion in connection with proposition 2 seems to be justified in principal. However, we then are involved in large-scale linear programming, and therefore we should try to take advantage either of the special structure of (3) or of particular properties of (1). In the first case we are handling implicitly due to proposition 3 an overall approximation of $\varphi(x)$, whereas in the second case we only try to approximate the minimum value of $c'x + \varphi(x)$ on $\bar{X}$.

II. Minimization of the Overall Approximation of the Objective Function

In this approach we choose a discrete distribution of $(\tilde{A}, \tilde{b})$ such that the error estimate (4) becomes sufficiently small. For this discrete distribution we set up problem (3), which for applications may become very large; if for example $\mu = 50$ and we have only 4 independent random coefficients each of them approximated by not more than 5 realizations, which does in general not yield high accuracy, we get at least 625 blocks of 50 rows, i.e. 31 250 constraints in (3).

In [8], the author proposed (for a special stochastic program) to solve the dual of (3) by a modified revised simplex method applying a so-called basis reduction technique in each pivot step [1], [2]. It was supposed that this method was faster than the unmodified simplex method. We want to determine the reduction in the number of essential operations (i.e. multiplications and divisions) that results from this technique applied to our problem (3).

The basis reduction can be shortly described, for an arbitrary linear program, as follows: let

$$\hat{B} = \begin{pmatrix} B & Y \\ L & Z \end{pmatrix} \tag{5}$$

be a feasible basis found after some step during the revised simplex method. Hence $\hat{B}$ is nonsingular, and B is also supposed to be regular. Let

$$B^{-1}$$
$$X = B^{-1}Y$$
$$(LX - Z)^{-1}$$

be given. To make the next pivot step, we have to carry through the following operations:

a) Pricing Out

Solve

$$\pi\hat{B} = \beta' \tag{6}$$

where β consists of the basic components of the objective's gradient.

Now (6) is equivalent to

$$\left.\begin{aligned} \pi_2(LX - Z) &= \beta_1' X - \beta_2' \\ \pi_1 B &= \beta_1' - \pi_2 L \end{aligned}\right\}, \tag{7}$$

$\beta' = (\beta_1', \beta_2')$, $\pi = (\pi_1, \pi_2)$, which is determined by matrix multiplication.

b) Optimality Test

For all nonbasic columns

$$D_j = \begin{pmatrix} D_j^{(1)} \\ D_j^{(2)} \end{pmatrix}$$

we must check, whether

$$d_j - \pi D_j = d_j - \pi_1 D_j^{(1)} - \pi_2 D_j^{(2)} \geq 0 \tag{8}$$

(d_j nonbasic component of the objective's gradient). Let (8) be violated for D_p. Hence D_p may enter the basis.

c) Updating of the Pivot Column

We need

$$\psi = \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} = \mathcal{B}^{-1} D_p, \tag{9}$$

which also can be determined by matrix multiplication as the solution of

$$\left.\begin{aligned} &B\psi_0 = D_p^{(1)} \\ &(LX - Z)\psi_2 = L\psi_0 - D_p^{(2)} \\ &\psi_1 = \psi_0 - X\psi_2 \end{aligned}\right\}. \tag{10}$$

Suppose that $\psi \not\leq 0$, because otherwise the linear program we try to solve would not have a finite solution.

d) Determination of the Pivot Row

We need the vector of basic variables

$$\begin{aligned} &\tilde{x} = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = \mathcal{B}^{-1} b, \\ &b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \end{aligned} \tag{11}$$

the right-hand side of the linear program, which according to (9), (10) is found by solving

$$\left.\begin{aligned} &B\tilde{x}_0 = b_1 \\ &(LX - Z)\tilde{x}_2 = L\tilde{x}_0 - b_2 \\ &\tilde{x}_1 = \tilde{x}_0 - X\tilde{x}_2 \end{aligned}\right\}. \tag{12}$$

Then the leaving variable $\tilde{x}_{\cdot u}$ is given by

$$\frac{\tilde{x}_{\cdot u}}{\psi_{\cdot u}} = \min \left\{ \frac{\tilde{x}_{iu}}{\psi_{iu}} \,\middle|\, i = 1, 2; \quad \psi_{iu} > 0 \right\}. \tag{13}$$

e) Pivoting

Case 1: The leaving column $\hat{B}_\mu$ is part of

$$\begin{pmatrix} Y \\ Z \end{pmatrix}, \qquad \hat{B}_\mu = \begin{pmatrix} Y_\mu \\ Z_\mu \end{pmatrix},$$

hence B and L are not altered. Y_μ is replaced by $D_\rho^{(1)}$ and X_μ by

$$\hat{X}_\mu = B^{-1}D_\rho^{(1)}. \tag{14}$$

In $LX - Z$ only the column $LX_\mu - Z_\mu$ is replaced by

$$\varphi = L\hat{X}_\mu - \hat{Z}_\mu, \qquad \hat{Z}_\mu = D_\rho^{(2)},$$

i.e. we get the new matrix

$$(LX - Z)^{-1}_{\text{new}} \tag{15}$$

by pivoting with the updated pivot column $(LX - Z)^{-1}\varphi$.

Case 2: The leaving column $\hat{B}_\mu$ is part of

$$\begin{pmatrix} B \\ L \end{pmatrix}, \qquad \hat{B}_\mu = \begin{pmatrix} B_\mu \\ L_\mu \end{pmatrix},$$

and

a) $(B^{-1}D_\rho^{(1)})_\mu = \psi_{0\mu} \neq 0$ (see (10)).

Replace B_μ by $D_\rho^{(1)}$, yielding B_{new} again regular,

L_μ by $D_\rho^{(2)}$

and compute

$$B^{-1}_{\text{new}} \text{ by pivoting with } \psi_0 = B^{-1}D_\rho^{(1)} \quad \text{(see (10))}, \tag{16}$$

$$X_{\text{new}} = B^{-1}_{\text{new}}Y \quad (Y \text{ unaltered}) \tag{17}$$

and

$$(LX - Z)^{-1}_{\text{new}}. \tag{18}$$

In this case $(LX - Z)^{-1}_{\text{new}}$ cannot be determined by a single pivot step, since in general by (17) several columns of X (and LX) are changed.

b) $(B^{-1}D_\rho^{(1)})_\mu = \psi_{0\mu} = 0.$

Then the regularity of $\hat{B}$ implies the existence of a column X_j of X such that $X_{\mu j} \neq 0$. Exchange

$$\begin{pmatrix} B_\mu \\ L_\mu \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} Y_j \\ Z_j \end{pmatrix}$$

and compute

$$B_{\text{new}}^{-1} \text{ by pivoting with } B^{-1}Y_j = X_j \tag{19}$$

$$X_{j\,\text{new}} = B_{\text{new}}^{-1}D_p^{(1)} \tag{20}$$

$$Z_{j\,\text{new}} = D_p^{(2)} \tag{21}$$

$$X_{i\,\text{new}} = B_{\text{new}}^{-1}Y_i, \quad i \neq j, \tag{22}$$

$$(LX - Z_{\text{new}}^{-1}). \tag{23}$$

The manipulations, especially in case 2) b), guarantee that the form

$$\begin{pmatrix} B & Y \\ L & Z \end{pmatrix}$$

is maintained for all feasible bases generated by this variant of the simplex method, with B always of the same size and regular. This is advantageous if B is large and of special structure, e.g. blockdiagonal as it will be in our problem. Then the main savings of operations result from (7), (10), (12) and (16), if we observe the blockdiagonality.

To apply this procedure for our problem we reformulate the dual of (3)

$$\left.\begin{array}{l} \max \left[\sum\limits_{i=1}^{r} b'(\omega_i)v^{(i)} + d'u\right] \\ \text{s.t. } W'v^{(i)} \leq p_i q, \quad i = 1, \ldots, r \\ \sum\limits_{i=1}^{r} A'(\omega_i)v^{(i)} + D'u \leq c \end{array}\right\} \tag{24}$$

according to a proposal in [3]. By assumption A1) the columns of the recourse matrix W may always be rearranged such that $W = (W_{\text{I}}, W_{\text{II}})$, where W_{I} is nonsingular. Then

$$W'v^{(i)} \leq p_i q \tag{25}$$

is equivalent to (set $v^{(i)} = (W_{\text{I}}')^{-1}(p_i q_{\text{I}} - s^{(i)})$)

$$W_{\text{II}}'(W_{\text{I}}')^{-1}(p_i q_{\text{I}} - s^{(i)}) \leq p_i q_{\text{II}}, \quad s^{(i)} \geq 0, \tag{26}$$

where $q' = (q_{\text{I}}', q_{\text{II}}')$ corresponds to $W = (W_{\text{I}}, W_{\text{II}})$.

By this substitution and introducing slack variables and replacing u by nonnegative variables, (24) becomes

$$\left.\begin{array}{l} \max \sum\limits_{i=1}^{r+1} \tilde{b}_i' z^{(i)} + \gamma \\ \text{s.t. } \tilde{W}z^{(i)} = \tilde{q}_i, \quad i = 1, \ldots, r \\ \sum\limits_{i=1}^{r+1} \tilde{A}_i z^{(i)} = \tilde{q}_{r+1} \\ z^{(i)} \geq 0, \quad i = 1, \ldots, r, \end{array}\right\} \tag{27}$$

where

$$\begin{aligned}
\tilde{W} &= (-W'_{\mathrm{II}}(W'_{\mathrm{I}})^{-1}, I) \\
\tilde{q}_i &= p_i q_{\mathrm{II}} - W'_{\mathrm{II}}(W'_{\mathrm{I}})^{-1} p_i q_{\mathrm{I}}, \quad i = 1, \dots, r \\
\tilde{q}_{r+1} &= c - \sum_{i=1}^{r} A'(\omega_i)(W'_{\mathrm{I}})^{-1} p_i q_{\mathrm{I}} \\
\tilde{A}_i &= (-A'(\omega_i)(W'_{\mathrm{I}})^{-1}, 0), \quad i = 1, \dots, r \\
\tilde{A}_{r+1} &= (D', -D', I) \\
\tilde{b}'_i &= (-b'(\omega_i)(W'_{\mathrm{I}})^{-1}, 0'), \quad i = 1, \dots, r \\
\tilde{b}'_{r+1} &= (d', -d', 0') \\
\gamma &= \sum_{i=1}^{r} b'(\omega_i)(W'_{\mathrm{I}})^{-1} p_i q_{\mathrm{I}}.
\end{aligned} \tag{28}$$

By this reformulation the number of rows, compared to (24), is reduced by $r \cdot \mu$ which is a large number, as stated in the beginning.

Now obviously every feasible basis of (27) may be, by rearranging columns, brought to a form (5) with blockdiagonal B:

$$B = \begin{pmatrix} B_{(1)} & & 0 \\ & \ddots & \\ 0 & & B_{(r)} \end{pmatrix},$$

where the $B_{(i)}$ are $[(\nu - \mu) \times (\nu - \mu)]$-matrices.

Going through the procedure described above, it can easily be checked that the number N of essential operations per simplex iteration amounts

$$N \sim \mathcal{O}(r) \tag{29}$$

or more precisely

$$N = r[\nu^2 + \nu \cdot n - \mu \cdot \nu + \nu - \mu] + \delta \tag{30}$$

where δ does not depend on r.

If we had not modified the revised simplex method in the above-mentioned way, we would have had to pivot on inverses of feasible bases

$$\hat{B} = \begin{pmatrix} B & Y \\ L & Z \end{pmatrix}$$

which obviously are

$$\hat{B}^{-1} = \begin{pmatrix} B^{-1} - X(LX - Z)^{-1}LB^{-1} & X(LX - Z)^{-1} \\ (LX - Z)^{-1}LB^{-1} & -(LX - Z)^{-1} \end{pmatrix}$$

and which in general do not have a special structure even if B is blockdiagonal. Hence the number of essential operations per simplex iteration would have been

$$N \sim \mathcal{O}([r(\nu - \mu) + n]^2), \tag{31}$$

which is considerably greater than (30) for large r.

Nevertheless (30) also indicates the range of applicability of this approach. If for example we assume as in the beginning $\mu = 50$, $r = 625$ and $\nu = n = 100$ and if our computer needs 20 microseconds per multiplication, one simplex iteration will take about three minutes.

The approximating program (3) becomes so large because we are handling an equally good approximation of the whole objective function $c'x + \varphi(x)$ on $\bar{X}$ with an a priori error estimate according to (4), instead of attempting only to approximate the minimum of $c'x + \varphi(x)$ on $\bar{X}$, which we are really interested in. The basic ideas for an approach of that type are described below. There the probabilistic nature of our problem is used to some extent.

III. Approximating the Minimum

Let ξ be a random vector on $(\Omega, \mathcal{F}, P)$ with existing expectation. Then we know from probability theory [7] that for any σ-algebra $\mathcal{G} \subset \mathcal{F}$ there is an almost surely uniquely determined $\mathcal{G}$-measurable function (vector valued) $E_{\mathcal{G}}(\xi)$, called the conditional expectation, such that

$$\int_{\mathfrak{A}} E_{\mathcal{G}}(\xi)\, dP = \int_{\mathfrak{A}} \xi\, dP \quad \forall \mathfrak{A} \in \mathcal{G}. \tag{32}$$

Furthermore if $\mathcal{H}$ is a σ-algebra such that $\mathcal{H} \subset \mathcal{G} \subset \mathcal{F}$, then almost surely

$$E_{\mathcal{G}}(E_{\mathcal{G}}(\xi)) = E_{\mathcal{H}}(\xi). \tag{33}$$

If we define

$$\rho(b(\omega) - A(\omega)x) = Q(x, \omega) \quad \text{(see (2))}, \tag{34}$$

it is well known [4], that there are finitely many vectors g_k, $k = 1, \ldots, t$, such that

$$\rho(b(\omega) - A(\omega)x) = \max_k g_k'(b(\omega) - A(\omega)x). \tag{35}$$

Now we can easily prove

Proposition 4: Let $\mathcal{G}$ be the σ-algebra generated by a finite disjoint partition $\{\mathfrak{A}_i\}$ of Ω, $\mathfrak{A}_i \in \mathcal{F}$, hence $\mathcal{G} \subset \mathcal{F}$. Then

$$\int_{\Omega} \rho(E_{\mathcal{G}}(b(\omega) - A(\omega)x))\, dP \leq \int_{\Omega} \rho(b(\omega) - A(\omega)x)\, dP = \varphi(x). \tag{36}$$

Proof:

$$\begin{aligned}
&\int_\Omega \rho(E_{\mathscr{G}}(b(\omega) - A(\omega)x))\,dP = \sum_i \int_{\mathfrak{A}_i} \rho(E_{\mathscr{G}}(b(\omega) - A(\omega)x))\,dP \\
&\text{(35):} \qquad\qquad\qquad\qquad\quad = \sum_i \int_{\mathfrak{A}_i} g'_{k_i}(E_{\mathfrak{A}_i}(b(\omega) - A(\omega)x)\,dP \\
&\text{(32):} \qquad\qquad\qquad\qquad\quad = \sum_i \int_{\mathfrak{A}_i} g'_{k_i}(b(\omega) - A(\omega)x)\,dP \\
&\qquad\qquad\qquad\qquad\qquad\quad\;\; \le \sum_i \int_{\mathfrak{A}_i} \max\nolimits_k g'_k(b(\omega) - A(\omega)x)\,dP \\
&\qquad\qquad\qquad\qquad\qquad\quad\;\; = \varphi(x).
\end{aligned}$$

■

As a corollary we get immediately

Proposition 5: Let $\mathscr{G}$ be generated by a finite partition $\{\mathfrak{A}_i\}$ of Ω and $\mathscr{H}$ be generated by a finite partition $\{\vartheta_j\}$ of Ω such that $\mathscr{H} \subset \mathscr{G} \subset \mathscr{F}$. Then

$$\begin{aligned}
\int_\Omega \rho(E_{\mathscr{H}}(b(\omega) - A(\omega)x))\,dP &\le \int_\Omega \rho(E_{\mathscr{G}}(b(\omega) - A(\omega)x))\,dP \\
&\le \int_\Omega \rho(b(\omega) - A(\omega)x)\,dP = \varphi(x).
\end{aligned} \tag{37}$$

Proof: The second inequality was already proved, and the first one follows by combining (33) and proposition 4. ■

These propositions generalize Madansky's inequalities [6], which were proved for expectations instead of conditional expectations, as follows:

Proposition 6: Under the assumptions of proposition 4 define $\tilde{\varphi}(x) = \int_\Omega \rho(E_{\mathscr{G}}(b(\omega) - A(\omega)x))\,dP$. Assume that $\hat{x}$ is a solution of min $\{c'x + \tilde{\varphi}(x) \mid x \in \bar{X}\}$. Then

$$c'\hat{x} + \tilde{\varphi}(\hat{x}) \le \min_{x \in X}\,[c'x + \varphi(x)] \le c'\hat{x} + \varphi(\hat{x}). \tag{38}$$

These statements suggest that we proceed as follows: Start with the trivial partition $\{\Omega\}$ of Ω, i.e. with the expectations $E(b(\omega))$, $E(A(\omega))$ and solve the resulting program (3).

In general: having solved (3) for a partition $\{\mathfrak{A}_i\}$ (obviously with $P(\mathfrak{A}_i) > 0 \forall i$) getting the solution $\hat{x}$, subdivide one or several sets $\mathfrak{A}_{i_k}$ into disjoint subsets, compute the conditional expectations of $b(\omega)$, $A(\omega)$ with respect to the subsets and solve the updated program (3). Proceeding along these lines yields, according to (37), a monotonically increasing sequence of optimal values of (3), approximating the optimal value of (1). From the proof of (36) we also can conclude which $\mathfrak{A}_{i_k}$ should be chosen for subdivision to yield a strictly positive increase of $\tilde{\varphi}(\hat{x})$ at least at $\hat{x}$: Choose $\mathfrak{A}_{i_k}$ such that the optimal feasible basis out of W to compensate

$E_{\mathfrak{A}_{\iota_k}}(b(\omega) - A(\omega)\hat{x})$ does not remain feasible with positive probability for $b(\omega) - A(\omega)\hat{x}$ on $\mathfrak{A}_{\iota_k}$.

According to the numerical examples computed so far, procedures of this type have several advantages. On the one hand we start with linear programs of reasonable size, in the beginning the constraint set is just

$$\begin{aligned} &Dx = d \\ &\bar{A}x + Wy = \bar{b}, \qquad \bar{A} = EA(\omega), \qquad \bar{b} = Eb(\omega), \\ &x \geq 0, \qquad y \geq 0, \end{aligned}$$

and subdividing $\mathfrak{A}_\iota$ into $\mathfrak{A}_{\iota_1} \cup \mathfrak{A}_{\iota_2}$ just means replacing (let A_ι be $E_{\mathfrak{A}_\iota}A(\omega)$ etc.)

$$A_\iota\, x + Wy^{(\iota)} = b_\iota$$

by

$$\begin{aligned} &A_{\iota_1}x + Wy^{(\iota_1)} = b_{\iota_1} \\ &A_{\iota_2}x + Wy^{(\iota_2)} = b_{\iota_2} \end{aligned}$$

which can be managed within the simplex algorithm by simple updating techniques.

On the other hand proper choice of the sets $\mathfrak{A}_{\iota_k}$ to be subdivided yielded a rather fast increase towards the optimal value of (1).

The disadvantage so far is that in general the a posteriori error estimate (38) is only of theoretical value because of the complicated evaluation of $\varphi(x)$.

References

[1] M. D. Bakes, *Solution of Special Linear Programming Problems with Additional Constraints*, Operational Res. Qu. *17*, 425–445 (1966).

[2] K. Beer, *Lösung grosser linearer Optimierungsaufgaben*, Dt. Verlag der Wiss., Berlin (1977).

[3] E. Blum, Der Rechenaufwand im Strazicky-Verfahren, Manuskript, Universität Zürich (1977).

[4] P. Kall, *Stochastic Linear Programming*, Springer, Berlin–Heidelberg–New York (1976).

[5] P. Kall, *Approximations to Stochastic Programs with Complete Fixed Recourse*, Numer. Math. *22*, 333–339 (1974).

[6] A. Madansky, *Inequalities for Stochastic Linear Programming Problems*, Man. Sci. *6*, 197–204 (1960).

[7] H. Richter, *Wahrscheinlichkeitstheorie*, 2. Aufl, Springer, Berlin–Heidelberg–New York (1966).

[8] B. Strazicky, *On an Algorithm for Solution of the Two-Stage Stochastic Programming Problem*, Methods of OR *XIX*, 142–156 (1974).

THE SIMPLEX METHOD FOR DYNAMIC LINEAR PROGRAMS*

A. Propoi and V. Krivonozhko

Institute for Systems Studies
USSR Academy of Sciences
Moscow

There are two major approaches in the finite-step methods of structured linear programming: decomposition methods, which are based on the Dantzig–Wolfe decomposition principle, and basis factorization methods, which may be viewed as special instances of the simplex method.

In this paper, the second approach is used for one of the most important classes of structured linear programming — dynamic linear programming (DLP).

The paper presents a finite-step method for DLP — the dynamic simplex method. This is a natural and straightforward extension of one of the most effective static LP methods — the simplex method — for DLP. A new concept — a set of local bases (for each time step) — is introduced, thus enabling considerable reduction in the computer core memory requirements and CPU time.

The paper is in two parts. Part I, "the dynamic simplex method: general approach" and Part II, "a basis factorization approach" give a description of the dynamic simplex method and its extensions.

In Part I construction of a set of local bases and their relation to the conventional "global" basis in LP are given. A special control variation and the corresponding objective function variation as applied to this set of local bases are described. This part is written in a language more familiar to control theory specialists.

Part II describes the separate procedures of the dynamic simplex method: primal solution, dual solution, pricing, updating, and the general scheme of the algorithm. The connection between the method and the basis factorization approach is also shown. A numerical example and a theoretical evaluation of the algorithms reveal the efficacy of the approach. The extensions of the method (dual and primal-dual versions of the algorithms, application to DLP problems with time lags) are briefly discussed in the final part of the paper. This part is closer to LP specialists.

More theoretical aspects of the method are treated in IIASA Research Report 78-14, of which this paper forms the last two parts.

*The authors would like to express their acknowledgment to George B. Dantzig for his long-standing interest in and support for this work. Many thanks also to Etienne Loute for his valuable comments and suggestions.

I. THE DYNAMIC SIMPLEX METHOD: GENERAL APPROACH

1. Introduction

Methods for solving general linear programming (LP) are now well developed and have resulted in an extensive field of applications [1,2]. Dynamic linear programming (DLP) is a special class of linear programs for planning and control of complex systems over time [3-6]. DLP applications tend to be too large to be solved by general LP methods. These applications have been hampered by lack of universal DLP computer codes. The few DLP problems that are solved are limited in size. They are solved by treating them as static problems and using for their solution standard LP codes (see, for example, [4,6]).

As DLP problems are principally large-scale, this "static" approach is limited in its possibilities, and the development of efficient algorithms specially oriented to dynamic LP problems continues to be needed. In recent years, methods for DLP have been proposed which make it possible to take into account the specific features of dynamic problems (e.g. [7-9]).* But extension of the most effective finite-step method -- the simplex method for solving LP -- to the dynamic case yet has to be implemented although there have been a number of proposals by Dantzig and others.

The dynamic simplex method as presented here was first suggested in [10,11]. In this approach, the main concept of the static simplex method -- the basis -- is replaced by a set of local bases, introduced for the whole planning period. It allows a significant saving in the amount of computation and computer core. It permits the development of a set of finite-step DLP methods (primal, dual and primal-dual) which are the direct analogues of the corresponding static finite-step methods.

This paper consists of two parts: the first part describes the proposed approach; the second part presents the separate

*See also references in [3].

procedures and the general scheme of the algorithm as well as the connection with the basis factorization approach.

Consider the DLP problem in the following form.

Problem 1.1. Find a control

$$u = \{u(0),\ldots,u(T-1)\}$$

and a trajectory

$$x = \{x(0),\ldots,x(T)\} \quad ,$$

satisfying the state equation

$$x(t+1) = A(t)x(t) + B(t)u(t) \quad (t = 0,1,\ldots,T-1) \qquad (1.1)$$

with initial condition

$$x(0) = x^0 \qquad (1.2)$$

and constraints

$$G(t)x(t) + D(t)u(t) = f(t) \qquad (1.3)$$

$$u(t) \geq 0 \quad \geq \quad (t = 0,1,\ldots,T-1) \qquad (1.4)$$

which maximize the objective function

$$J_1(u) = a(T)x(T) \quad . \qquad (1.5)$$

Here the vector $x(t) = \{x_1(t),\ldots,x_n(t)\}$ defines the state of the system at step t in the state space E^n, which is assumed to be the n-dimension euclidean space; the vector $u(t) = \{u_1(t), \ldots,u_r(t)\} \in E^r$ (r-dimension euclidean space) specifies the controlling action at step t; vectors x^0, $f(t)$ and the matrices $A(t)$, $B(t)$, $G(t)$, $D(t)$, respectively are of dimensions $(n \times 1)$, $(m \times 1)$ and $(n \times n)$, $(n \times r)$, $(m \times n)$, $(m \times r)$, and are assumed to be given. In vector products, the right vector is a column, the left vector is a row.

There are a number of modifications of Problems 1.1 which can either be reduced to this problem [12,13] or the results stated below may be used directly for their solution. For example, constraints on the state and control variables can be separate; state variables may be nonnegative; state equations include time lags; the objective function depends on the whole sequences $\{u(t)\}$ and/or $\{x(t)\}$, etc. [3,12]).

Along with the primal Problem 1.1, use will be made of its dual [12].

Problem 1.2. Find a dual control

$$\lambda = \{\lambda(T-1),\dots,\lambda(0)\}$$

and a dual (conjugate) trajectory

$$p = \{p(T),\dots,p(0)\} \quad ,$$

satisfying the costate (conjugate) equation

$$p(t) = p(t+1)A(t) - \lambda(t)G(t) \quad (t=T-1,\dots,1,0) \tag{1.6}$$

with boundary condition

$$p(T) = a(T) \tag{1.7}$$

and constraints

$$p(t+1)B(t) - \lambda(t)D(t) \leq 0 \quad (t=T-1,\dots,1,0) \tag{1.8}$$

which minimize the objective function

$$J_2(\lambda) = p(0)x^0 + \sum_{t=0}^{T-1} \lambda(t)f(t) \quad . \tag{1.9}$$

Definition 1.1. *A feasible control* of the DLP Problem 1.1 is a vector sequence $u = \{u(0),\dots,u(T-1)\}$ which satisfies with some trajectory $x = \{x(0),\dots,x(T)\}$ conditions (1.1) to (1.4).

An *optimal control* of Problem 1.1 is a feasible control u*, which maximizes (1.5) subject to (1.1) - (1.4).

Feasible dual control λ *and optimal dual control* λ^* to the dual Problem 1.2 are defined in a similar way.

Let $U = E^{rT}$; $u = \{u(0),\ldots,u(T-1)\} \in U$ be the control space of Problem 1.1. In the control space U Problem 1.1 can be rewritten as follows [13].

One can obtain from the state equation (1.1), that

$$x(t) = \Psi(t,0)x(0) + \sum_{\tau=0}^{t-1} \Psi(t,\tau+1)B(\tau)u(\tau) \qquad (1.10)$$

where

$$\Psi(t,\tau) = A(t-1) \cdot A(t-2) \cdots A(\tau) \qquad (0 \leq \tau \leq t-1) \quad ,$$

$$\Psi(t,t) = I \quad .$$

I is the identity.

By substituting (1.10) into (1.3) and taking into account (1.2), we obtain the constraints on controls u, given in explicit form:

$$\sum_{\tau=0}^{t} W(t,\tau)u(\tau) = h(t) \quad , \qquad (1.11)$$

$$u(t) \geq 0 \quad , \quad (t = 0,\ldots,T-1) \quad .$$

Here

$$W(t,\tau) = G(t)\Psi(t,\tau+1)B(\tau) \qquad (t > \tau) \quad ,$$

$$W(t,t) = D(t) \quad , \qquad h(t) = f(t) - G(t)\Psi(t,0)x^0 \quad .$$

The matrices $W(t,\tau)$ are of dimension $(m \times r)$ and vectors h(t) are of dimension $(m \times 1)$.

The objective function (1.5) will be rewritten, respectively, in the form

$$J_1(u) = \sum_{t=0}^{T-1} c(t)u(t) + q(0)x^0 \quad , \tag{1.12}$$

where

$$c^T(t) = q(t+1)B(t) \quad .$$

Here vectors $q(t)$ are generated recursively by

$$q(t) = q(t+1)A(t) \qquad (t = T-1,\dots,0)$$

$$q(T) = a(T) \quad .$$

Denoting the constraint matrix of (1.11) by W (dimension is $mT \times rT$), we can reformulate Problem 1.1 in the following equivalent form.

Problem 1.1a. Find a control u, satisfying the constraints

$$Wu = h \qquad u \geq 0 \quad ,$$

which maximizes the objective function

$$\tilde{J}_1(u) = cu \quad .$$

Here $u = \{u(t)\}$; $h = \{\check{h}(t)\}$; $c = \{c(t)\}$ $(t = 0,1,\dots,T-1)$ and $\tilde{J}_1$ differs from J_1 by the constant $q(0)x^0$.

It is evident that the sets of optimal controls for Problem 1.1 and 1.1a are the same.

Now the general scheme of the simplex method as applied to Problems 1.1a will be described.

Let u be a feasible control; we shall define the index sets

$$I(u) = \{(i,t)\,|\,u_i(t) > 0 \,;\, i = 1,\dots,r \,;\, t = 0,\dots,T-1\}$$

$$\bar{I}(u) = \{(i,t)\,|\,u_i(t) = 0 \,;\, i = 1,\dots,r \,;\, t = 0,\dots,T-1\}$$

$$I = I(u) \cup \bar{I}(u) \quad .$$

Denote also the columns of matrix W by $w_i(t)$ $(i = 1,\ldots,r;$ $t = 0,1,\ldots,T-1;$ $w_i(t) \in E^{mT}$. In this case the constraints (1.11) can be rewritten as

$$\sum_{(i,t)\in I} w_i(t)u_i(t) = h \quad ; \qquad u_i(t) \geq 0 \quad .$$

Definition 1.2. A *basic feasible control* of Problem 1.1 is a feasible control u, for which vectors $w_i(t)$, $(i,t) \in I(u)$, are linearly independent.

A *nondegenerate basic feasible control* is a basic feasible control u, for which vectors $w_i(t)$, $(i,t) \in I(u)$, constitute a basis in E^{mT}.

The *basis of a basic feasible control* u is a system of mT linearly independent vectors $w_i(t)$, which contains all vectors $w_i(t)$, $i(t) \in I(u)$.

As usual without any loss in generality we can assume that Problem 1.1a (1.1) is feasible and that any basic feasible control is nondegenerate [1].

Denote by $I_B(u)$ the set of indices corresponding to the basic vectors $w_i(t)$; $I_N(u)$ is the set of indices corresponding to the remaining vectors $w_i(t)$ of matrix W. Let

$$u_B = \{u_i(t) | (i,t) \quad I_B(u)\} \quad ,$$

$$u_N = \{u_i(t) \quad (i,t) \quad I_N(u)\} \quad ,$$

and m(t) is the number of basic components of a basic control u at step t. Evidently

$$\sum_{t=0}^{T-1} m(t) = mT \quad .$$

Then, any basic feasible control may be represented as

$$u = \{u_B, u_N\}, \text{ with } u_B \geq 0 \quad , \qquad u_N = 0 \quad .$$

Denote by W_B the matrix with columns $w_i(t)$, $(i,t) \in I_B(u)$ (basic matrix). Then $u_B = W_B^{-1}h$.

Let $w_j(t_1), (j,t_1) \in I$, be an arbitrary column vector of W, then

$$w_j(t_1) = W_B v_j(t_1) \quad , \tag{1.13}$$

where vector $v_j(t_1) = \{v_{ij}(t_1,\tau)\}$, $(i = 1,\ldots,m\, , \tau = 0,\ldots,T-1)$ has dimension mT.

Define

$$z_j(t_1) = c_B v_j(t_1) \quad .$$

Thus, we can rewrite

$$\begin{aligned} c_j(t) &= q(t+1)b_j(t) \\ z_j(t) &= \sum_{\tau=0}^{T-1} q(\tau+1)B_B(\tau)v_j(t,\tau) \quad . \end{aligned} \tag{1.14}$$

Here $b_j(t)$ is a column of the matrix $B(t)$; the matrix $B_B(\tau)$ is generated by the basic columns $b_i(\tau)$, $(i,\tau) \in I_B(u)$ of the matrix $B(\tau)$; $(j,t) \in I$.

The direct application of the simplex method to Problem 1.1 (1.1a) gives the following basic operations:

1. The computation of the sign ş or $z_j(t) - c_j(t)$ for all $(j,t) \in I$, to determine whether an optimal control has been found; that is the case when $z_j(t) - c_j(t) \geq 0$ for all j and t. If yes, the algorithm terminates with a printout of the optimal solution. If not, then

2. the selection of the vector to be introduced into the basis, that is selection of a vector with a value of $z_j(t) - c_j(t) < 0$. Let the pair of indices associated with this vector be (j,t_1).

3. The selection of the vector to be removed from the basis. The pair of indices associated with this vector will be denoted by (ℓ, t_2). If (ℓ, t_2) cannot be found, the algorithm terminates with a printout of information of how to generate a class of feasible solutions such that $J_1(u) \to +\infty$. If not, then

4. the basis and basic feasible control is updated. The new basic feasible control $u^{(1)} = \{u_B^{(1)}, 0\}$ is defined by

$$\begin{aligned} u^{(1)}_{s_i}(\tau) &= u_{s_i}(\tau) - \theta_0 v_{s_{i_j}}(t_1,\tau) \qquad (s_i,\tau) \in I_B(u) \\ u^{(1)}_j(t_1) &= \theta_0 \qquad\qquad\qquad\qquad\qquad\qquad (1.15) \\ u^{(1)}_i(\tau) &= 0 \qquad (i,\tau) \neq (j,t_1); \qquad (i,\tau) \in I_N(u) \quad , \end{aligned}$$

where the outgoing pair of indices (ℓ, t_2) is given by the value θ_0 which is calculated from

$$(\ell, t_2) = \underset{\substack{v_{s_{i_j}}(t_1,\tau)>0 \\ (s_i,\tau)\in I(u)}}{\text{arg-min}} \frac{u_{s_i}(\tau)}{v_{s_{i_j}}(t_1,\tau)} \qquad (1.16)$$

and θ_0 by

$$\theta_0 = \frac{u_{s_\ell}(t_2)}{v_{\ell j}(t_1,t_2)} \quad .$$

The numbers $z_j(t)$ are usually computed from $z_j(t) = \lambda w_j(t)$, where $\lambda = \{\lambda_i(\tau),\ (i,\tau) \in I_B(u)\}$ are simplex multipliers for the basis W_B:

$$\lambda = c_B W_B^{-1} \quad . \qquad (1.17)$$

The general scheme considered above is in practice ineffective for the solution of Problem 1.1 (1.1a) when the dimension of the matrix W is large. Besides, the input data are usually given in the form of Problem 1.1 rather than in the form of Problem 1.1a and no exploitation has been made of its special structure. Therefore the simplex procedure directly designed for the solution of Problem 1.1 will be described.

2. Local Bases

The matrices $D(t)$ $(t = 0,\dots,T-1)$ of constraints (1.3) will be assumed to have the rank m. This assumption is not restrictive because one could always insert, if necessary, additional artificial columns, as in the static case, see [1].

Let us denote $\hat{f}(0) = f(0) - G(0)x^0$. Then constraints (1.3) can be rewritten as

$$D(0)u(0) = \hat{f}(0) \quad . \tag{2.1}$$

In accordance with our assumption we can choose m linearly independent column-vectors $d_i(0)$ of the matrix $D(0)$. Denote these columns by $D_0(0)$ and the rest of $D(0)$ by $D_1(0)$. Thus

$$D(0) = [D_0(0);D_1(0)] \quad .$$

As determinant $|D_0(0)| \neq 0$, the constraints (2.1) can be rewritten in the form

$$u_0(0) = D_0^{-1}(0)\hat{f}(0) - D_0^{-1}(0)D_1(0)u_1(0) \quad , \tag{2.2}$$

where components of the vector $u_0(0) \in E^m$ correspond to the matrix $D_0(0)$ and components of the vector $u_1(0) \in E^{r-m}$ correspond to the matrix $D_1(0)$.

The partition of the matrix $B(0)$ is carried out similarly to that of the partition of $D(0)$: $B(0) = [B_0(0);B_1(0)]$. Therefore

$$x(1) = A(0)x(0) + B_0(0)u_0(0) + B_1(0)u_1(0) \quad . \tag{2.3}$$

Substitution (2.2) into (2.3) yields

$$x(1) = x^*(1) + B^1(0)u_1(0) \quad , \tag{2.4}$$

where

$$B^1(0) = B_1(0) - B_0(0)D_0^{-1}(0)D_1(0) \quad ,$$
$$x^*(1) = A(0)x^0 + B_0(0)u_0^*(0) \quad ,$$
$$u_0^*(0) = D_0^{-1}(0)\hat{f}(0) \quad .$$

Now we consider a step t, $0 \leq t \leq T - 1$. Let

$$\hat{D}(t)\hat{u}(t) = \hat{f}(t) \tag{2.5}$$

where

$$\hat{D}(t) = [G(t)B^1(t-1);D(t)] \tag{2.6}$$
$$\hat{u}(t) = [\hat{u}_1(t-1);u(t)]^T \tag{2.7}$$
$$\hat{f}(t) = f(t) - G(t)x^*(t) \quad . \tag{2.8}$$

In (2.6) to (2.8), the matrix $B^1(t-1)$ and vectors $\hat{u}_1(t-1)$, $x^*(t)$ are defined from recurrent relations, which will be obtained below.

By construction, the matrix $\hat{D}(t)$ contains m linearly independent columns $\hat{d}_i(t)$. The matrix formed by these columns will be denoted as $\hat{D}_0(t)$: the matrix from the rest of the columns -- as $\hat{D}_1(t)$. Thus, (2.5) can be rewritten as

$$\hat{D}_0(t)\hat{u}_0(t) + \hat{D}_1(t)\hat{u}_1(t) = \hat{f}(t)$$
$$\hat{D}(t) = [\hat{D}_0(t);\hat{D}_1(t)] \quad .$$

Hence

$$\hat{u}_0(t) = \hat{D}_0^{-1}(t)\hat{f}(t) - \hat{D}_0^{-1}(t)\hat{D}_1(t)\hat{u}_1(t) \quad . \tag{2.9}$$

Or

$$\hat{u}_0(t) = \hat{u}_0^*(t) - \Phi(t)\hat{u}_1(t) \quad , \tag{2.10}$$

where

$$\hat{u}_0^*(t) = \hat{D}_0^{-1}(t)\hat{f}(t) \quad , \tag{2.11}$$

$$\Phi(t) = \hat{D}_0^{-1}(t)\hat{D}_1(t) \quad . \tag{2.12}$$

Let

$$x(t) = x^*(t) + B^1(t-1)\hat{u}_1(t-1) \quad , \tag{2.13}$$

where $x^*(t)$ and $B^1(t-1)$ will be defined later.

By substituting (2.14) into state equation (1.1), we obtain

$$x(t+1) = A(t)x^*(t) + \hat{B}(t)\hat{u}(t) \quad , \tag{2.14}$$

where

$$\hat{B}(t) = [A(t)B^1(t-1);B(t)] \quad , \tag{2.15}$$

the vector $\hat{u}(t)$ is defined by (2.7).

Considering the representation

$$\hat{B}(t) = [\hat{B}_0(t);\hat{B}_1(t)] \quad ,$$

$$\hat{u}(t) = [\hat{u}_0(t);\hat{u}_1(t)]^T$$

and substituting (2.10) into (2.14), we again obtain equations (2.13) for the next step $t+1$:

$$x(t+1) = x^*(t+1) + B^1(t)\hat{u}_1(t) \quad ,$$

where

$$x^*(t+1) = A(t)x^*(t) + \hat{B}_0(t)\hat{u}_0^*(t) \quad , \tag{2.16}$$

$$B^1(t) = \hat{B}_1(t) - \hat{B}_0(t)\Phi(t) \quad . \tag{2.17}$$

Initial conditions for (2.14), (2.5) are

$$x^*(0) = x(0); \qquad \hat{u}(0) = u(0) \quad ,$$
$$\hat{B}(0) = B(0) \quad , \quad \hat{D}(0) = D(0) \quad . \tag{2.18}$$

The specific of such a representation of Problem 1.1 is a recurrent determination of control $\hat{u}(t)$, that is, using (2.7) we obtain

$$\begin{aligned}\hat{u}(t) &= [\hat{u}_1(t-1),u(t)]^T \qquad (2.19)\\ &= [\hat{u}_2(t-2),u_1(t-1),u(t)]^T = \ldots = [u_t(0),u_{t-1}(1),\ldots,u_{t-i}(i),\\ &\quad \ldots,u_1(t-1),u(t)]^T\end{aligned}$$

where the vector $u_{t-i}(i)$ is formed from those components of the control u which are recomputed from a step i to the step t by virtue of the procedure which was described above. The relations (2.19) show that the vector $\hat{u}(t)$ may include components $u_i(\tau)$ from preceding steps $\tau = t-1,\ldots,1,0$.

Consider now the last step

$$\hat{D}_0(T-1)\hat{u}_0(T-1) + \hat{D}_1(T-1)\hat{u}_1(T-1) = \hat{f}(T-1)$$

where $\hat{D}_0(T-1)$ is a nonsingular matrix. Let

$$\hat{u}_1(T-1) = 0 \quad , \tag{2.20}$$

then

$$\hat{u}_0(T-1) = \hat{D}_0^{-1}(T-1)\hat{f}(T-1) \quad . \tag{2.21}$$

Determining the value of the vector $\hat{u}(T-1) = [\hat{u}_0(T-1),\hat{u}_1(T-1)]^T$ from (2.20), (2.21), one can compute the values of feasible control $\{u(t)\}$ for a given set of local bases $\{\hat{D}_0(t)\}(t = 0,1,\ldots,T-1)$. This procedure will be called Procedure 1.

Definition 2.1: The set of m linearly independent columns $\hat{d}_i(t)$ of the matrix $\hat{D}(t)$ is called the *local basis* at the step t $(t = 0,1,\ldots,T-1)$.

The set of all indices (i,t) associated with the components of local basis matrix $\hat{D}_0(t)$ (t = 0,...,T-1) will be denoted by $I_0(u)$, and its complement with respect to I will be denoted by $\bar{I}_0(u)$.

Theorem 2.1: Let a control u *be computed from Procedure 1 for a given set of local bases* $\{\hat{D}_0(t)\}$ *with boundary conditions*

$$\hat{u}_0(T-1) = \hat{D}^{-1}(T-1)\hat{f}(T-1)$$
$$\hat{u}_1(T-1) = 0$$

and let

$$u_i(t) \geq 0 \quad \text{for all} \quad (i,t) \in I_0(u) \quad .$$

Then u *is a basic feasible control and*

$$u = \{u_B, u_N\} \quad ,$$
$$u_B = \{u_i(t) \mid (i,t) \in I_0(u)\} \quad ,$$
$$u_N = \{u_i(t) \mid (i,t) \in \bar{I}_0(u)\} \quad .$$

Proof: Let W_0 be the matrix which is generated by the columns $w_i(t)$ of the constraint matrix W, associated with variables $\hat{u}_0(t)$, that is,

$$W_0 = \|w_i(t)\| \quad , \qquad (i,t) \in I_0(u) \quad .$$

By construction, W_0 is a square matrix of dimension of mT × mT.

For proof of the theorem, we shall need the following assertion.

Lemma 2.1: The matrix W_0 *is nonsingular if and only if the matrices* $\hat{D}_0(t)\,(t = 0,1,\ldots,T-1)$ *are nonsingular.*

Proof: *Sufficiency*. The procedure of computing $\{\hat{u}_0(t)\}$ described above is a block modification of the Gauss method [14] where pivot blocks are matrices $\hat{D}_0(t)$. The Gauss algorithm transforms the matrix W_0 to an upper block triangular matrix with $\hat{D}_0(t)$ on its diagonal:

$$W_0 = \begin{bmatrix} \hat{D}_0(0) & * & & & & \\ & \hat{D}_0(1) & * & & & 0 \\ & & \ddots & & & \\ & & & \hat{D}_0(t) & * & \\ & 0 & & & \ddots & \\ & & & & & \hat{D}_0(T-1) \end{bmatrix}$$

where nonzero elements of W_0 are denoted by $*$.

The Gauss algorithm does not change the rank of the original matrix [14]. In fact, the relation

$$\left| |W_0| \right| = \left| |D_0(0)| \ \dots \ |D_0(T-1)| \right| \tag{2.22}$$

holds, where $\left| |W_0| \right|$ is the absolute value of the determinant of a matrix W_0. The relation (2.22) implies that, if matrices $\hat{D}_0(t)$ $(t = 0, 1, \dots, T-1)$ are nonsingular, then the matrix W_0 is also nonsingular.

Necessity: Suppose that k iterations of the Gauss algorithm have been done and W_0^k is a matrix obtained after k iterations:

$$W_0^k = \begin{bmatrix} \hat{D}_0(0) & * & & & & \\ & \hat{D}_0(1) & * & & 0 & \\ & & \ddots & & & \\ & & & \hat{D}_0(k-1) & * & \\ & 0 & & & \ddots & \\ & & & & & \tilde{W}_0^k \end{bmatrix}$$

Here $\tilde{W}_0^k$ is a submatrix, generated by elements of W_0^k which are outside of pivot rows and columns of previous iterations. In this case, the relation (2.22) should be replaced by

$$\left| |W_0| \right| = \left| |\hat{D}_0(0)| \ \ldots \ |\hat{D}_0(k-1)| \ |\tilde{W}_0^k| \right| \quad .$$

The first block-row of $\tilde{W}_0^k$ is $[\hat{D}(k);0]$. Suppose that the matrix $\hat{D}(k)$ cannot generate any nonsingular square submatrix $\hat{D}_0(k)$ of dimension m. This implies that the rows of the matrix $\hat{D}(k)$ are linearly dependent and the matrix $\tilde{W}_0^k$ is singular with $|\tilde{W}_0^k| = 0$. Then $|W_0| = 0$, which contradicts the assumption of the lemma.

Thus, if the matrix W_0 is nonsingular then at each step of the Gauss algorithm a nonsingular matrix $\hat{D}_0(k)$ can be constructed. This completes the proof of the lemma.

The proof of the theorem now follows directly. By definition, matrices $\hat{D}_0(t)\,(t = 0,\ldots,T-1)$ are nonsingular, which implies that the matrix W_0 is also nonsingular and vectors $w_i(t)$, $(i,t) \in I_0(u)$, are linearly independent.

It follows from Procedure 1 that

$$u_i(t) = 0 \quad \text{for all} \quad (i,t) \in \bar{I}_0(u) \quad .$$

As $u_i(t) \geq 0$ for all $(i,t) \in I_0(u)$, then in accordance with definition 1.2 u is a basic feasible control. This completes the proof of the theorem.

The proof of Theorem 3.1 shows that Procedure 1 permits operations not with the inverse W_B^{-1} of dimension $mT \times mT$ but with T inverses $\hat{D}_0^{-1}(t)$ of dimension $m \times m$. Hence, Procedure 1 is basic to this approach. However, as will be seen further, it is not used explicitly.

In fact, as follows from the proof of the theorem, only basic submatrices of matrices $\hat{D}(t)$ should be handled in the algorithm. Besides, there is no necessity to compute local bases at each iteration, only the updating of some of the T local bases is needed.

3. Control Variation

In accordance with Theorem 2.1, the basis W_B is equivalent to the set of local bases $\{\hat{D}_{0B}(t)\}$. Therefore, our aim is to develop the simplex operations for solution of Problem 1.1 relative to the set of local bases $\{\hat{D}_{0B}(t)\}$.

For a given basic feasible control $u=\{u_B,u_N\}$, let us fix the pair of indices $(j,t_1)\in I$ such that the corresponding column $d_j(t_1)$ of the matrix $D(t_1)$ is not in the basis, that is, $(j,t_1) \in I_N(u)$.

We first consider the procedure for selection of the column $d_j(t_1)$ to be introduced into the basis, that is, into the set of local bases $\{\hat{D}_{0B}(t)\}$. In accordance with Section 2, the constraints (1.3) at step t can be written as

$$\hat{D}_{0B}(t)\hat{u}_{0B}(t) + \hat{D}_{1B}(t)\hat{u}_{1B}(t) = \hat{f}(t) \qquad (3.1)$$

where

$$[\hat{D}_{0B}(t);\hat{D}_{1B}(t)] = \hat{D}_B(t) \quad ,$$
$$[\hat{u}_{0B}(t);\hat{u}_{1B}(t)] = \hat{u}_B(t) \quad , \qquad \hat{u}_B(t) \geq 0 \quad .$$

Here the subscript B denotes submatrices and vectors associated with a given basis W_B or $\{\hat{D}_{0B}(t)\}$.

Let a vector $\hat{v}^*_{0B}(t_1)\in E^m$ define representation of the vector $d_j(t_1)$ in terms of column-vectors of the matrix $\hat{D}_{0B}(t_1)$, that is,

$$\hat{v}^*_{0B}(t_1) = \hat{D}^{-1}_{0B}(t_1)d_j(t_1) \quad . \qquad (3.2)$$

Taking into account (3.2), we can rewrite (3.1) as

$$\hat{D}_{0B}(t_1)\left[\hat{u}_{0B}(t_1)-\theta\hat{v}^*_{0B}(t_1)\right] + \hat{D}_{1B}(t_1)\hat{u}_{1B}(t_1) + \theta d_j(t_1) = \hat{f}(t_1) \qquad (3.3)$$

where θ is a real number.

It is evident that the equality (3.3) is true for any value of the parameter θ. It follows from (3.3) that a new control $u^{\theta}(t_1)$ is introduced at step t_1:

$$\hat{u}^{\theta}(t_1) = \left[\hat{u}^{\theta}_{0B}(t_1);\hat{u}^{\theta}_{1B}(t_1);\hat{u}^{\theta}_{N}(t_1)\right]^{T} ,$$

where

$$\begin{aligned}\hat{u}^{\theta}_{0B}(t_1) &= \hat{u}_{0B}(t_1) - \theta\hat{v}^{*}_{0B}(t_1)\\ u^{\theta}_{1B}(t_1) &= \hat{u}_{1B}(t_1)\\ \hat{u}^{\theta}_{N}(t_1) &= [0,\ldots,\theta,\ldots,0]^{T} .\end{aligned} \tag{3.4}$$

By substituting the control $\hat{u}^{\theta}(t_1)$ in state equation (2.14), we obtain

$$x^{\theta}(t_1+1) = x(t_1+1) - \theta y^{*}(t_1+1) , \tag{3.5}$$

where

$$\begin{aligned} x(t_1+1) &= x^{*}(t_1+1) + B^{1}_{B}(t_1)\hat{u}_{1B}(t_1) ,\\ y^{*}(t_1+1) &= \hat{B}_{0B}(t_1)\hat{v}^{*}_{0B}(t_1) - b_j(t_1) .\end{aligned} \tag{3.6}$$

Substituting (3.5) into formulation (2.5) of constraints (1.3), we see that they will be true if

$$\hat{D}_B(t_1+1)\hat{u}^{\theta}_{B}(t_1+1) - \theta G(t_1+1)y^{*}(t_1+1) = \hat{f}(t_1+1) . \tag{3.7}$$

Let us express the vector $-G(t_1+1)y^{*}(t_1+1)$ in terms of column vectors of the matrix $\hat{D}_{0B}(t_1+1)$:

$$\hat{v}^{*}_{0B}(t_1+1) = -\hat{D}^{-1}_{0B}(t_1+1)G(t_1+1)y^{*}(t_1+1) . \tag{3.8}$$

Considering (3.8), the equality (3.7) can be rewritten as

$$\begin{aligned}\hat{D}_{0B}(t_1+1)&\left[\hat{u}_{0B}(t_1+1) - \theta\hat{v}^{*}_{0B}(t_1+1)\right]\\ &+\hat{D}_{JB}(t_1+1)\hat{u}_{1B}(t_1+1) - \theta G(t_1+1)y^{*}(t_1+1)\\ &=\hat{f}(t_1+1) .\end{aligned}$$

We see that the introduction of the compensating term into the equality (3.7) is equivalent to the introduction of a new control $\hat{u}^{\theta}(t_1 + 1)$ at step $t_1 + 1$:

$$\hat{u}^{\theta}(t_1 + 1) = [\hat{u}^{\theta}_{0B}(t_1 + 1); \hat{u}^{\theta}_{1B}(t_1 + 1); \hat{u}^{\theta}_{N}(t_1 + 1)] \quad ,$$

where

$$\begin{aligned} \hat{u}^{\theta}_{0B}(t_1 + 1) &= \hat{u}_{0B}(t_1 + 1) - \theta\hat{v}^{*}_{0B}(t_1 + 1) \\ \hat{u}^{\theta}_{1B}(t_1 + 1) &= \hat{u}_{1B}(t_1 + 1) \\ \hat{u}^{\theta}_{N}(t_1 + 1) &= 0 \quad . \end{aligned} \tag{3.9}$$

Thus, the variation of the control (3.4) at step t_1, where vector $\hat{v}^{*}_{0B}(t_1)$ is defined by (3.2), induces a variation of control (3.9) at the next steps $\tau = t_1 + 1,\ t_1 + 2, \ldots, T - 2$ with

$$\hat{v}^{*}_{0B}(\tau) = -\hat{D}^{-1}_{0B}(\tau)G(\tau)y^{*}(\tau) \quad . \tag{3.10}$$

Vectors $y^{*}(\tau)$ are satisfied to the following difference equation:

$$y^{*}(\tau + 1) = A(\tau)y^{*}(\tau) + \hat{B}_{0B}(\tau)\hat{v}^{*}_{0B}(\tau) \tag{3.11}$$

where vectors $\hat{v}^{*}_{0B}(\tau)$ $(\tau = t_1 + 1, \ldots, T - 1)$ are defined from (3.10) and vector $\hat{v}^{*}_{0B}(t_1)$ is defined from (3.2).

Now we consider the last step:

$$\begin{aligned} \hat{D}_{B}(T - 1) \left[\hat{u}_{B}(T - 1) - \theta\hat{v}_{B}(T - 1)\right] - \theta G(T - 1)y^{*}(T - 1) \\ = \hat{f}(T - 1) \quad . \end{aligned} \tag{3.12}$$

As $u = \{u_B, 0\}$ is a basic feasible control, then by virtue of Theorem 2.1, the matrix $\hat{D}_{B}(T - 1)$ is nonsingular and

$$\hat{D}_{B}(T - 1) = \hat{D}_{0B}(T - 1) \quad .$$

Therefore (3.12) yields that

$$\hat{v}_B(T-1) = \hat{v}^*_{0B}(T-1) = -\hat{D}^{-1}_{0B}(T-1)G(T-1)\underline{y}^*(T-1) \quad .$$

By construction, the structure of vector $\hat{v}_B(T-1)$ is similar to the structure of vector $\hat{u}_B(T-1)$. Hence, define a vector:

$$\hat{v}_B(T-1) = \left[\hat{v}_{1B}(T-2), v_B(T-1)\right] \tag{3.13}$$

where vector $v_B(T-1)$ is associated with the variation of vector $u_B(T-1)$, vector $\hat{v}_{1B}(T-2)$ is associated with the variation of vector $\hat{u}_{1B}(T-2)$:

$$\hat{u}^{\theta}_{1B}(T-2) = \hat{u}_{1B}(T-2) - \theta\hat{v}_{1B}(T-2) \quad .$$

To satisfy the constraints at step $T-2$, the additional term $-\theta\hat{D}_{1B}(T-2)v_{1B}(T-2)$ must be compensated by the additional variation $\hat{v}^1_{0B}\ (T-2)$ of control $\hat{u}_{0B}(T-2)$:

$$\hat{u}^{\theta}_{0B}(T-2) = \hat{u}_{0B}(T-2) - \theta\left[\hat{v}^*_{0B}(T-2) - \hat{v}^1_{0B}(T-2)\right] \quad ,$$

where

$$\hat{v}^1_{0B}(T-2) = \hat{D}^{-1}_{0B}(T-2)\hat{D}_{1B}(T-2)\hat{v}_{1B}(T-2) = \phi_B(T-2)\hat{v}_{1B}(T-2) \quad .$$

Let $\hat{v}_{0B}(T-2) = \hat{v}^*_{0B}(T-2) - \hat{v}^1_{0B}(T-2)$. As in the case of (2.7), we can write

$$\begin{aligned} \hat{v}_B(T-2) &= \left[\hat{v}_{0B}(T-2), \hat{v}_{1B}(T-2)\right] \\ &= \left[\hat{v}_{1B}(T-3), v_B(T-2)\right] \quad . \end{aligned} \tag{3.14}$$

By induction, we find that in order to satisfy the constraints (1.2) for all θ and $\tau = 0,1,\ldots,T-1$, we must define

$$\hat{D}_B(T-1)\,[\hat{u}_B(T-1) - \theta\hat{v}_B(T-1)] - \partial G(T-1)y^*(T-1) = \hat{f}(T-1)$$

$$\begin{aligned}
&\hat{D}_{0B}(\tau)\,[\hat{u}_{0B}(\tau) - \theta(\hat{v}^*_{0B}(\tau) - \hat{v}^1_{0B}(\tau))] + \hat{D}_{1B}(\tau)\,[\hat{u}_{1B}(\tau) - \partial\hat{v}_{1B}(\tau)] \\
&\quad - \theta G(\tau)y^*(\tau) = \hat{f}(\tau) \quad \text{if} \quad t_1 + 1 \leq \tau \leq T-2 \quad , \\
&\hat{D}_{0B}(t_1)\,[\hat{u}_{0B}(t_1) - \theta(\hat{v}^*_{0B}(t_1) - \hat{v}^1_{0B}(t_1))] \\
&\quad + \hat{D}_{1B}(t_1)\,[\hat{u}_{1B}(t_1) - \theta\hat{v}_{1B}(t_1)] + \theta d_j(t_1) = \hat{f}(t_1) \qquad (3.15) \\
&\hat{D}_{0B}(\tau)\,[\hat{u}_{0B}(\tau) + \theta\hat{v}^1_{0B}(\tau)] + \hat{D}_{1B}(\tau)\,[\hat{u}_{1B}(\tau) - \theta\hat{v}_{1B}(\tau)] \\
&\quad = \hat{f}(\tau) \quad \text{if} \quad 0 \leq \tau \leq t_1 - 1 \quad .
\end{aligned}$$

The vectors $\hat{v}^*_{0B}(\tau)$must satisfy the following relations:

$$\hat{v}^*_{0B}(T-1) = -\hat{D}^{-1}_{0B}(T-1)G(T-1)y^*(T-1) = \hat{v}_B(T-1) \quad ,$$

$$\hat{v}^*_{0B}(\tau) = -\hat{D}^{-1}_{0B}(\tau)G(\tau)y^*(\tau) \quad \text{if} \quad t_1 + 1 \leq \tau \leq T-2 \quad ,$$

$$\hat{v}^*_{0B}(t_1) = \hat{D}^{-1}_{0B}(t_1)d_j(t_1) \quad .$$

The vectors $\hat{v}^1_{0B}(\tau)$ satisfy the relations $(0 \leq \tau \leq T-2)$:

$$\hat{v}^1_{0B}(\tau) = \hat{D}^{-1}_{0B}(\tau)\hat{D}_{1B}(\tau)\hat{v}_{1B}(\tau) = \Phi_B(\tau)\hat{v}_{1B}(\tau) \quad .$$

Thus the variation $\hat{v}_{0B}(\tau)$ of control $\hat{u}_{0B}(\tau)\,(\tau = 0,1,\ldots,T-1)$ is defined by:

$$\begin{aligned}
&\hat{v}_{0B}(T-1) = \hat{v}^*_{0B}(T-1) \quad , \\
&\quad \hat{v}_{0B}(\tau) = \hat{v}^*_{0b}(\tau) - \hat{v}^1_{0B}(\tau)\,, \text{ if } t_1 + 1 \leq \tau \leq T-2 \qquad (3.16) \\
&\quad \hat{v}_{0B}(\tau) = -\hat{v}^1_{0B}(\tau) \quad , \quad \text{if } 0 \leq \tau \leq t_1 \quad .
\end{aligned}$$

Using (3.12) and (3.13) we can define the values of vectors $\{v_B(\tau)\}$ associated with the variation of control $\{u_B(\tau)\}$. Thus, if a new column $w_j(t_1)$ associated with a column $d_j(t_1)$ is introduced into the basis W_B, then the variation of a basic feasible control $\{u_B, u_N\}$ is defined by (cf. (1.15)):

$$\hat{u}^{\theta}_{0B}(\tau) = \hat{u}_{0B}(\tau) - \theta\hat{v}_{0B}(\tau) \quad . \tag{3.17}$$

We shall refer to the determining of the variation $\{\hat{u}^{\theta}(\tau)\}$ of a feasible control $\{\hat{u}(t)\}$ as Procedure 2. The variation $\{\hat{u}^{\theta}(\tau)\}$ is satisfied to the constraints (1.1) to (1.3) of Problem 1.1 by definition. As $\{\hat{u}(\tau)\}$ is a feasible control, then the constraints (1.4) will also be satisfied for sufficiently small $\theta \geq 0$. Hence the control $\{\hat{u}^{\theta}(\tau)\}$ is feasible if $0 \leq \theta \leq \theta_0$. The value of θ_0 is defined by relations (cf. (1.16)):

$$(\ell,t_2) = \text{arg-min}\, \frac{\hat{u}_{0i}(\tau)}{\hat{v}_{0i}(\tau)} \quad ; \tag{3.18}$$

$$\theta_0 = \frac{\hat{u}_{0\ell}(t_2)}{\hat{v}_{0\ell}(t_2)} \quad ,$$

where the minimum if taken over all $(i,\tau) \in I_0(u), \hat{v}_{0i}(\tau) > 0$ and $\hat{u}_{0i}(\tau)$, $\hat{v}_{0i}(\tau)$ are the i-th components of vectors $\hat{u}_{0B}(\tau)$, $\hat{v}_{0B}(\tau)$.

The equality (3.18) follows from (1.4) and (3.16); minimum in (3.18) is achieved at single pair (ℓ,t_2) in the nondegenerate case.

Let us now define the variation of trajectory $\{x(t)\}$. Considering (3.5), (3.13) and (3.15), we find that the variation of trajectory $x^{\theta}(\tau) = x(\tau) - \theta y(\tau)$ $(\tau = 1,\ldots,T)$ will be defined by

$$\begin{aligned} y(T) &= y^{*}(T) \\ y(\tau+1) &= y^{*}(\tau+1) + B^{1}_{B}(\tau)\hat{v}_{1B}(\tau) \qquad (\tau = T-2,\ldots,1,0) \end{aligned} \tag{3.19}$$

where the vectors $y^{*}(\tau) = 0$ if $0 \leq \tau \leq t_1$, and $y^{*}(\tau+1) = A(\tau)y^{*}(\tau) + \hat{B}_{0B}(\tau)\hat{v}^{*}_{0B}(\tau)$, if $t_1 + 1 \leq \tau \leq T-1$.

4. Objective Function Variation

The special feasible variation of a basic feasible control has been built up in the previous section. Now we determine the corresponding variation of the objective function (1.5) when a column vector $d_j(t_1)$, $(j,t_1) \in I_N(u)$ is introduced in the basis W_B.

In accordance with (3.19),

$$J_1(u^\theta) = a(T)x(T) - \theta a(T)y^*(T) \quad .$$

Denote the variation of the objective function by

$$\Delta_j(t_1) \equiv \Delta J_1(u^\theta) = (J_1(u^\theta) - J_1(u))/\theta = a(T)y^*(T) \quad , \quad (4.1)$$

where indices (j,t_1) show that the variation has been caused by introduction of the column $d_j(t_1)$, $(j,t_1) \in I_N(u)$ to the basis.

By substituting $y^*(T)$ from (3.11) with $\tau = T-1$ into (4.1), we obtain

$$\Delta_j(t_1) = a(T)A(T-1)y^*(T-1) + a(T)\hat{B}_{0B}(T-1)\hat{v}^*_{0B}(T-1) \quad . \quad (4.2)$$

Considering (3.16), (2.15) and (1.12), we rewrite (4.2) as

$$\begin{aligned} \Delta_j(t_1) &= q(T-1)y^*(T-1) + q(T-1)B^1_B(T-2)\hat{v}_{1B}(T-2) \\ &\quad + q(T)B_B(T-1)v_B(T-1) \quad , \end{aligned} \quad (4.3)$$

where $B_B(T-1)$ is the matrix generated by basis columns of the matrix $B(T-1)$, variation $v_B(T-1)$ is associated with basic components of the vector $u_B(T-1)$.

By substituting

$$y^*(T-1) = A(T-2)y^*(T-2) + \hat{B}_{0B}(T-2)\hat{v}^*_{0B}(T-2)$$

into (4.3) and again using (1.12), we obtain

$$\begin{aligned} \Delta_j(t_1) &= q(T-2)y^*(T-2) + q(T-1)\hat{B}_{0B}(T-2)\hat{v}^*_{0B}(T-2) \\ &\quad + q(T-1)B^1_B(T-2)\hat{v}_{1B}(T-2) + q(T)B_B(T-1)v_B(T-1) \quad . \end{aligned}$$

Considering (2.17) and (3.16), we can express $\Delta_j(t_1)$ in the form

$$\Delta_j(t_1) = q(T-2)y^*(T-2) + q(T-1)\hat{B}_{0B}(T-2)\hat{v}_{0B}(T-2)$$
$$+ q(T-1)\hat{B}_{1B}(T-2)\hat{v}_{1B}(T-2) + q(T)B_B(T-1)v_B(T-1) \quad .$$

Hence and from (2.15) it follows that

$$\Delta_j(t_1) = q(T-2)y^*(T-2) + q(T-1)\hat{B}_B(T-2)\hat{v}_B(T-2)$$
$$+ q(T)B_B(T-1)v_B(T-1) \quad .$$

Eventually by induction we obtain for all $(j,t_1) \in I_N(u)$:

$$\Delta_j(t_1) = \sum_{\tau=0}^{T-1} q(\tau+1)B_B(\tau)v_B(\tau) - q(t_1+1)b_j(t_1) \quad . \qquad (4.5)$$

One can see that vectors $v_B(\tau)$ $(\tau = 0,1,\ldots,T-1)$ are a solution of the equations system (1.13). The solution is obtained by means of the compact inverse matrix Procedure 2, which is analogous to Procedure 1 of basic feasible control computation.

Comparing (4.5) and (1.14), we can write

$$\Delta_j(t_1) = z_j(t_1) - c_j(t_1) = \sum_{\tau=0}^{T-1} q(\tau+1)B_B(\tau)v_B(\tau)$$
$$- q(t_1+1)b_j(t_1) \quad .$$

Using the dual Problem 1.2, we can now obtain another form for the definition of the objective function variation $\Delta_j(t_1)$. This form corresponds to (1.17) and is more convenient in practice.

By substituting the expression $\hat{v}^*_{0B}(T-1)$ from (3.10) at $\tau = T-1$ into (4.2), one can obtain

$$\Delta_j(t_1) = a(T)A(T-1)y^*(T-1) - a(T)\hat{B}_{0B}(T-1)\hat{D}_{0B}^{-1}(T-1)G(T-1)y^*(T-1) \quad .$$

Define a vector $\lambda(T-1)$ as $\lambda(T-1) = a(T)\hat{B}_{0B}(T-1)\hat{D}_{0B}^{-1}(T-1)$. Then $\Delta_j(t_1) = p(T-1)y^*(T-1)$, where the vector $p(T-1)$ is computed from dual state equation (1.6) with boundary condition (1.7) at $t = T-1$.

By induction we obtain

$$\Delta_j(t_1) = \lambda(t_1)d_j(t_1) - p(t_1+1)b_j(t_1) \quad , \quad (j,t_1) \in I_N(u) \quad ,$$

where

$$\lambda(t) = p(t+1)\hat{B}_{0B}(t)\hat{D}^{-1}_{0B}(t) \tag{4.6}$$

and the variables $\lambda(t)$, $p(t+1)$ satisfy the dual state equation (1.6) with boundary condition (1.7).

Theorem 5.1: Vectors $\{\lambda(t)\}$ *computed from (4.6), (1.6) and (1.7) are the simplex-multipliers for the basis* W_B.

Proof: It is sufficient to show, in accordance with the definition of simplex-multipliers [1], that vectors $\lambda(t)$ satisfy the dual constraints (1.8) as equalities for basic indices; that is,

$$p(t+1)b_j(t) - \lambda(t)d_j(t) = 0 \quad , \quad (j,t) \in I_B(u) \quad .$$

For this, let us consider the constraints (1.8) of the dual Problem 2.1 relative to the current basis W_B of the primal Problem 1.1. They can be written at $t=0$ as

$$\lambda(0)D_B(0) = p(1)B_B(0) \quad . \tag{4.7}$$

As a nonsingular matrix $\hat{D}_{0B}(0)$ can be generated by columns of the matrix $D_B(0)$, then (4.7) can be rewritten as

$$\lambda(0)\hat{D}_{0B}(0) = p(1)\hat{B}_{0B}(0)$$
$$\lambda(0)\hat{D}_{1B}(0) = p(1)\hat{B}_{1B}(0) \quad .$$

Now we obtain

$$p(1)\left[\hat{B}_{0B}(0)\hat{D}^{-1}_{0B}(0)\hat{D}_{1B}(0) - \hat{B}_{0B}(0)\right] = 0$$

or, in accordance with (2.17),

$$p(1)B_B^1(0) = 0 \quad . \qquad (4.8)$$

Using the state equations (1.6), the conditions (4.8) can be rewritten as

$$p(2)A(1)B_B^1(0) - \lambda(1)G(1)B_B^1(0) = 0 \quad .$$

Hence and from (1.8), we obtain for the next step,

$$\lambda(1) = p(2)\hat{B}_{0B}(1)\hat{D}_{0B}^{-1}(0) \quad .$$

By induction,

$$\lambda(t) = p(t+1)\hat{B}_{0B}(t)\hat{D}_{0B}^{-1}(t)$$

holds for all $t = 1,2,\ldots,T-1$, where matrices $\hat{B}_{0B}(t)$ and $\hat{D}_{0B}^{-1}(t)$ are defined in Section 2. This completes the proof.

Define Procedure 3 by formulas (4.6), (1.6), and (1.7). Procedure 3 allows computation of the values of simplex-multipliers $\{\lambda(t)\}$ for the current basis W_B.

It should be noted that for computing both the values of vectors $\{\lambda(t),p(t+1)\}$ and the values of vectors $\{u(t),x(t)\}$, one can use the same matrices $\hat{D}_{0B}^{-1}(t)$, $\hat{D}_{1B}(t)$, $\hat{B}_{0B}(t)$, and $B_B^1(t)$.

5. Conclusion

As has been shown above, the basis W_B of dimension $mT \times mT$ of the equivalent Problem 1.1a can be replaced by the system of T local bases $\{\hat{D}_{0B}(t)\}$ of dimensions $m \times m$. In this case, all simplex operations (primal, dual solutions, pricing, etc.) can be effectively implemented using this system of local bases.

On the other hand, the original Problem 1.1 can be considered as a structured linear programming problem with constraints (1.1) to (1.4). The basic matrix $\bar{B}$ for this problem has dimension

$(m+n)T \times (m+n)T$. One can easily see that the basic control $u = \{u_B, u_N\}$, determined from Procedure 1 of Section 2 with the corresponding trajectory x, is a basic solution for linear programming Problem 1.1.

The separate operations and the whole algorithm of the dynamic simplex method will be considered in the next part.

References

[1] Dantzig, G.B., *Linear Programming and Extensions*, University Press, Princeton, N.J., 1963.

[2] Kantorovich, L.V., *The Best Use of Economic Resources*, Harvard University Press, Cambridge, Mass., 1965.

[3] Propoi, A.I., *Problems of Dynamic Linear Programming*, RM-76-78, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1976.

[4] Haefele, W. and A.S. Manne, *Strategies of a Transition from Fossil to Nuclear Fuels*, RR-74-07, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1974.

[5] Aganbegian, A.G. and K.K. Valtukh, Izpol'zovanie Narodnokhoziastvennykh Modelei v Planirovanii (Utilization of National Economy Models in Planning), *Ekonomica*, M., (1975) (in Russian).

[6] Swart, W. a.o., Expansion Planning for a Large Dairy Farm, in H. Salkin and J. Saha, eds., *Studies in Linear Programming*, North-Holland, New York, 1975.

[7] Glassey, C.R., Dynamic Linear Programming for Production Scheduling, *Operations Research*, 18, 1 (1970).

[8] Ho, J.K. and A.S. Manne, Nested Decomposition for Dynamic Models, *Mathematical Programming*, 6, 2 (1974).

[9] Propoi, A.I. and A.B. Yadykin, Parametric Iterative Methods for Dynamic Linear Programming. I. Non-Degenerative Case, *Avtomatika i Telemekhanika*, 12, (1975); II. General Case, *Avtomatika i Telemekhanika*, 1, (1976) (in Russian).

[10] Krivonozhko, V.E. and A.I. Propoi, A Method for DLP with the Use of Basis Matrix Factorization, IX International Symposium on Mathematical Programming, Budapest, 1976.

[11] Krivonozhko, V.E. and A.I. Propoi, Method of Successive Control Improvement for DLP, I, II, *Izv.Akad.Nauk SSSR, Technicheskaia Kivernetika*, N3 and N4, 1978(in Russian).

[12] Propoi, A., *Dual Systems of Dynamic Linear Programming,* Part I of this issue.

[13] Propoi, A., *Elementy Teorii Optimalnykh Diskretnykh Protsessov* (Elements of the Theory of Optimal Discrete Processes), Nauka, Moscow, 1973, (in Russian).

[14] Gantmacher, F.R., *The Theory of Matrices*, Chelsea Publishing Co., New York, 1960.

II. THE DYNAMIC SIMPLEX METHOD: A BASIS FACTORIZATION APPROACH

1. Introduction

In this part, separate operations and the general scheme of the dynamic simplex-method will be described. An illustrative numerical example and the theoretical evaluation of the algorithm are given. In conclusion, we consider briefly important extensions of the algorithm (non-negative state constraints, time delays in state and control variables, etc.).

For convenience, we repeat the statement of the problem below [1].

Problem 1.1: Find a control $u = \{u(0),\ldots,u(t-1)\}$ and a corresponding trajectory $x = \{x(0),\ldots,x(T)\}$ satisfying the state equations

$$x(t+1) = A(t)x(t) + B(t)u(t) \qquad (1.1)$$

with initial condition

$$x(0) = x^0 \qquad (1.2)$$

and constraints

$$G(t)x(t) + B(t)u(t) = f(t) \qquad (1.3)$$

$$u(t) \geq 0 \qquad (1.4)$$

which maximize the objective function

$$J_1(u) = a(T)'x(T) \quad . \qquad (1.5)$$

Here we use the same notations as in Part I.

Problem T.1 can be considered as some "large" linear programming problem with constraints (1.1) to (1.4). The constraint matrix of Problem 1.1 has a staircase structure and dimension $(r+n)T \times (m+n)T$; decision variables are $\{u,x\} = \{u_k(t),x_i(t+1)$ $(k=1,\ldots,r;\ i=1,\ldots,n;\ t=0,\ldots,T-1)\}$.

We shall denote a basic feasible solution of Problem 1.1 by $\{u_B, x\}$ (the free variables x are always in a basis). Evidently, u_B is a basic feasible control in the sense of Definition 1.2 [1].

2. Basis Factorization Approach

The method which was considered in [1], can be interpreted as some basis factorization approach to Problem 1.1's solution. Below we describe the method in these terms.

We need the following assertion.

Theorem 2.1: [2]: *Let a non-singular square matrix* F *be partitioned into blocks*

$$F = \begin{array}{cc} \overbrace{\hphantom{H}}^{m} & \overbrace{\hphantom{P}}^{n} \\ \end{array} \left[\begin{array}{c:c} H & P \\ \hdashline Q & R \end{array}\right]\begin{array}{l}\}m \\ \}n\end{array} \qquad (2.1)$$

where H is a non-singular matrix.

Then F *is represented as the product of upper and lower triangular matrices in the form*

$$F = \bar{F} \cdot U = \left[\begin{array}{c:c} H & O \\ \hdashline Q & C \end{array}\right] \cdot \begin{array}{c} \overbrace{\hphantom{I_m}}^{m} \; \overbrace{\hphantom{\Phi}}^{n} \\ \left[\begin{array}{c:c} I_m & \Phi \\ \hdashline O & I_n \end{array}\right]\begin{array}{l}\}m \\ \}n\end{array} \end{array} , \qquad (2.2)$$

where

$$C = R - QH^{-1}P \;, \qquad |C| \neq 0 \;, \qquad \Phi = H^{-1}P \;, \qquad (2.3)$$

I_m *and* I_n *are the identity matrices of appropriate dimensions; the inverse of each of the factors is readily obtained and their product yields the inverse of* F:

$$F^{-1} = \left[\begin{array}{c:c} H^{-1}+H^{-1}PC^{-1}QH^{-1} & -H^{-1}PC^{-1} \\ \hdashline -C^{-1}QH^{-1} & C^{-1} \end{array}\right] . \qquad (2.4)$$

Theorem 2.1 is not stated in [2] in explicit form, but directly follows from results given in [2].

We now apply the theorem to Problem 1.1. The basis matrix $\bar{B}$ of Problem 1.1 has the same structure as the constraint matrix:

$$\bar{B} = \begin{bmatrix} D_B(0) & & & & & & \\ B_B(0) & -I & & & & & \\ & G(1) & D_B(1) & & & & \\ & A(1) & B_B(1) & -I & & & \\ & & & \ddots & & & \\ & & & & -I & & \\ & & & & G(T-1) & D_B(T-1) & \\ & & & & A(T-1) & B_B(T-1) & -I \end{bmatrix} \quad (2.5)$$

where I is the identity matrix of dimension n × n, $D_B(t)$ and $B_B(t)$ are submatrices, formed by basic columns of the constraint matrix.

As the rows of $D_B(0)$ are linearly independent, one can choose m linearly independent columns in the matrix $D_B(0)$. These columns generate the matrix $\hat{D}_{0B}(0)$.

By column permutation, we can transform the matrix $D_B(0)$ and obtain $D_B(0) = [\hat{D}_{0B}(0);\hat{D}_{1B}(0)]$, where $\hat{D}_{1B}(0)$ is the submatrix, consisting of the columns of the matrix $D_B(0)$ which are not in the matrix $\hat{D}_{0B}(0)$.

The column permutation of the matrix $D_B(0)$ implies the corresponding partition of the matrix $B_B(0)$: $B_B(0) = [\hat{B}_{0B}(0);\hat{B}_{1B}(0)]$.

In accordance with Theorem 2.1, one can show that the matrix $\bar{B}$ is expressed as

$$\bar{B} = \bar{B}_0 U_0 \quad (2.6)$$

where U_0 is the upper triangular matrix whose dimensions conform with those of $\bar{B}$.

$$\bar{U}_0 = \begin{bmatrix} 1 & & & & \\ & \ddots & & \Phi_B(0) & \\ & & \ddots & & \\ & 0 & & \ddots & \\ & & & & 1 \end{bmatrix}$$

In the matrix U_0, the dimension and location of the matrix

$$\Phi_B(0) = \hat{D}_{0B}^{-1}(0)\hat{D}_{1B}(0)$$

coincide with the dimension and location of the matrix $\hat{D}_{1B}(0)$ in $\bar{B}$. The matrix $\bar{B}_0$ is obtained from the matrix $\bar{B}$ through replacement $[\hat{D}_{0B}(0);\hat{D}_{1B}(0)]$ by $[\hat{D}_{0B}(0);0]$ and $\hat{B}_{1B}(0)$ by $B_B^1(0) = \hat{B}_{1B}(0) - \hat{B}_{0B}(0)\Phi_B(0)$.

In the matrix $\bar{B}_0$, we permute the submatrix -I and the submatrix $B_B^1(0)$. Then we permute the submatrices G(1) and A(1) in the matrix $\bar{B}_0$ and the submatrix $\Phi_B(0)$ in the matrix U_0 respectively.

By analogy with (2.6), we can write that $\bar{B}_0 = \bar{B}_1\bar{V}_0$, where V_0 is the upper triangular matrix of the matrix $\bar{B}_0$ dimension:

$$V_0 = \begin{bmatrix} 1 & & & & \\ & \ddots & & -B_B^1(0) & \\ & & \ddots & & \\ & 0 & & \ddots & \\ & & & & 1 \end{bmatrix} ,$$

$$B_B^1(0) = \hat{B}_{1B}(0) - \hat{B}_{0B}(0)\Phi_B(0) ,$$

and

$$\bar{B}_1 = \begin{bmatrix} \hat{D}_{0B}(0) & & & & & & & & & & \\ \hat{B}_{0B}(0) & -I & & & & & & & & & \\ & G(1) & G(1)B_B^1(0) & D_B(1) & & & & & & & \\ & A(1) & A(1)B_B^1(0) & B_B(1) & -I & & & & & & \\ & & & & G(2) & D_B(2) & & & & & \\ & & & & A(2) & B_B(2) & -I & & & & \\ & & & & & & & \ddots & & & \\ & & & & & & & & G(T-1) & D_B(T-1) & \\ & & & & & & & & A(T-1) & B_B(T-1) & -I \end{bmatrix}$$

The dimension and location of the matrix $-B_B^1(0)$ in V_0 coincide with the dimension and location of the matrix $B_B^1(0)$ in $\bar{B}_0$. The matrix $\bar{B}_1$ is obtained from $\bar{B}_0$ by the replacement of submatrices

$$\begin{array}{rcl} [-I : B_B^1(0)] & \text{by} & [-I : 0] \quad , \\ [G(1) : 0 : D_B(1)] & \text{by} & [G(1) : G(1)B_B^1(0) : D_B(1)] \quad , \\ [A(1) : 0 : B_B(1)] & \text{by} & [A(1) : A(1)B_B^1(0) : B_B(1)] \quad . \end{array}$$

In accordance with Theorem 2.1, a matrix, obtained from the matrix $\bar{B}_1$ by cutting out the rows coinciding with the rows of submatrices $\hat{D}_{0B}(0)$ and $\hat{B}_{0B}(0)$ and by cutting out the columns coinciding with the columns of submatrices $\hat{D}_{0B}(0)$ and $G(1)$, is nonsingular. Consequently, the rows of the matrix

$$[G(1)B_B^1(0) : D_B(1)]$$

are linearly independent, and by column permutation, this matrix can be reduced to the form

$$[G(1)B_B^1(0) : D_B(1)] = [\hat{D}_{0B}(1) : \hat{D}_{1B}(1)] \quad ,$$

where the matrix $\hat{D}_{0B}(1)$ is nonsingular and the matrix $\hat{D}_{1B}(1)$ is generated by columns $[G(1)B_B^1(0) : D(1)]$, which are not in the matrix $\hat{D}_{0B}(1)$.

The matrices

$$[A(1)B_B^1(0) : B_B(1)] = [\hat{B}_{0B}(1) : \hat{B}_{1B}(1)]$$

and $\Phi_B(0)$ in matrix U_0, as well as the matrix $-B_B^{-1}(0)$ in the matrix V_0, are partitioned similarly.

Proceeding in a similar way, we obtain

$$\bar{\bar{B}} = B^* V_{T-2} U_{T-2} \cdots\cdots V_0 U_0 = B^* U \quad , \tag{2.7}$$

where

$$B^* = \begin{bmatrix} \hat{D}_{0B}(0) & & & & & & \\ \hat{B}_{0B}(0) & -I & & & & & \\ & G(1) & \hat{D}_{0B}(1) & & & & \\ & A(1) & \hat{B}_{0B}(1) & -I & & & \\ & & & & \ddots & & \\ & & & & & G(T-1) & \hat{D}_{0B}(T-1) \\ & & & & & A(T-1) & \hat{B}_{0B}(T-1) \quad -I \end{bmatrix} \tag{2.7a}$$

where $\hat{D}_{0B}(t)$ $(t = 0,\ldots,T-1)$ is a square non-singular matrix of dimension $m \times m$ and is formed either by columns of the matrix $D(t)$ or by some columns of matrices $D(\tau)$ $(\tau = 0,\ldots,t-1)$, which are recomputed to step t during factorization process. Evidently, the matrices $\hat{D}_{0B}(t)$ $(t = 0,1,\ldots,T-1)$, obtained in such a way, coincide with the local bases, which were defined in [1].

The matrices U_t and V_t $(t = 0, 1, \ldots, T-2)$ are

$$U_t = \begin{bmatrix} 1 & & & & 0 & & \\ & 1 & & & & & \\ & & .\ 0 \ldots \Phi_0^{t+1}(t) \ldots 0 \ldots \Phi_i^j(t) \ldots 0 \ldots \Phi_t^{T-1}(t) \ldots 0 & & & & \\ & & \ddots & & & 0 & \\ & & & \ddots & & & \\ & 0 & & & \ddots & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix}$$

and

$$V_t = \begin{bmatrix} 1 & & & & 0 & & \\ & 1 & & & & & \\ & & .\ 0 \ldots -B_0^{t+1}(t) \ldots 0 \ldots -B_i^j(t) \ldots 0 \ldots -B_t^{T-1}(t) \ldots 0 & & & & \\ & & \ddots & & & 0 & \\ & & & \ddots & & & \\ & 0 & & & \ddots & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix}$$

where $\Phi_i^j(t)$ and $-B_i^j(t)$ correspond to those basic control variables $u_B(i)$, which enter local basis $\hat{D}_{0B}(j)$. Location of rows of submatrices $\Phi_i^j(t)$ and $-B_i^j(t)$ correspond to the location of rows of submatrices $\hat{D}_{0B}(t)$ and $\hat{B}_{0B}(t)$ in B^*.

We denote the non-zero columns in the right corners of the matrices U_t and V_t by $\Phi_B(t)$ and $B_B^1(t)$:

$$\Phi_B(t) = [\Phi_0^{t+1}(t) \ \dots \ \Phi_i^j(t) \ \dots \ \Phi_t^{T-1}(t)]$$

$$B_B^1(t) = [-B_0^{t+1}(t) \ \dots \ -B_i^j(t) \ \dots \ -B_t^{T-1}(t)] \quad .$$

By construction, these matrices are defined from

$$\Phi_B(t) = \hat{D}_{0B}^{-1}(t)\hat{D}_{1B}(t) \quad , \tag{2.8}$$

$$B_B^1(t) = \hat{B}_{1B}(t) - \hat{B}_{0B}(t)\Phi_B(t) \quad . \tag{2.9}$$

One can see that these matrices conform with the matrices defined by fomulas (2.12) and (2.17) in [1].

Taking into account the permutation of basis columns in the factorization process, we can write the basic variables as

$$\{u_B, x\} = \{\hat{u}_{0B}(0), x(1), \hat{u}_{0B}(1), \dots, \hat{u}_{0B}(T-1), x(T)\} \quad ,$$

where vector $\hat{u}_{0B}(t)$ corresponds to matrix $\hat{D}_{0B}(t)$ $(t = 0, 1, \dots, T-1)$.

At each simplex iteration, it is necessary to solve three system of linear equations for:

(1) determination of a basic solution;

(2) computation of coefficients $\{v, y\}$ which are the representation of the incoming vector

$$Y_j(t_1) = (0, \dots, 0, d_j^T(t_1), b_j^T(t_1), 0, \dots, 0)^T$$

in terms of the basis;

(3) determination of the simplex-multipliers.

Now we describe these procedures for factorized representation of the basis. We single out the following procedures: the primal solution, the dual solution; pricing and updating.

3. Primal Solution

Vector $X = (u_B, x)$ is calculated from the solution of the system

$$\bar{B}X = B^*UX = B^*V_{T-2} \ldots U_0X = b \quad , \tag{3.1}$$

where b is the constraint vector of Problem 1.1.

Denote

$$X^* = UX \quad ;$$

then the calculation of the vector X reduces to subsequent solution of two systems of linear equations in forward and backward runs:

$$B^*X^* = b \quad , \tag{3.2}$$

$$UX = X^* \quad . \tag{3.3}$$

The solution of (3.2) is determined by recurrent formulas:

$$\begin{aligned} \hat{u}^*_{0B}(t) &= \hat{D}^{-1}_{0B}(t)(f(t) - G(t)x^*(t)) \qquad (t = 0,\ldots,T-1) \quad , \\ x^*(t+1) &= A(t)x^*(t) + \hat{B}_{0B}(t)\hat{u}^*_{0B}(t) \qquad (t = 0,\ldots,T-1) \quad , \\ x^*(0) &= x(0) \quad . \end{aligned} \tag{3.4}$$

The system (3.3), considering (2.3), can be written as

$$X = U_0^{-1} \ldots V_{T-2}^{-1}X^* \quad .$$

It is easy to see that the matrices U_t^{-1} and V_t^{-1} are obtained from the matrices U_t and V_t by simply changing the signs of the elements which are above the main diagonal. Therefore the solution of the system (3.3) reduces to the recurrent formulas:

$$\begin{aligned} x(T) &= x^*(T) \quad , \\ u(T-1) &= u^*(T-1) \quad , \\ x(T) &= x^*(t) + \sum_{i=0}^{t-1} \sum_{j=t}^{T-1} [B_i^j(t) : 0]\hat{u}_{0B}(j) \ , \ (t=T-1,\ldots,1) \\ \hat{u}_{0B}(t) &= \hat{u}^*_{0B}(t) - \sum_{i=0}^{t} \sum_{j=t+1}^{T-1} [\Phi_i^j(t) : 0]\hat{u}_{0B}(j), (t=T-2,\ldots,0) \quad . \end{aligned} \tag{3.5}$$

Here the notation $[B_i^j(t) : 0]$ and $[\Phi_i^j(t) : 0]$ denote that the matrices $B_i^j(t)$ and $\Phi_i^j(T)$ are augmented by zeros, if necessary, so that the matrices conform with multiplying.

The coefficients

$$\bar{Y}_j(t_1) = (\hat{v}_{0B}(0), y(1), \hat{v}_{0B}(1), \ldots, y(T)) \quad ,$$

which represent the vector $Y_j(t_1)$ in terms of the basis, are calculated from the solution of the system

$$\bar{B}\bar{Y}_j(t_1) = Y_j(t_1) \quad .$$

On the forward run, we find the vector sequence (v*,y*):

$$\begin{aligned} \hat{v}^*_{0B}(t) &= 0 \quad , \\ y^*(t+1) &= 0 \quad , \quad (t=0,\ldots,t_1-1) \quad , \\ \hat{v}^*_{0B}(t_1) &= \hat{D}^{-1}_{0B}(t_1)d_j(t_1) \quad , \\ y^*(t_1+1) &= \hat{B}_{0B}(t_1)\hat{v}^*_{0B}(t_1) - b_j(t_1) \quad , \\ \hat{v}^*_{0B}(t) &= -\hat{D}^{-1}_{0B}(t)G(t)y^*(t) \quad , \\ y^*(t+1) &= A(t)y^*(t) + \hat{B}_{0B}(t)\hat{v}^*_{0B}(t), \quad (t=t_1+1,\ldots,T-1) \quad . \end{aligned} \tag{3.6}$$

On the backward run, we find vector sequence (v,y);

$$
\begin{aligned}
y(T) &= y^*(T) \quad , \\
\hat{v}_{0B}(T-1) &= v^*_{0B}(T-1) \quad , \\
y(t) &= y^*(t) + \sum_{i=0}^{t-1}\sum_{j=t}^{T-1} [B_i^j(t) : 0]\hat{v}_{0B}(j) , \qquad (t = T-1,\dots,1) \quad , \qquad (3.7) \\
\hat{v}_{0B}(t) &= \hat{v}^*_{0B}(t) - \sum_{i=0}^{t}\sum_{j=t+1}^{T-1} [\Phi_i^j(t) : 0]\hat{v}_{0B}(j) \qquad (t = T-2,\dots,0) \quad .
\end{aligned}
$$

For given sequences $\hat{u}$ and $\hat{v}$, the pair of indices (ℓ,t_2) which correspond to the outgoing vector, is defined by

$$
\theta_0 = \min_{\substack{(i,\tau) \\ \hat{v}_{0i}(\tau)>0}} \frac{\hat{u}_{0i}(\tau)}{\hat{v}_{0i}(\tau)} = \frac{\hat{u}_{0\ell}(t_2)}{\hat{v}_{0\ell}(t_2)} \quad . \qquad (3.8)
$$

4. Dual Solution

We define $(n+m)T$-vector $\pi = \{\lambda,p\}$ as

$$
c_B = \pi\bar{B} \qquad (4.1)
$$

where $\bar{B}$ is a basis matrix (2.5) and $c_B = \{0,\dots,0,a(T)\}$. From (4.1) and representation (2.7) of the basis matrix $\bar{B}$, we can calculate the simplex-multipliers $\{\lambda,p\} = \{\lambda(0),p(1),\dots,\lambda(T-1),p(T)\}$ in a similar way using the same matrices $\hat{B}_{0B}(t)$ and $\hat{D}_{0B}^{-1}(t)$:

$$
\begin{aligned}
p(T) &= a(T) \quad , \\
\lambda(t) &= p(t+1)\hat{B}_{0B}(t)\hat{D}_{0B}^{-1}(t) , \qquad (t = T-1,\dots,0) \qquad (4.2) \\
p(t) &= p(t+1)A(t) - \lambda(t)G(t) , \qquad (t = T-1,\dots,1) \quad .
\end{aligned}
$$

One can see that the formulas (3.4) to (3.7) are the explicit expression of Procedure 1 [1] for determination of basic variables and coefficients, expressing a column not in the basis by the basis columns. The formulas (4.2) for determination of simplex-multipliers coincide with the formulas of Procedure 3 [1].

5. Pricing

The pricing procedure is now constructed straightforwardly. To price out a vector $d_j(t)$ which is not in the basis, we use formulas [1]:

$$\Delta_j(t) = \lambda(t)d_j(t) - p(t+1)b_j(t) \tag{5.1}$$

where the simplex-multipliers $\lambda(t)$ and $p(t+1)$ are defined from (4.2).

It should be noted that the method requires only partial pricing: that is, to determine $\lambda(t_1)$ and $p(t_1+1)$, which are needed for pricing out the nonbasic components of vector $u(t_1)$, one has to calculate vectors $\lambda(t)$ and $p(t+1)$ only for $t = T-1, T-2, \ldots, t_1+1$. These computations require only a part of the basis inverse representation, in particular, only a few of the local bases. In a standard approach it is generally not possible to compute part of the components of the simplex-multiplier vector without computing the whole vector.

6. Updating

The pricing procedure of computing the values $\lambda_j(t)$ for vectors $d_j(t)$, $(j,t) \in I_N(u)$, which are not in the basis allows us to define the vector to be introduced into the basis and the vector to be removed from the basis.

Let $d_j(t_1)$ be the ingoing column vector and $\hat{d}_{0\ell}(t_2)$ be the outgoing column vector. Here $d_j(t_1)$ is the j-th nonbasic column of the matrix $D(t_1)$ and $\hat{d}_{0\ell}(t_2)$ is the ℓ-th column of the matrix $\hat{D}_{0B}(t_2)$, $0 \leq t_1, t_2 \leq T-1$.

Replacing the vector $d_j(t_1)$ by the vector $\hat{d}_{0\ell}(t_2)$ implies the updating of the old system of local basis $\{\hat{D}_{0B}(t)\}$ by a new system of local bases $\{\hat{D}_{0B}(t)\}'$.

As in the case of the static simplex method, the updating procedure must be done efficiently as it constitutes the main effort of each iterative cycle of the algorithm.

In the dynamic simplex method, we operate with the system of inverses $\{\hat{D}_{0B}^{-1}(t)\,(t = 0,1,\ldots,T-1)\}$ of local bases. Hence the efficiency of the method will be directly defined by the updating scheme of the inverses $\{\hat{D}_{0B}^{-1}(t)\}$.

The main complications of the updating procedure in the dynamic case is the fact that, first, the updating of a local basis at step t can change the subsequent local bases $\hat{D}_{0B}(\tau)$ $(\tau = t+1,\ldots,T-1)$ and that, second, the outgoing vector $\hat{d}_{0\ell}(t_2)$ may belong to the local basis $\hat{D}_{0B}(t_2)$ at another step t_2, $t_2 \neq t_1$.

The theorem below gives a sufficient condition when the replacement of a basis column in a local basis $\hat{D}_{0B}(t)$ does not change the other local bases.

Theorem 6.1: The replacement of the i-*th column in a local basis* $\hat{D}_{0B}(t)$ *does not change the other local bases, if the* i-*th row of the matrix* $\Phi_B(t)$, *where* $\Phi_B(t)$ *defined by (2.8), vanishes.*

Proof: When we replace the i-th column in the matrix $\hat{D}_{0B}(t)$, then in accordance with (2.7), the updating of the matrix $\Phi_B(t)$ will be similar to the updating of the inverse $\hat{D}_{0B}^{-1}(t)$, that is, the i-th pivot row of the matrix is added to the other row with some coefficients [3].

Therefore, if the i-th row of the matrix $\Phi_B(t)$ is zero, the matrix $\Phi_B(t)$ will not change. In accordance with (2.9), the matrix $B_B^1(t)$ does not change either. Considering the construction of matrices $\hat{D}_{0B}(t)$ at next steps, we find that all subsequent local bases $\hat{D}_{0B}(\tau)$ $(\tau = t+1, t+2,\ldots,T-1)$ also do not change.

Consequence 6.1: If an element $\Phi_{ij}(t)$ of the matrix $\Phi_B(t)$ is zero, then the replacement of the i-th column in the local basis $\hat{D}_{0B}(t)$ does not change the j-th column in the matrix $B_B^1(t)$.

Now let us consider the interchange of the ℓ-th column of local basis $\hat{D}_{0B}(t)$ with a column of local basis $\hat{D}_{0B}(t+1)$ and find how the inverses $\hat{D}_{0B}^{-1}(\tau)$ and matrices $\Phi_B(\tau)$, $B_B^1(\tau)$ $(\tau = t,\ldots,T-1)$ are updated at this interchange. For this, we need the following theorem.

Theorem 6.2: Let the k*-th column of submatrix* $\begin{bmatrix} H \\ Q \end{bmatrix}$ *of the matrix* F *in (2.1) be interchanged with the* ℓ*-th column of submatrix* $\begin{bmatrix} P \\ R \end{bmatrix}$ *and let the element* $\phi_{k\ell}$ *of the matrix* $\Phi = H^{-1}P$ *be not zero (pivoting element). Then, denoting the updated submatrices in* F *as* H', Q', P' *and* R'*, the following relations hold:*

$$\text{(i)} \quad (H^{-1})' = E_k H^{-1} \tag{6.1}$$

where E_k *is an elementary* $(m \times m)$ *column matrix with elements of the non-zero k-th column:*

$$\eta_i = -\frac{\phi_{i\ell}}{\phi_{k\ell}} \quad (i = 1,\ldots,n,\ i \neq k)$$

$$\eta_k = \frac{1}{\phi_{k\ell}}$$

$$\text{(ii)} \quad \Phi_i' = E_k \Phi_i \quad i \neq \ell, \ \Phi_\ell' = [\eta_1,\ldots,\eta_m]^T \quad , \tag{6.2}$$

$$(C^{-1})' = E_\ell C^{-1} \quad , \tag{6.3}$$

where Φ_i *is the* i*-th column of* Φ, E_ℓ *is an elementary* $(n \times n)$ *row matrix with elements of the non-zero* ℓ*-th row equal to* $-\phi_{\ell i}$ $(i = 1,\ldots,n)$*;*

$$\text{(iii)} \quad C' = CE_\ell^{-1} \tag{6.4}$$

where E_ℓ^{-1} *is an elementary* $(n \times n)$ *row matrix with elements of the non-zero* ℓ*-th row:*

$$\eta_i = -\frac{\phi_{ki}}{\phi_{k\ell}} \qquad (i = 1,\ldots,n; i \neq \ell)$$

$$\eta_\ell = -\frac{1}{\phi_{k\ell}}$$

Proof: Formulas (6.1) and (6.2) are the basis updating formulas in the simplex method [3]. Now, to prove (6.3): the column permutations of the matrix F can be written as a matrix product $\hat{F} = FT$, where

$$T = \left[\begin{array}{ccccccc} 1 & & & \vdots & & & \\ & \ddots & & \vdots & & & \\ & & 0\cdots & \vdots & \cdots 1 & & \\ & & \vdots\ \ddots & \vdots & \vdots & & \\ \cdots & \cdots & \vdots\ \cdots 1 & \vdots & \cdots\ \vdots & \cdots & \cdots \\ & & \vdots & \vdots & 1 & & \\ & & \vdots & \vdots & \vdots\ \ddots & & \\ & & 1\cdots & \vdots & \cdots 0 & & \\ & & & \vdots & & \ddots & \\ & & & \vdots & & & 1 \\ & & \underbrace{\quad}_{k} & & \underbrace{\quad}_{\ell} & & \end{array}\right] \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} m \\ \\ \left.\begin{array}{l} \\ \\ \end{array}\right\} n \end{array} .$$

As $T^{-1} = T$, then $\hat{F}^{-1} = TF^{-1}$. Taking into account the partitioning of the matrices and using Theorem 2.1, we obtain

$$\hat{C}^1 = \begin{bmatrix} 0\cdots 0\cdots 0 \\ \vdots \quad \vdots \quad \vdots \\ 0\cdots 1\cdots 0 \\ \vdots \quad \vdots \quad \vdots \\ 0\cdots 0\cdots 0 \end{bmatrix}_{\underbrace{\quad}_{k}} H^{-1}PC^{-1} + \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 0 & \\ & & \vdots\ \ddots & \\ & & & 1 \end{bmatrix}_{\underbrace{\quad}_{\ell}} C^{-1}$$

$$= -\begin{bmatrix} 0\cdots 0\cdots 0 \\ \vdots \quad \vdots \quad \vdots \\ 0\cdots 1\cdots 0 \\ \vdots \quad \vdots \quad \vdots \\ 0\cdots 0\cdots 0 \end{bmatrix} \Phi C^{-1} + \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 0 & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} C^{-1} = E_\ell C^{-1} \quad .$$

The relation (6.4) follows directly from (6.3). This completes the proof of the theorem.

Now let $\phi_{\ell q}(t) \neq 0$ be the pivoting element of the matrix $\Phi_B(t)$, which correspond to the q-th component of the vector $\hat{u}_{0B}(t+1)$. According to Theorem 6.2, at the interchange of the ℓ-th column of the matrix $\hat{D}_{0B}(t)$ with the q-th column of the matrix $\hat{D}_{1B}(t)$, the inverse $\hat{D}_{0B}^{-1}(t)$ is updated by premultiplying on the elementary matrix. The elementary matrix has dimension m × m and differs from the identity matrix by the ℓ-th column with components [3]:

$$\eta_i = -\frac{\phi_{iq}(t)}{\phi_{\ell q}(t)} \quad (i = 1,\dots,m;\ i \neq \ell) \quad ;$$

$$\eta_i = \frac{1}{\phi_{\ell q}(t)} \quad (i = \ell) \quad .$$

The column permutation in matrices $\hat{B}_{0B}(t)$ and $\hat{B}_{1B}(t)$ is carried out in a similar way. The matrix $B_B^1(t)$ is updated according to Theorem 6.2 as

$$[B_B^1(t)]' = B_B^1(t)E_q \quad , \tag{6.5}$$

where E_q is an elementary row matrix, which differs from the identity matrix by the q-th row with components

$$\xi_i(t) = \frac{\phi_{\ell i}(t)}{\phi_{\ell q}(t)} \quad , \quad i \neq q \quad ;$$

$$\xi_i(t) = -\frac{1}{\phi_{\ell q}(t)} \quad , \quad i = q \quad .$$

Define now the updating of the inverses $\hat{D}_{0B}^{-1}(\tau)$ $(\tau = t+1,\dots, T-1)$.

Theorem 6.3: Let $\phi_{\ell q}(t) \neq 0$ be the pivoting element of the matrix $\Phi_B(t)$ (which corresponds to the q-th component of vector $\hat{u}_{0B}(t+1)$).

Then at the interchange of the ℓ-th column of $\hat{D}_{0B}(t)$ with the q-th column of $\hat{D}_{0B}(t+1)$ the following relations hold:

$$[\hat{D}_{0B}^{-1}(t+1)]' = L_q^{-1}\hat{D}_{0B}^{-1}(t+1) \tag{6.6}$$

$$[\hat{B}_{0B}(t+1)]' = \hat{B}_{0B}(t+1)L_q \tag{6.7}$$

$$[\Phi_B(t+1)]' = L_q^{-1}N_q + L_q^{-1}\Phi_B(t+1) \tag{6.8}$$

where L_q *is an elementary row* (m × m) *matrix, which differs from the unit matrix by the* q*-th row, and* N_q *is a matrix, which differs from the zero matrix by the* q*-th row.*

The matrix $B_B^1(t+1)$ *is not changed at this permutation, neither are all the subsequent local bases* $\hat{D}_{0B}(\tau)$ *and matrices* $\Phi_B(\tau)$, $B_B^1(\tau)$ $(\tau = t+2,\dots,T-1)$.

Proof: Taking into account the structure of the matrices $\hat{D}$ (t+1) and $\hat{B}_B(t+1)$, we can write (after column permutations):

$$\begin{aligned}\hat{D}_B(t+1) &= [G(t+1)B_B^1(t+1);\ D_B(t+1)] \\ &= [\hat{D}_{0B}(t+1);\ \hat{D}_{1B}(t+1)]\end{aligned} \tag{6.9}$$

and

$$\begin{aligned}\hat{B}_B(t+1) &= [A(t+1)B_B^1(t+1);\ B_B(t+1)] \\ &= [\hat{B}_{0B}(t+1),\ \hat{B}_{1B}(t+1)] \quad .\end{aligned} \tag{6.10}$$

Considering (6.9) and (6.10), we obtain

$$\begin{bmatrix}[\hat{D}_{0B}(t+1)]'[\hat{D}_{1B}(t+1)]' \\ [\hat{B}_{0B}(t+1)]'[\hat{B}_{1B}(t+1)]'\end{bmatrix} = \begin{bmatrix}\hat{D}_{0B}(t+1) & \hat{D}_{1B}(t+1) \\ \hat{B}_{0B}(t+1) & \hat{B}_{1B}(t+1)\end{bmatrix}\begin{bmatrix}L_q & N_q \\ 0 & I\end{bmatrix} \tag{6.11}$$

Here $[\hat{D}_B(t+1)]'$, $[\hat{B}_B(t+1)]'$ are the updated matrices corresponding to the new basis; L_q is the elementary row (m × m) matrix; N_q is the (m × k) matrix; I is the identity matrix of dimension (k × k).

The right matrix in (6.11) is built up as follows: the matrix E_q in (6.5) is enlarged up to dimension $(m+k) \times (m+k)$ in such a way that in the added part the main diagonal contains units and all the remaining added elements are zero; then the elements of the q-th row are permuted in accordance with the columns permutations of the matrix $\hat{D}_B(t+1)$, when it is partitioned on the matrices $\hat{D}_{0B}(t+1)$ and $\hat{D}_{1B}(t+1)$.

Multiplying the right-hand matrices in (6.11) and taking into account their partitioning, we obtain (6.6) and (6.7). Besides,

$$\begin{aligned} [\hat{D}_{1B}(t+1)]' &= \hat{D}_{0B}(t+1)N_q + \hat{D}_{1B}(t+1) \\ [\hat{B}_{1B}(t+1)]' &= \hat{B}_{0B}(t+1)N_q + \hat{B}_{1B}(t+1) \quad . \end{aligned} \qquad (6.12).$$

According to (2.9), we have

$$[B_B^1(t+1)]' = [\hat{B}_{1B}(t+1)' - [\hat{B}_{0B}(t+1)]'[\hat{D}_{0B}^{-1}(t+1)]'[\hat{D}_{1B}(t+1)]' \qquad (6.13)$$

Substituting (6.6) and (6.7) into (6.13), we obtain

$$\begin{aligned} [B_B^1(t+1)' &= \hat{B}_{0B}(t+1)N_q + \hat{B}_{1B}(t+1) - \hat{B}_{0B}(t+1)L_q L_q^{-1} \\ &\times \hat{D}_{0B}^{-1}(t+1)\ (\hat{D}_{0B}(t+1)N_q + \hat{D}_{1B}(t+1)) \\ &= \hat{B}_{1B}(t+1) - \hat{B}_{0B}(t+1)\hat{D}_{0B}^{-1}(t+1)\hat{D}_{1B}(t+1) \\ &= B_B^1(t+1) \quad . \end{aligned}$$

The matrix $B_B^1(t+1)$ is not changed, therefore all the subsequent local bases $\hat{D}_{0B}(\tau)$ and matrices $\Phi_B(\tau)$, $B_B^1(\tau)$ $(\tau = T+2, \ldots, T-1)$ are not changed.

Finally, taking into account (6.6) and (6.12), we obtain (6.8):

$$[\Phi_B(t+1)]' = [\hat{D}_{0B}^{-1}(t+1)]'[\hat{D}_{1B}(t+1)]' = L_q^{-1}N_q + L_q^{-1}\Phi_B(t+1) \quad .$$

This procedure we shall call *the interchange of the ℓ-th column of the matrix* $\hat{D}_{CB}(t)$ *with the q-th column of the matrix* $\hat{D}_{0B}(t+1)$.

Now let us consider the interchange of the ℓ-th column of the matrix $\hat{D}_{0B}(t)$ with some column of the matrix $\hat{D}_{0B}(t^*)$, $t^* > t+1$.

In the ℓ row of the matrix $\Phi_B(t)$, let the first non-zero element $\phi_{\ell q}(t)$ correspond to the basic variable, which is recomputed to the local basis $\hat{D}_{0B}(t^*)$, and all elements $\phi_{\ell i}(t)$, corresponding to the variables which are recomputed to local bases $\hat{D}_{0B}(\tau)$, $t < \tau < t^*$, equal to zero. Now we partition the matrices $\Phi_B(t)$ and $\hat{B}_{1B}(t)$ into two parts:

$$\begin{aligned} \Phi_B(t) &= [\Phi_1(t); \Phi_2(t)] \quad ; \\ \hat{B}_{1B}(t) &= [\hat{B}_{11}(t); \hat{B}_{12}(t)] \quad ; \\ B_B^1(t) &= [B_1^1(t); B_2^1(t)] \quad . \end{aligned} \qquad (6.14)$$

Let the columns corresponding to the variables which are recomputed into the local bases $\hat{D}_{0B}(\tau)$, $t < \tau < t^*$, enter the matrix $\Phi_1(t)$ $(\hat{B}_{11}(t))$, $B_1^1(t)$), and the remaining columns enter the matrix $\Phi_2(t)$ $(\hat{B}_{12}(t)$, $B_2^1(t))$.

Then, in accordance with (6.5) and consequence 6.1, the matrix $B_1^1(t)$ does not change at the interchange of the ℓ-th column of the matrix $\hat{D}_{0B}(t)$ with the q-th column of the matrix $\hat{D}_{1B}(t)$. The matrix $B_2^1(t)$, which is defined from (6.14) is transformed in accordance with the formula

$$[B_2^1(t)]' = B_2^1(t)E_k \quad ,$$

where k is the number of the column of the matrix $\Phi_2(t)$ which contains the element $\Phi_{\ell q}(t)$. The order of matrix E_k is equal to the number of columns of matrix $B_2^1(t)$.

Let the k-th column of matrix $\Phi_2(t)$ correspond to the k-th component (6.9), (6.10), the columns of matrices

$$G(t+1)B_2^1(t) \text{ and } A(t+1)B_2^1(t)$$

do not enter the matrices

$$\hat{D}_{0B}(t+1) \text{ and } \hat{B}_{0B}(t+1) \quad .$$

Therefore the matrices $\hat{D}_{0B}(t+1)$, $\hat{B}_{0B}(t+1)$ do not change.

Let us partition the matrices $\Phi_B(t+1)$, $B_B^1(t+1)$ and $\hat{B}_{1B}(t+1)$ into two submatrices

$$\begin{aligned} \Phi_B(t+1) &= [\Phi_1(t+1);\ \Phi_2(t+1)] \quad ; \\ B_B^1(t+1) &= [B_1^1(t+1);\ B_2^1(t+1)] \quad ; \\ B_{1B}^1(t+1) &= [\hat{B}_{11}(t+1);\ \hat{B}_{12}(t+1)] \quad . \end{aligned}$$

The columns of the matrix $\Phi_B(t+1)$, which correspond to the same basic elements as the columns of the matrix $\Phi_2(t)$, enter the matrix $\Phi_2(t+1)$.

In accordance with the partitioning, the matrices $\Phi_1(t+1)$ and $\hat{B}_{11}(t+1)$ are not changed by the column permutations.

The matrices $\Phi_2(t+1)$ and $\hat{B}_{12}(t+1)$ are updated by formulas

$$\begin{aligned} [\Phi_2(t+1)]' &= \Phi_2(t+1)E_k \quad , \\ [\hat{B}_{12}(t+1)]' &= \hat{B}_{12}(t+1)E_k \quad . \end{aligned} \tag{6.15}$$

As

$$\hat{B}_2^1(t+1) = \hat{B}_{12}(t+1) - \hat{B}_{0B}(t+1)\Phi_2(t+1)$$

then, taking into account (6.15), we obtain

$$[B_2^1(t+1)]' = B_2^1(t+1)E_k \quad . \tag{6.16}$$

Similar reasoning is valid up to the step t^*. Thus, the interchange of the q-th column of the matrix $\hat{D}_{0B}(\tau)$ with k-th column of the matrix $\hat{D}_{0B}(t^*)$ causes changes neither in the local bases $\hat{D}_{0B}(\tau)$ nor in the matrices $\hat{B}_{0B}(\tau)$ $(\tau = t+1,\dots,t^* - 1)$; the matrices $\Phi_2(\tau)$ and $B_2^1(\tau)$ are updated by formulas (6.15), (6.16) if $t+1 = \tau(\tau = t+1, t+2,\dots,t^* - 1)$.

At step t^*, part of the columns of the matrix $G(t^*)B_2^1(t^* - 1)$ enters the matrix $\hat{D}_{0B}(t^*)$. Therefore, the updating of the matrices at this step reduces to the case considered above.

This procedure we shall call *the interchange of the ℓ-th column of the matrix $\hat{D}_{0B}(t)$ with the k-th column of the matrix $\hat{D}_{0B}(t^*)$, where $t^* > t+1$.*

The procedures of column permutation of the matrices $\hat{D}_{0B}(t)$ and $\hat{D}_{0B}(t^*)$ $(t^* > t+1)$ allow us to describe the updating procedure of the old local bases $\{\hat{D}_{0B}(t)\}$ into new ones $\{\hat{D}_{0B}(t)\}'$.

When a vector $\hat{d}_{0\ell}(t_2)$ is replaced by a vector $d_j(t_1)$, two cases are possible.

Case 1: $t_2 < t_1$

In this case, the ℓ-th row of the matrix $\Phi_B(t)$ contains a nonzero pivot element. In fact, the index of the outgoing variable is defined by the relation (3.8). Hence the ℓ-th component of the vector $\hat{v}_{0B}(t_2)$ is not zero.

From (2.8) and (3.7), we find that

$$\hat{v}_{0B}(t_2) = -\Phi_B(t_2)\hat{v}_{1B}(t_2) \text{ if } t_2 < t_1 \quad .$$

Therefore, the ℓ-th row of the matrix $\Phi_B(t_2)$ contains at least one non-zero element.

Let the pivot element correspond to the j-th component of vector $\hat{u}_{0B}(t_2 + \tau)$.

Replace the ℓ-th column of the matrix $\hat{D}_{0B}(t_2)$ by the j-th column of the matrix $\hat{D}_{0B}(t_2 + \tau)$. This interchange does not change the basic solution. Therefore, if $t_2 + \tau < t_1$, the above reasonings are true and we can proceed with the interchanges. In result, we obtain the following case.

Case 2: $t_2 \geq t_1$

Proceeding with these subsequent interchanges, we remove the outgoing vector into such a local basis $\hat{D}_{0B}(t_3)$, $t_3 \geq t_1$, which satisfies the condition of Theorem 6.1.

If such $t_3 \leq T-1$ does not exist, then we replace the outgoing column into the last local basis $\hat{D}_{0B}(T-1)$.

In turn, the outgoing column can be replaced in the local basis $\hat{D}_{0B}(t_3)$.

Let the outgoing vector be the ℓ-th column of the matrix $\hat{D}_{0B}(t_3)$. Before introducing the vector $d_j(t_1)$ into the basis, it is necessary to recompute it at the step t_3.

In result we obtain

$$\begin{aligned}
\hat{v}^*_{0B}(t_1) &= \hat{D}^{-1}_{0B}(t_1)d_j(t_1) \quad , \\
y^*(t_1+1) &= -b_j(t_1) + \hat{B}_{0B}(t_1)\hat{v}^*_{0B}(t_1) \\
\hat{v}^*_{0B}(\tau) &= -\hat{D}^{-1}_{0B}(\tau)G(\tau)y^*(\tau) \quad , \qquad\qquad (6.17) \\
y^*(\tau+1) &= A(\tau)y^*(\tau) + \hat{B}_{0B}(\tau)\hat{v}^*_{0b}(\tau) \quad , \\
\tau &= t_1+1,\ t_1+2, \ldots, t_3 \quad .
\end{aligned}$$

In these formulas, the new local bases $\{\hat{D}_{0B}(t)\}$ are used.

The above considered updating of the ingoing column $d_j(t_1)$ is possible as the ℓ-th (pivot) element of the vector $\hat{v}^*_{0B}(t_3)$ is not zero.

In fact, the ℓ-th element of the vector $\hat{v}^*_{0B}(t_3)$ is not zero, in accordance with (3.8) and the updating formulas (6.17) coincide with the formulas (3.6) and (3.7).

In accordance with (2.8) and (3.7)

$$\hat{v}_{0B}(t_3) = \hat{v}_{0B}(t_3) - \Phi_B(t_3)\hat{v}_{1B}(t_3) \quad .$$

But as the ℓ-th row of the matrix $\Phi_B(t_3)$ vanishes, $\hat{v}_{0\ell}(t_3) = \hat{v}^*_{0\ell}(t_3) \neq 0$. Thus a new set of local bases is obtained.

7. General Scheme of the Dynamic Simplex Method

Let at some iteration there be known: $\{\hat{D}_{0B}^{-1}(t)\}$, the inverse bases; $\{\hat{u}_{0B}(t)\}$, the basic feasible control; $\{x(t)\}$, the corresponding trajectory; $\{\lambda(t), p(t)\}$, the dual variables (simplex-multipliers).

As in the static simplex method, one can introduce artificial variables at zero iteration if necessary. In that case, the zero iteration local bases are the identity matrices.

In accordance with Sections 3 to 6, the general procedure of the dynamic simplex method comprises the following stages:

1. Choose some pair of indices (j,t_1), for which $\Delta_j(t_1) < 0$, $(j,t_1) \in I_N(u)$, where $\Delta_j(t_1)$ are defined from Section 5. Usually, a pair (j,t_1) with maximal absolute value of $\Delta_j(t_1)$ is selected. If all $\Delta_j(t_1) \geq 0$, $(j,t) \in I_N(u)$, then we have an optimal solution of the problem.

2. Define sequences of vector coefficients $\{v,y\}$ from (3.6) and (3.7).

3. Find the indices (ℓ,t_2) for the outgoing column from (3.8). If all $\hat{v}_{0i}(t) \leq 0$, then, the solution is unbounded.

4. Compute the new basic feasible control $\{u'(t)\}$:

$$u_i'(\tau) = \begin{cases} u_i(\tau) - \theta_0 \hat{v}_{iB}(\tau), & (i,\tau) \in I_B(u) \\ \theta & , (i,\tau) = (j,t_1) \\ 0 & , (i,\tau) \in I_N(u), \ (i,\tau) \neq (j,t_1) \end{cases} .$$

5. Update the local bases:

   a) set $t = t_2$

   b) if $t \geq t_1$, then go to stage e);

   c) select the non-zero element in the pivot row of the matrix $\Phi_B(t)$. (The index of the pivot row equals the index of the outgoing column).

d) let the pivot element of the matrix $\Phi_B(t)$ correspond to the component of the basic control, which was recomputed into the local basis at step $t+\tau$. Then

   - interchange the variables between local bases $\hat{D}_{0B}(t)$ and $\hat{D}_{0B}(t+\tau)$
   - set $t \to t+\tau$
   - go to stage b;

e) if $t = T-1$, then go to stage g;

f) if the pivot element of $\Phi_B(t)$ is nonzero, go to c;

g) replace the column to be removed by the column to be introduced into $\hat{D}_{0B}(t)$.

6. Compute the dual variables $\{\lambda, \dot{p}\}$ from (4.2). Go to stage 1.

It should be noted that only an outline of the algorithm is given here. The concrete implementation of the algorithm depends on the specifics of a problem, the type of computer, the strategy used as to which column selected and introduced into (or removed from) the set of local bases, etc.

8. Degeneracy

It was assumed above that all basic feasible controls were nondegenerate.

This assumption was necessary in order to guarantee that for each successive set of local feasible bases, the associated value of the objective function is larger than those that precede it. This guarantees that we will reach the optimal solution in a finite number of possible sets of local feasible bases.

For the degenerate case, there is the possibility of computing a θ_0 at step 3 of the method, for which $\theta_0 = 0$. Therefore, the selection of a vector to be removed from and a vector to be introduced into the set of local bases will give a new basic feasible control with the value of the objective function being equal to the preceding one. Theoretical examples have been constructed to show that in this case cycling of the procedure is possible. In practical examples this has never happened (with one possible exception). In order to protect against this possibility, a special rule for selecting the outgoing column can be introduced to prevent cycling in the case of degeneracy.

Here we can use the method of overcoming degeneracy of the simplex method [3]. For this we need the columns of the inverse $\bar{B}^{-1}$ (see (2.5)). The j-th column y_j of the inverse $\bar{B}^{-1}$ is a solution of the system of equations:

$$\bar{B}y_j = e_j \quad , \tag{8.1}$$

where e_j is the unit vector of dimension $(m+n)T$ with the j-th component equal to one.

The system (8.1) can be solved by using the factorized representation of the basis matrix, which is similar to the primal solution procedure (Section 3).

9. Numerical Example

Experimental results of tests with the algorithm and its numerical evaluation will be described in a separate paper. Here we consider an illustrative numerical example and give a theoretical evaluation (Section 10) of the method.

We consider the problem with scalar state equations and constraints (that is, $n = m = 1$). In this case, the dimension of the "global" basis matrix will be $2T \times 2T$, hence the corresponding static LP problem is not a very trivial one for large T. Using the dynamic simplex method, we do not need to invert the global basis; what is more, we do not need, for a considered example, to invert local bases either, because if $m = 1$, the local bases are simply numbers.

Problem: Given the state equations

$$x(t+1) = x(t) + u(t) - v(t) \quad (t = 0,\ldots,4) \tag{9.1}$$

with

$$x(0) = 0 \tag{9.2}$$

where $x(t)$, $u(t)$, $v(t)$ are scalars. Find $\{u(0),\dots,u(4)\}$, $\{v(0),\dots,v(4)\}$ and $\{x(0),\dots,x(5)\}$ which satisfy (9.1), (9.2) and constraints

$$x(t) + u(t) + v(t) = f(t) \tag{9.3}$$

$$u(t) \geq 0 \quad ; \quad v(t) \geq 0$$

where $f(0) = 10$; $f(1) = 5$; $f(2) = 5$; $f(3) = 10$; $f(4) = 5$ and minimize

$$J = 10x(5) \quad .$$

The tableau form of the problem is given below

u(0)	v(0)	x(1)	u(1)	v(1)	x(2)	u(2)	v(2)	x(3)	u(3)	v(3)	x(4)	u(4)	v(4)	x(5)	
1	1														= 10
1	-1	-1													= 0
		1	1	1											= 5
		1	1	-1	-1										= 0
					1	1	1								= 5
					1	1	-1	-1							= 0
								1	1	1					= 10
								1	1	-1	-1				= 0
											1	1	1		= 5
											1	1	-1	-1	= 0

Thus, if we solve the problem by the standard simplex method, it is necessary to handle with 10 × 10 "global" basis at each iteration.

Now we proceed by the dynamic algorithm. Let $\{u^{(0)}(0), v^{(0)}(0), x^{(0)}(1), x^{(0)}(2), u^{(0)}(2), x^{(0)}(3), u^{(0)}(3), v^{(0)}(3), x^{(0)}(4), v^{(0)}(4)\}$ be the first basic variables. The corresponding local bases $\hat{D}_{0B}(t)$ $(t = 0,\dots,4)$ are the following:

$$\begin{array}{llll}
\hat{D}_{0B}^{(0)}(0) = 1 \ ; & \hat{u}_{0B}^{(0)}(0) = u(0) \\
\hat{D}_{0B}^{(0)}(1) = -2 \ ; & \hat{u}_{0B}^{(0)}(1) = v(0) \\
\hat{D}_{0B}^{(0)}(2) = 1 \ ; & \hat{u}_{0B}^{(0)}(2) = u(2) \\
\hat{D}_{0B}^{(0)}(3) = 1 \ ; & \hat{u}_{0B}^{(0)}(3) = u(3) \\
\hat{D}_{0B}^{(0)}(4) = -2 \ ; & \hat{u}_{0B}^{(0)}(4) = v(3) \\
\hat{D}_{0B}^{(0)}(5) = -2 \ ; & \hat{u}_{0B}^{(0)}(5) = v(4) \quad .
\end{array}$$

Note that control variable v(0) from step t = 0 enters the local basis $\hat{D}_{0B}^{(0)}(1)$ at the next step t = 1. As variable x(5) does not enter the "global" basis on this iteration, it is necessary to introduce an additional local basis $\hat{D}_{0B}^{(0)}(5)$ which corresponds to variable v(4).

The corresponding set of $\Phi_B(t)$ and $B_B^1(t)$ $(t = 0,..,4)$ are the following:

$$\begin{array}{ll}
\Phi_B^{(0)}(0) = 1 \ ; & B_B^{1(0)}(0) = -2 \\
\Phi_B^{(0)}(3) = 1 \ ; & B_B^{1(0)}(3) = -2 \\
\Phi_B^{(0)}(4) = -0.5 \ ; &
\end{array} \qquad (9.6)$$

$\Phi_B^{(0)}(1)$; $\Phi_B^{(0)}(2)$, $B_B^{1(0)}(1)$, $B_B^{1(0)}(2)$, $B_B^{1(0)}(4)$ are zeros.

Using (3.4) and (3.5) for (9.1), (9.2) and (9.5), we obtain the first primal solution:

$$\begin{array}{lllll}
u^{(0)}(0) = 7.5 & & u^{(0)}(2) = 0 & u^{(0)}(3) = 2.5 & \\
v^{(0)}(0) = 2.5 & & & v^{(0)}(3) = 2.5 & v^{(0)}(4) = 5 \\
x^{(0)}(1) = 5 & x^{(0)}(2) = 5 & x^{(0)}(3) = 5 & x^{(0)}(4) = 5 &
\end{array} \qquad (9.7)$$

the value of objective function: $x^{(0)}(5) = 0$.

As coefficients of the objective function for basic variables are zeros, then all simplex-multipliers (according to (4.2)) are also zeros. Therefore, we have all Δ_j are zeros but $\Delta^{(0)}(x(5)) = -10$. Hence, x(5) is to be introduced to the basis.

Denoting coefficients $\hat{v}_{0B}(t)$ for variables u(t), v(t) and x(t) as $\alpha(t)$, $\beta(t)$ and $\gamma(t)$ respectively, we calculate using (3.6) and (3.7), that $\alpha^{(0)}(3) = -0.25$; $\beta^{(0)}(3) = 0.5$; $\beta^{(0)}(4) = 0.5$; $\gamma^{(0)}(4) = -0.25$ the other $\alpha^{(0)}(t)$, $\beta^{(0)}(t)$ and $\gamma^{(0)}(t)$ are zeros.

From (3.8)

$$\theta_0^{(0)} = \min \left\{ \frac{2.5}{-0.25} ; \frac{2.5}{0.25} ; \frac{5}{0.5} \right\} = -10$$

(it should be taken into account that {x(t)} are free variables). Thus, $x^{(1)}(5) = \theta_0^{(0)} = -10$ and u(3) is to be removed from the basis.

The new primal solution will be the following

$u^{(1)}(0) = 7.5$;		$u^{(1)}(2) = 0$		
$v^{(1)}(0) = 2.5$;			$v^{(1)}(3) = 5$	$v^{(1)}(4) = 10$
$x^{(1)}(1) = 5$;	$x^{(1)}(2) = 5$;	$x^{(1)}(3) = 5$;	$x^{(1)}(4) = 0$	$x^{(1)}(5) = -10$

Now old local bases (9.5) are updated. As variable u(3) leaves the basis, we have to interchange variables u(3) and v(3). After interchange: $\hat{D}_{0B}(3) = 1$, $\Phi_B(3) = 1$; $B_B^1(3) = 2$; $\hat{D}_{0B}(4) = 2$; $\Phi_B(4) = 0.5$.

Then u(3) and v(4) should be interchanged. Hence $\hat{D}_{0B}(4) = 1$; $\Phi_B(4) = 2$; $\hat{D}_{0B}(5) = 4$. Finally, we replace u(3) by x(5), then $\hat{D}_{0B}(5) = -1$.

Thus, the updated local bases are

$$\begin{array}{ll} \hat{D}_{0B}^{(1)}(0) = 1 & \hat{D}_{0B}^{(1)}(3) = 1 \\ \hat{D}_{0B}^{(1)}(1) = -2 & \hat{D}_{0B}^{(1)}(4) = 1 \\ \hat{D}_{0B}^{(1)}(2) = 1 & \hat{D}_{0B}^{(1)}(5) = 4 \quad . \end{array} \tag{9.8}$$

We can begin new iterations now. Using (4.2), the dual solution is obtained for local bases (9.8):

$$\begin{array}{lll} p^{(1)}(5) = 10 & p^{(1)}(3) = 40 & p^{(1)}(1) = 0 \\ \lambda^{(1)}(4) = -10 & \lambda^{(1)}(2) = 40 & \lambda^{(1)}(0) = 0 \\ p^{(1)}(4) = 20 & p^{(1)}(2) = 0 & \\ \lambda^{(1)}(3) = -20 & \lambda^{(1)}(1) = 0 & \end{array} \qquad (9.9)$$

From (9.9) and (5.1), $\Delta^{(1)}(u(4)) = -20$; $\Delta^{(1)}(u(3)) = -60$; $\Delta^{(1)}(v(2)) = 80$, the other $\Delta^{(1)}$ are zeros. Hence, variable $v(2)$ should be introduced into local bases. Calculating θ_0 for this iteration, we find that $\theta_0^{(1)} = 0$ and $u(2)$ should be removed from the bases. As $\Phi_B^{(1)}(2) = 0$ and variables $u(2)$ and $v(2)$ are from the same step $t = 2$, only local basis $\hat{D}_{0B}(2)$ at this step $t = 2$ must be updated. In result, $\hat{D}_{0B}(2) = 1$ and the other local bases have the same values as in (9.8). The new iteration yields

$$\begin{array}{lll} p^{(2)}(5) = 10 & p^{(2)}(3) = 40 & p^{(2)}(1) = 0 \\ \lambda^{(2)}(4) = -10 & \lambda^{(2)}(2) = -40 & \lambda^{(2)}(0) = 0 \\ p^{(2)}(4) = 20 & p^{(2)}(2) = 80 & \\ \lambda^{(2)}(3) = -20 & \lambda^{(2)}(1) = 80 & \end{array} \qquad (9.10)$$

and $\Delta^{(2)}(u(4)) = -20$; $\Delta^{(2)}(u(2)) = -80$; $\Delta^{(2)}(v(1)) = 160$; $\Delta^{(2)}(u(3)) = -40$; $\Delta^{(2)}(u(1)) = 0$.

Hence $v(1)$ is introduced to the local bases, $\theta_0^{(2)} = 15$ and $u(0)$ is removed from the local bases. At this iteration, the local bases $\hat{D}_{0B}(0)$ and $\hat{D}_{0B}(1)$ are updated. In result, we obtain

$$\begin{array}{lll} v^{(3)}(0) = 10 & v^{(3)}(1) = 15 & v^{(3)}(2) = 30 \\ x^{(3)}(1) = -10 & x^{(3)}(2) = -25 & x^{(3)}(3) = -55 \\ v^{(3)}(3) = 65 & v^{(3)}(4) = 130 & \\ x^{(3)}(4) = -120 & x^{(3)}(5) = -250 & \end{array} \qquad (9.11)$$

and $p^{(3)}(2) = 80$; $p^{(3)}(1) = 160$; $\lambda^{(3)}(1) = -80$; $\lambda^{(3)}(0) = -160$, the other $p^{(3)}(t)$ and $\lambda^{(3)}(t)$ are the same as in (9.10). All values of $\Delta^{(3)}(\cdot)$ are negative now. Therefore, (9.11) is an optimal solution.

10. Evaluation of Algorithm

Above we considered an illustrative numerical example which is not so easy to solve by hand using the conventional "static" simplex method, but is very simple to handle by the dynamic algorithm.

Now we give some theoretical evaluation of the dynamic simplex method.

As can be seen from Section 7, for realization of the algorithm it is sufficient to operate only with the matrices $\hat{D}_{0B}^{-1}(t)$; $\Phi_B(t)$, $\hat{B}_{0B}(t)$, $B_B^1(t)$, $G(t)$, $A(t)$ $(t = 0,1,\dots,T-1)$.

Theorem 10.1: The number of columns of matrices $\Phi_B(t)$ *and* $B_B^1(t)$ *does not exceed* n.

Proof: Let 2T steps of the factorization process be carried out.

Then the formula (2.7) can be rewritten as

$$\bar{B} = \bar{B}_{2t-1}V_{t-1}U_{t-1} \cdot \cdot \cdot V_0U_0 \quad .$$

On the main diagonal of the matrix $\bar{B}_{2t-1}$ there is the submatrix

$$F = \begin{bmatrix} \hat{D}_{0B}(t) & \hat{D}_{1B}(t) \\ \hat{B}_{0B}(t) & \hat{B}_{1B}(t) \end{bmatrix} \quad .$$

The columns of the submatrix F are linearly independent as the matrix B_{2t-1} is nonsingular. Consequently, the number of columns of matrices $\hat{D}_{1B}(t)$ and $\hat{B}_{1B}(t)$ cannot be larger than n. Hence, one can obtain the statement of the theorem.

The matrices $\hat{D}_{0B}^{-1}(t)$, $\hat{B}_{0B}(t)$, $G(t)$, $A(t)$ have dimensions $(m \times m)$, $(n \times m)$, $(m \times n)$, $(n \times n)$ respectively. Therefore, the algorithm operates only with the set of T matrices, each containing no more than m or n columns.

At the same time, the straightforward application of the simplex method to Problem 1.1 (in the space of $\{u,x\}$) leads to the necessity of operating with the basis matrix of dimension $(m+n)T \times (m+n)T$ or of dimension $mT \times mT$, if the state variables are excluded beforehand.

Thus, in some respects, the dynamic simplex method realizes a decomposition of the problem that allows a substantial saving inthe number of arithmetical operations and in the core memory.

As was mentioned above, the DLP Problem 1.1 can be considered as some "large" static LP problem and thus the simplex method can be used for its solution. Let us find an upper estimation of a number of iterations. At each iteration, the simplex method requires no more than k^2 multiplications for updating of the inverse, where k is the number of rows of the basic matrix. Hence, the total number of multiplications for the basis updating is no more than $(m+n)^2T^2$. To compute the coefficients which express the column to be introduced into a basis in terms of columns of the current basis, the simplex method requires some $(m+n)^2T$ multiplications.

Now we shall evaluate the number of multiplications for the dynamic simplex method. It was shown that at one interchange, the local bases are updated by multiplication on the elementary column or row matrix. The interchange of columns between two neighboring local bases $\hat{D}_{0B}(t)$ and $\hat{D}_{0B}(t+1)$ requires no more than $3(m+n)^2$ multiplications. (The matrices $\hat{D}_{0B}^{-1}(t)$, $\hat{B}_{0B}(t)$, $\Phi_B(t)$, $B_B^1(t)$, $\hat{D}_{0B}^{-1}(t+1)$, $\hat{B}_{0B}(t+1)$, $\Phi_B(t+1)$ are updated). In the worst case, when the outgoing column from the local bases $\hat{D}_{0B}(0)$ is entered into the local basis $\hat{D}_{0B}(T-1)$, one needs T interchanges. We assume that the average number of interchanges is T/2. Thus the dynamic simplex method requires approximately $1.5(m+n)^2T$ multiplications for local bases updating per iteration.

Calculation of the coefficients expressing the ingoing vector requires about $(m+n)^2T$ multiplications. In addition, local bases can be represented in factorized form, thus enabling use of the effective procedures of static LP [3].

Solution of Problem 1.1 by the static simplex method requires storage of the inverse of dimension $(m+n)T \times (m+n)T$. The dynamic simplex method requires storage of only T matrices of dimension $m \times m$ $(\hat{D}_{0B}^{-1}(t), \hat{B}_{0B}(t))$ and $T-1$ matrices of dimension $m \times n$ $(\phi_B(t))$ and $n \times n$ $(B_B^1(t))$.

Thus, comparing the estimates of the static and dynamic algorithms for solution of Problem 1.1, one can see that the volume of computation and the core memory increases linearly with T for the dynamic algorithm and by quadratic law for the static algorithm.

It is more important that only part of the local bases be updated at each iteration. Therefore the dynamic simplex method may turn out to be superior in comparison with a conventional revised simplex algorithm not only because it offers a more compact substitute for the basic inverse but also because it allows the use of only a part of the basic inverse representation required at each iteration.

11. Dual Algorithms

The introduction of local bases and techniques of their handling allows us to develop dual and primal-dual versions of the dynamic simplex method. The main advantage of using the dual methods is that the dual statements of many problems have explicit solutions. The other is connected with the choice of different selection strategies to the vector pair which enters and leaves the basis.

In the primal version of the dynamic simplex method, there are some options for choice of a column with the most negative price from all non-basic columns or from some set of these columns, etc. But a column to be removed from the basis is unique in the nondegenerate case.

Contrarily, in dual methods, there are options in the choice of a column to be removed from the basis. It can be effectively used in dual versions of the method. In practical problems, local bases $\{\hat{D}_{0B}(t)\}$ can be rather large, therefore part of the local bases should be stored at the external storage capacities. Input-output operations are comparatively time-consuming. Hence, to reduce the total solution time, it is desirable to have more pivoting operations with a given local basis.

Thus, the usage of different dual and primal-dual strategies allows us to adjust the algorithm to the specifics of the computer to be used and to the problem to be solved.

12. Extensions

The approach considered above is flexible and allows different extensions and generalizations. Below, we describe briefly two of them.

First, in Problem 1.1, the state variables x(t) are considered to be free. The case when $x(t) \geq 0$ or $0 \leq x(t) \leq \alpha(t)$ can be treated by the approach very easily. In fact, from the point of view of the computer implementation of the algorithm, it is better to handle with the multiplicative form of the inverse of

$$\tilde{D}_{0B}(t) = \begin{pmatrix} \hat{D}_{0B}(t) & 0 \\ \hat{B}_{0B}(t) & -I \end{pmatrix}$$

rather than with $\hat{D}_{0B}^{-1}(t)$, because the addition of the unit matrix -I does not generate additional zeros in the "eta-file". If x(t) are not constrained, then by handling with the inverse of $\tilde{D}_{0B}(t)$ we can consider the rows corresponding to low blocks of $\tilde{D}_{0B}(t)$, that is, $\hat{B}_{0B}(t)$ and -I, as free. In this case, all x(t) are in the basis.

If $x(t) \geq 0$, then the state variables x(t) should be handled in the same way as control variables $u(t) \geq 0$. In this case, not all x(t) will be in the basis.

Evidently, this includes the case when both state and control variables have upper bound constraints. (The inclusion of generalized upper bound constraints is also possible).

The second case, which has many important applications, is DLP with time delays. Instead of (1.1) and (1.3), we have in this case

$$\begin{aligned} x(t+1) &= \sum_{\nu} A(t,\tau_{\nu})x(t-\tau_{\nu}) + \sum_{\mu} B(t,\tau_{\mu})u(t-\tau_{\mu}) \\ &\sum_{\nu} G(t,\tau_{\nu})x(t-\tau_{\nu}) + \sum_{\mu} D(t,\tau_{\mu})u(t-\tau_{\mu}) \qquad (12.1) \\ &= f(t) \end{aligned}$$

with given values for $x(t)$ and $u(t-1)$ if $t \leq 0$. Here $\{\tau_{\nu}\}$, $\{\tau_{\mu}\}$ are given ordered sets of integers.

New submatrices will appear to the left from the main staircase of the diagonal of B* in (2.7a) (see Figure 1a and b).

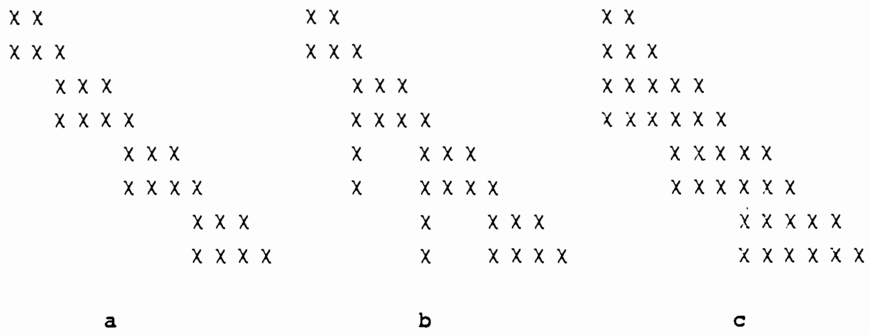


Figure 1.

Because the main staircase structure is not changed in this case (Figure 1), we can use the same procedure as in the case without time delays. There will be only one difference. Now local bases $\hat{D}_{0B}(t)$ will contain recomputed columns both from previous steps $\tau < t$ and columns from time "delayed" matrices $D(t,\tau)$ $\tau \leq t$, which enter the constraints (12.1) at step t.

Thus, both of these important extensions of Problem 1.1 can be handled by the algorithm almost without any modifications.

The extensions considered above concern the extension of Problem 1.1 within the DLP framework. It should be underlined that the approach is also applicable to solve LP problems with general structure (such as in Figure 1, if by χ one means some arbitrary matrix).

In this case, the approach will be related to factorization methods considered in [4,5].

13. Conclusion

The general scheme and basic theoretical properties of the dynamic simplex method specially developed for solution of dynamic linear programs have been described and discussed.

Theoretical reasonings show that this algorithm may serve as a base for developing effective computer codes for the solution of DLP problems. However, the final judgment of the efficiency of the algorithm can be made only after a definite period of its exploitation in practice.

It should also be very interesting to compare (both from the theoretical and the computational point of view) the approach given in this paper with the finite-step DLP algorithms based on the Dantzig-Wolfe decomposition principle [6,7,8] and other methods of solving structured LP problems [4-9].

References

[1] Krivonozhko, V.E. and A.I. Propoi, *II. The Dynamic Simplex Method: General Approach*, this issue.

[2] Gantmacher, F.R., *The Theory of Matrices*, Chelsea Publishing Company, New York, 1960.

[3] Dantzig, G.B., *Linear Programming and Extensions*, University Press, Princeton, N.J., 1963.

[4] Bulavskii, V.A., R.A. Zvjagina and M.A. Yakovleva, Numerical Methods of Linear Programming (Special Problems) *Nauka*, 1977, (in Russian).

[5] Kallio, M. and E. Porteus, Triangular Factorization and Generalized Upper Bounding Techniques, *Operations Research*, 25, 1 (1977).

[6] Ho, J.K. and A.S. Manne, Nested Decomposition for Dynamic Models, *Mathematical Programming*, 6, 2 (1974).

[7] Krivonozkho, V.E., Decomposition Algorithm for Dynamic Linear Programming Problems, Izvestia AN SSSR, *Technicheskya Kibernetika*, 3, (1977), 18-25, (in Russian).

[8] Beer, K., Lösung großer linearer Optimierungsaufgaben, VEB Deutscher Verlag der Wissenschaften, Berlin, 1977, (in German).

[9] Wollmer, R.D., A Substitute Inverse for the Basis of a Staircase Structure Linear Program, *Math. of Operations Research*, 2, 3 (1977).

DECOMPOSITION ALGORITHMS

A NESTED DECOMPOSITION APPROACH FOR SOLVING STAIRCASE-STRUCTURED LINEAR PROGRAMS

Phillip Abrahamson[*]

Systems Optimization Laboratory
Department of Operations Research
Stanford University

The algorithm solves a T-period staircase-structured linear program by applying a compact basis-inverse scheme for the simplex method in conjunction with a choice mechanism which uses the dual of the Nested Decomposition Principle of Manne and Ho to determine the incoming basic column. A sequence of one-period problems is solved in which, typically, information is provided to period t from previous and subsequent periods in the form of surrogate columns and modified right-hand side, and surrogate rows and modified cost coefficients, respectively.

[*] Currently at:
Department of Industrial Engineering and Operations Research
University of Massachusetts

I. Introduction

This paper describes preliminary work on an algorithm for solving staircase-structured linear programs. Such problems often arise in the modeling of phenomena which are naturally described as evolving over a sequence of temporal or spatial "periods". A pure nested decomposition algorithm transforms the problem into an ordered set of smaller problems, one for each period, which are coordinated only through price and activity communication between adjacent periods. The process of achieving an optimal coordination involves repeated solution of the individual problems. Preliminary experience has indicated that the convergence of the pure algorithm may be slow. To accelerate this convergence, the algorithm is modified to enable the individual problems to communicate in an implicit fashion whenever possible. This modification involves the generation of surrogate columns which are passed to subsequent periods, allowing these period to parametrically adjust solutions to earlier periods. A compact basis-inverse scheme is used to represent these parametric variations.

Section 2 states the problem of interest and describes the pure nested decomposition algorithm. Section 3 outlines the modified approach and discusses some details of implementation.

2. Nested Decomposition of the Staircase Structure

2.1 The Staircase Structure

The problem of interest is

$$\text{minimize} \quad \sum_{t=1}^{T} c_t x_t$$

$$\begin{aligned} \text{subject to} \quad & A_1 x_1 && = b_1 \\ & -B_{t-1} x_{t-1} + A_t x_t && = b_t , \qquad t = 2, \ldots, T \\ & \qquad\qquad x_t \geq 0 && \qquad , \qquad t = 1, \ldots, T \end{aligned}$$

where x_t is $n_t \times 1$, A_t is $m_t \times n_t$, and all other vectors and matrices are of conformable dimension. This linear program is said to be staircase-structured because the constraint matrix has the form:

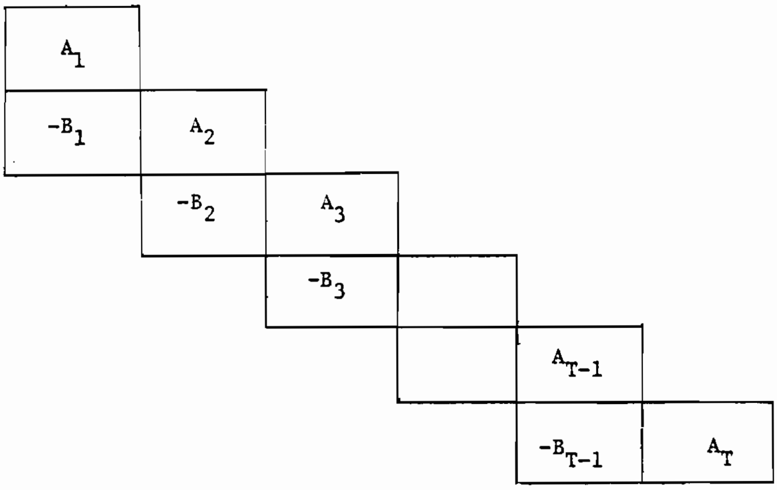


Figure 1

Activities in period t are represented by the matrix A_t . The inventory provided by period t to period $t + 1$ is described by the matrix $-B_t$. Otherwise, the activities in period t have no direct effect on either previous or subsequent periods.

The dual of (P) is

$$\text{(D)} \qquad \text{maximize} \quad \sum_{t=1}^{T} \pi_t b_t$$

$$\text{subject to} \quad \pi_t A_t - \pi_{t+1} B_t \leq c_t , \qquad t = 1, \ldots, T - 1$$

$$\pi_T A_T \leq c_T .$$

2.2 Nested Decomposition

Manne and Ho [7] and Glassey [5] repeatedly applied the decomposition principle of linear programming developed by Dantzig and Wolfe [3] to achieve a nested decomposition of (P) . A sequence of applications, modifications, and improvements has led to advanced implementations by Ho and Loute [6], who have solved some large-scale problems more rapidly in this fashion than by directly applying commercial linear programming to (P) .

Van Slyke and Wets [8] describe an algorithm for solving (P) in the case $T = 2$ which is equivalent to applying the decomposition algorithm to (D) . Dantzig [2] outlined an algorithm consisting of a nested decomposition of (D) . This paper represents preliminary work on the development of a technique which is based on his approach.

2.3 A Nested Decomposition Algorithm for (D)

The algorithm of Dantzig [2] consists of the following nested decomposition:

$$\begin{array}{lllll}
\text{maximize} & \pi_1 b_1 + \pi_2 b_2 + \pi_3 b_3 + \pi_4 b_4 + \cdots + \pi_T b_T & & \text{master}_1 & \\
\text{subject to} & \pi_1 A_1 - \pi_2 B_1 & \leq c_1 & & \\
 & \quad \pi_2 A_2 - \pi_3 B_2 & \leq c_2 & & \text{master}_2 \\
 & \qquad \pi_3 A_3 - \pi_4 B_3 & \leq c_3 & & \\
 & \qquad\quad \cdot & & & \\
 & \qquad\qquad \cdot & & \text{sub}_1 & \cdot \\
 & \qquad\qquad\quad \cdot & & & \text{sub}_2 \quad \cdot \\
 & \qquad\qquad\qquad \pi_{T-1} A_{T-1} - \pi_T B_{T-1} & \leq c_{T-1} & & \qquad \text{master}_{T-1} \\
 & \qquad\qquad\qquad\qquad \pi_T A_T & \leq c_T & & \qquad \text{sub}_{T-1}
\end{array}$$

Figure 2

Let x_{t-1}^0 be multipliers for master_{t-1} , and let

$$\bar{b}_t = b_t + B_{t-1} x_{t-1}^0 \quad . \tag{1}$$

Then the t-th period master problem has the form

$$\begin{array}{llll}
(D_t) & \text{maximize} & \pi_t \bar{b}_t + \sigma_t f_t + \rho_t \ell_t & \\
 & \text{subject to} & \pi_t A_t + \sigma_t F_t + \rho_t L_t \leq c_t & : x_t \\
 & & \rho_t e \leq 1 & : \theta_t . \\
 & & \sigma_t , \ \rho_t \geq 0 & ,
\end{array}$$

where (F_t, f_t) and (L_t, ℓ_t) are, respectively, the extreme ray and extreme point proposals generated by period $t + 1$. For $t = 1$, $\bar{b}_1 = b_1$, and for $t = T$, only the terms involving π_T are present. Note that each proposal is a row vector.

The dual of (D_t) is

$$
\begin{array}{llll}
(P_t) & \text{minimize} & z_t = c_t x_t + \theta_t & \\
& \text{subject to} & A_t x_t \qquad = b_t + B_{t-1} x^0_{t-1} & : \pi_t \\
& & F_t x_t \qquad \geq f_t & : \sigma_t \\
& & L_t x_t + e\theta_t \geq \ell_t & : \rho_t \\
& & x_t \geq 0 \ . &
\end{array}
$$

In these problems, $\bar{b}_t$ represents the original term b_t from (P) as modified by the inventory $B_{t-1} x^0_{t-1}$ supplied by the current period t - 1 solution x^0_{t-1}. The dependence of $\bar{b}_t$ on x^0_{t-1} will usually be suppressed for clarity. The constraints $F_t x_t \geq f_t$ and $L_t x_t + e\theta_t \geq \ell_t$ are necessary conditions for a solution x_t to not lead to future infeasibilities or non-optimal solutions. Accordingly, they are called feasibility and look-ahead (optimality) cuts, respectively. The constraints $A_t x_t = b_t + B^0_{t-1} x^0_{t-1}$ are called the body of the constraints.

The full master problem (D_1), which includes all possible proposals from the future, is equivalent to (P). Computationally, a restricted master is maintained at each period by including proposals as they are generated and occasionally purging old proposals which are no longer basic in (D_t). It will be clear from context which sets of proposals are indicated by (F_t, f_t) and (L_t, ℓ_t).

While nested decomposition of (D) produces the problems (D_t) , it is usually more instructive and more convenient computationally to work with their respective duals (P_t) . In general, the problem at period t acts as a subproblem with respect to periods 1, ..., t - 1 , and as a restricted master problem with respect to periods t + 1, ..., T . The communication between the problems (P_t) can be represented schematically:

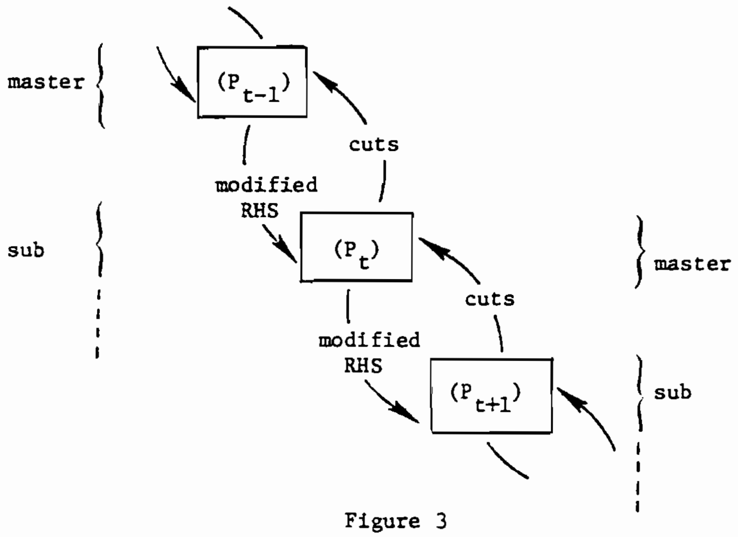


Figure 3

Note that the form of the look-ahead cuts is such that they essentially modify the objective function in each (P_t) .

The act of solving one of the restricted master problems (P_t) is called a step. If, at some step, (P_t) is infeasible, a feasibility cut is generated and imposed on (P_{t-1}) . If a step terminates with a finite optimal solution to (P_t) , a modified right-hand side for (P_{t+1}) is generated. If in the latter case it is also found that

$$z_t^* > \theta_{t-1}^0 , \tag{2}$$

where z_t^* is the new optimal objective for (P_t) and θ_{t-1}^0 corresponds to the most recent solution to (P_{t-1}) , a look-ahead cut is generated which may be imposed on (P_{t-1}) . If (P_t) is found to have a class of solutions with objective unbounded below, a solution x_t^0 and a homogeneous solution h_t^0 are generated in the usual fashion. The desired effect of providing $(P_{t\ 1})$ with a right-hand side of the form

$$b_{t+1} + B_t x_t^0 + \alpha B_t h_t^0 \quad , \quad \alpha \geq 0 \tag{3}$$

is achieved by introducing into (P_{t+1}) a surrogate activity, with level $\alpha \geq 0$, represented by the column

$$-B_t h_t^0 \tag{4}$$

with cost coefficient $c_t h_t^0$. If (P) has a class of solutions with objective unbounded below, eventually a ray indicating this will be generated in (P_T) . Otherwise, a look-ahead cut will be generated in $(P_{t'})$, for some $t' > t$, which "cuts off" the ray successively in $(P_{t'-1})$, ..., (P_t) .

A wide variety of computational strategies may be employed within the framework described above. There is freedom both in the order in which the problems (P_t) are solved and in when to pass information between problems in the form of cuts and modified right-hand sides. Computational experience has indicated that the rate of convergence of the algorithm can vary significantly when different strategies are employed. This experience has also indicated that in order to attain the ease of solution initially envisioned for this approach, signficant modifications to the algorithm described are necessary.

3. A Modified Nested Decomposition Approach

3.1 Passing Surrogate Columns Forward

Suppose, at some step in the course of executing the algorithm described in Section 2.3, a finite optimal solution to (P_t) is obtained. The optimal basis must include θ_t and, assuming A_t is of full rank, at least m_t of the variables x_t. Some slack variables corresponding to cuts which have been imposed may also be basic. Let $k_1 \geq 0$ and $k_2 \geq 1$ be the number of feasibility and look-ahead cuts, respectively, whose slack variables are not basic. Then $m_t + k_1 + k_2 - 1$ of the variables x_t are in the optimal basis. Ordinarily, the optimal solution x_t^0 is used to form a right-hand side

$$b_{t+1} + B_t x_t^0 \tag{5}$$

for the body of the constraints in (P_{t+1}).

A modification to this technique is outlined as follows. Let

$$k = k_1 + k_2 - 1 \tag{6}$$

be the number of "surplus" variables, and let β_t^0 be the optimal basis, excluding slacks from cuts. Partition

$$\beta_t^0 = LB_t \cup NB_t \cup \{\theta_t\} \quad , \tag{7}$$

where $|LB_t| = m_t$, $|NB_t| = k$, and $A_{LB_t}^{-1}$ exists, where A_{LB_t} is a convenient abuse of the proper notation $(A_t)_{\cdot(LB_t)}$. The variables x_{LB_t} are called the local basis. Solve the body of the constraints in (P_t) versus the right-hand side $\bar{b}_t$ to yield the locally basic solution

$$(x_{LB_t})^0 = A_{LB_t}^{-1} \bar{b}_t \quad . \tag{8}$$

Represent the remaining basic variables in terms of the local basis,

$$Y_{NB_t} = A_{LB_t}^{-1} A_{NB_t} \quad , \tag{9}$$

so that

$$x_t^0 = \begin{pmatrix} x_{LB_t}^0 \\ x_{NB_t}^0 \\ 0 \end{pmatrix} = \begin{pmatrix} (x_{LB_t})^0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -Y_{NB_t} \\ I \\ 0 \end{pmatrix} x_{NB_t}^0 \quad . \tag{10}$$

Let

$$H_t = \begin{pmatrix} -Y_{NB_t} \\ I \\ 0 \end{pmatrix} \tag{11}$$

be the set of homogeneous solutions to the body of the constraints which are generated by the representations Y_{NB_t} . In contrast to (5), use the locally basic solution $(x_{LB_t})^0$ to form a right-hand side

$$b_{t+1} + B_t(x_{LB_t})^0 \tag{12}$$

for (P_{t+1}) . In addition, introduce into (P_{t+1}) a set of k surrogate activities, with levels $\alpha_{t+1} \geq 0$, represented by columns with

$$S_{t+1} = -B_t H_t \tag{13}$$

in the body of the constraints, zero coefficients in the cuts on (P_{t+1}), and cost coefficients $s_{t+1} = c_t H_t$. The modified problem is

$(\tilde{P}_{t+1})$

$$\begin{array}{llll}
\text{minimize} & \tilde{z}_{t+1} = s_{t+1}\alpha_{t+1} + c_{t+1}x_{t+1} + \theta_{t+1} & & \\
\text{subject to} & S_{t+1}\alpha_{t+1} + A_{t+1}x_{t+1} & = \bar{b}_{t+1} & :\pi_{t+1} \\
& F_{t+1}x_{t+1} & \geq f_{t+1} & :\sigma_{t+1} \\
& L_{t+1}x_{t+1} + \theta_{t+1} & \geq \ell_{t+1} & :\rho_{t+1}
\end{array}$$

$$\alpha_{t+1},\ x_{t+1} \geq 0 \quad .$$

This structure, in essence, endows $(\tilde{P}_{t+1})$ with a right-hand side which is parametrized in terms of the k variables x_{NB_t} , and allows $(\tilde{P}_{t+1})$ to select the coefficients α_{t+1} of the parametrization. Setting $\alpha_{t+1} = x^0_{NB_t}$ corresponds to the right-hand side (5) obtained in the pure nested decomposition framework. Passing activities down the staircase and allowing α_{t+1} to vary from these values provides an avenue of implicit communication between $(\tilde{P}_{t+1})$ and $(\tilde{P}_t)$, ..., $(\tilde{P}_1)$ which decreases the number of steps needed to obtain an optimal coordination of the single-period problems.

There are two potential disadvantages inherent in this approach. First, it seems that forcing $(\tilde{P}_{t+1})$ to have some or all of the variables α_{t+1} in its basis would unacceptably limit the number of basic variables chosen from x_{t+1} . Second, the framework sketched above allows $(\tilde{P}_{t+1})$ to use the surrogate activities at any nonnegative levels. In order to satisfy the body of the constraints in $(\tilde{P}_t)$, the relationship

$$x_{LB_t}(\alpha_{t+1}) = (x_{LB_t})^0 - (Y_{NB_t})\alpha_{t+1} \tag{14}$$

must be maintained. Since $(\tilde{P}_t)$ may include surrogate activities inherited from $(\tilde{P}_{t-1})$, in general a parametrization of all variables which are locally basic in $(\tilde{P}_t)$, ..., $(\tilde{P}_1)$ is obtained in terms of the surrogate activities received by $(\tilde{P}_{t+1})$. Clearly, nonnegative levels of α_{t+1} could cause the parametrized values of variables in earlier periods to have negative components. These two points are addressed in the next sections.

3.2 Degeneracy in Optimal Solutions Generated by Nested Decomposition of (D)

It is well-known that bases for (P) inherit the staircase structure of the general problem and exhibit a "surplus-shortage" property which generalizes the fact that a basis which contains $m_1 + k$ of the variables x_1, with $k > 0$, must exhibit a "shortage" of k, relative to the number of remaining constraints, $\sum_{t=2}^{T} m_t$, in the number of basic variables selected from $x_2, \ldots, x_T$. Fourer [4] has developed a set of bounds on the magnitudes of the surpluses and shortages which each period of a basis may possess.

The following result describes a manifestation of this property in the setting of the algorithm developed in Section 2.3.

Theorem. Suppose an optimal coordination of the single-period problems (P_t) has been obtained. If $t < T$ and (P_t) has $k + 1$ cuts whose slack variables are nonbasic, then the basic solution to (P_{t+1}) has at least k degenerate variables.

Under the modification outlined in Section 3.1, surrogate columns passed from period t replace degenerate variables in the optimal basis in period $t + 1$. These degenerate variables may include basic slack variables for the cuts in period $t + 1$. The fact that the surrogate columns replace degenerate variables guarantees that no solutions of interest are excluded in $(\tilde{P}_{t+1})$.

3.3 Maintaining Feasibility of Locally Basic Solutions

As indicated at the end of Section 3.1, nonnegativity of α_{t+1} does not imply nonnegativity of the parametrized values of the variables which are locally basic in periods $1, \ldots, t$. Two general approaches to surmounting this difficulty are possible. First, the feasibility of earlier periods can be ignored during the optimization of the modified problem $(\tilde{P}_{t+1})$ and restored in a subsequent procedure which would minimize an appropriate infeasibility form. The alternative is to employ, while optimizing $(\tilde{P}_{t+1})$, an extended minimum-ratio test which follows the parametric variation of locally basic variables in earlier periods and indicates when such a variable blocks the increase of an incoming column in $(\tilde{P}_{t+1})$.

The latter approach is adopted here. The representation of an incoming column in $(\tilde{P}_{t+1})$ includes weights on any surrogate columns which are in the local basis. Using these weights and repeatedly applying (14) and (9) yields a representation of the incoming column in terms of the variables which are locally basic in periods $1, \ldots, t+1$. This representation is used to implement the extended minimum-ratio test. This scheme of local inverses linked by representations of surrogate columns can be viewed as a compact basis-inverse technique which maintains a nearly block-angular inverse of the columns which are locally basic in periods $1, \ldots, t+1$.

When the extended minimum-ratio test reveals that a variable in a period prior to $t+1$ blocks the increase of an incoming column in $(\tilde{P}_{t+1})$ several strategies may be employed. The choice in this work is to "shuffle" the structure of the surrogate columns by exchanging the roles of some of the activities which are and are not locally basic. This process is designed to take the blocking variable and pass it down to $(\tilde{P}_{t+1})$ as a surrogate column. Primal and dual solutions in each period are unchanged, and when optimization of $(\tilde{P}_{t+1})$ is resumed, the same incoming variable is blocked by this surrogate column. The indicated pivot in $(\tilde{P}_{t+1})$ is made and maintains the nonnegativity of locally basic solutions in periods prior to $t + 1$.

3.4 Computational Strategies and Further Work

Again, a wide variety of computational strategies may be employed within the framework described above. There is freedom in the order in which the problem $(\tilde{P}_t)$ are solved and in when to pass information between problems in the form of cuts, modified right-hand sides, and surrogate columns. The presence of the surrogate columns opens additional options, including variations of the "shuffle" described in Section 3.3.

Currently, computational experience is being obtained with a code written by Wittrock in Mathematical Programming Language at Stanford University. This language facilitates experimentation of the type needed at this stage of the work. The thrust of this experimentation is to devise computational strategies which tend to minimize the computational effort needed to obtain an optimal solution to (P) . Since the manipulations of data structures, and the form and frequency of updates to the local inverses, depend heavily upon the computational strategies which are employed, decisions about these factors have not yet been made.

BIBLIOGRAPHY

[1] Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey (1963).

[2] Dantzig, G.B., "Solving Staircase Systems", Technical Report SOL 80-00, Systems Optimization Laboratory, Department of Operations Research, Stanford University (1979).

[3] Dantzig, G.B., and P. Wolfe, "Decomposition Principle for Linear Programs", Operations Research 8 (1960), 101-110.

[4] Fourer, R., "Sparse Gaussian Elimination of Staircase Systems", Technical Report SOL 79-17, Systems Optimization Laboratory, Department of Operations Research, Stanford University (1979).

[5] Glassey, C.R., "Nested Decomposition and Multi-stage Linear Programs", Management Science 20 (1973), 282-292.

[6] Ho, J.K., and E. Loute, "A Comparative Study of Two Methods for Staircase Linear Programs", Transations on Mathematical Software, ACM 6 (1980), 17-30.

[7] Ho, J.K., and A.S. Manne, "Nested Decomposition for Dynamic Models", Mathematical Programming 6 (1974), 121-140.

[8] van Slyke, R.M., and R. Wets, "L-shaped Linear Programs with Applications to Optimal Control and Stochastic Programming", SIAM Journal of Applied Mathematics 17 (1969), 638-663.

[9] Wittrock, R., private communication (1980).

LIFT: A NESTED DECOMPOSITION ALGORITHM FOR SOLVING LOWER BLOCK TRIANGULAR LINEAR PROGRAMS

D. Ament,* J. Ho,** E. Loute,*** M. Remmelswaal*

*Econometric Institute, Erasmus University, Rotterdam
**Applied Mathematics Department, Brookhaven National Laboratory, Upton, N.Y.
***CORE, Catholic University of Louvain, Louvain-la-Neuve

The lower block triangular structure is typical of time phased linear programs with multiple lags in the variables. We propose an algorithm for solving this class of problems based on the Ho and Manne (1973) nested decomposition algorithm. A computer code of this algorithm (LIFT) has been developed based on state-of-the-art modular linear programming software (IBM's MPSX/370).

We present and discuss the implementation aspects of this code and discuss several computational strategies currently available in LIFT. Preliminary computational experience with large (5000–6000 rows) energy technology assessment models is presented.

1. INTRODUCTION

Nested decomposition of linear programs is the result of a multilevel, hierarchical application of the Dantzig-Wolfe decomposition principle [4]. It has been shown to be a promising approach to large scale problems with the staircase structure [6] [9]. Staircase LP's arise from dynamic models without explicit time-lags or feedbacks. This means that a variable is involved only in its own time period and perhaps also in the one followinp immediately. An important generalization is to allow explicit time-lags, i.e. a variable may be involved in its own and any succeeding time period. The general structure is called lower block-triangular and permits direct accounting of long term effects of investment, service life, etc. Although lower block-triangular problems can be transformed to equivalent staircase problems by the addition of variables and constraints, the more compact and more natural formulation would undoubtedly be favored by modellers. We may assume that models with time lags will be generated as such. Now, the use of additional software to convert the input and output data will be too costly and cumbersome. Therefore, in designing ar advanced implementation of nested decomposition primarily for the staircase structure, we decided to treat the more general lower block-triangular structure. The derivation of the algorithm is similar to that in [5] but the formulas are now more complicated when there is time-lagged coupling. The implementation known as LIFT is based on state-of-the-art modular LP software (IBM's MPSX/370). Its design is along the same lines as DECOMPSX [9].

This talk outlines both the algorithmic and software aspects of LIFT and presents computational results.

2. BLOCK-TRIANGULAR LP MODELS

A block-triangular LP problem is a linear programming problem formulated as follows :

$$\min z = \sum_{t=1}^{T} c_t x_t \qquad (0)$$

$$\text{subject to } \sum_{s=1}^{t} A_{ts}\, x_s = d_t \qquad t=1, \ldots, T \qquad (1,t)$$

$$x_t \geqslant 0, \; t=1, \ldots, T$$

where c_t is $1 \times n_t$, x_t is $n_t \times 1$, d_t is $m_t \times 1$ for $t = 1, \ldots, T$ and A_{ts} is $m_t \times n_s$ for $s, t = 1, \ldots, T$ and $s \leqslant t$.

The constraint matrix of such a problem exhibits a lower block-triangular structure (figure 1). Nonzero coefficients of the constraint matrix can be found only in the submatrices A_{ts} $s,t=1, \ldots, T$; $s \leqslant T$.

Many structured LP problems can be cast in that form if one allows some of the submatrices A_{ts} to be zero (see figures 2, 3). The most important among these are dynamic LP models, also referred to as multistage, multiperiod or time phased LP problems, which are considered as difficult problems to solve when their size is large. LIFT is primarily intended for solving dynamic LP problems with the staircase structure (figure 2) or with the structure of figure 3. Other special casesof lower block-triangular LP's include : primal or dual block-angular problems, and primal-dual block-angular problems. Although LIFT may be applied to such problems it is unlikely to be as efficient as other single-level decomposition algorithms (Dantzig Wolfe or Benders decomposition algorithms, etc.).

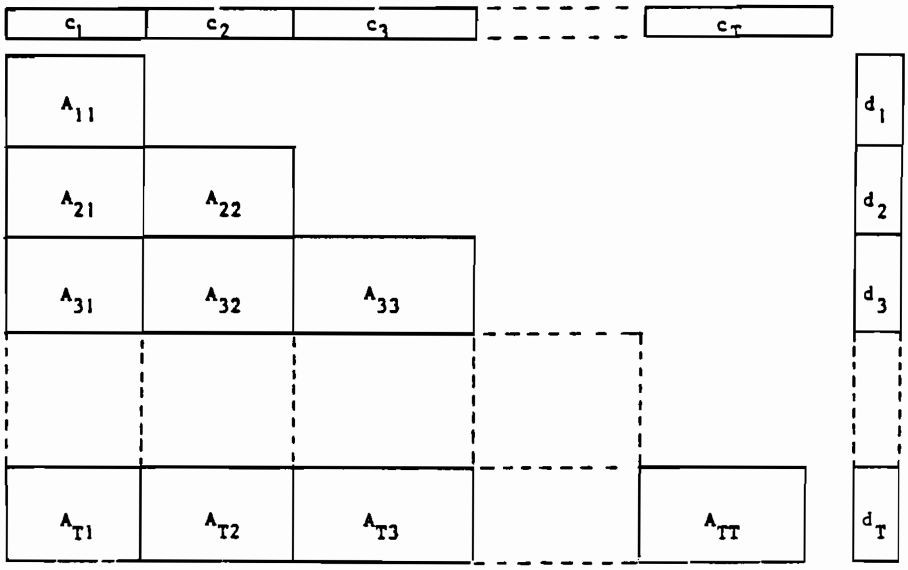


Figure 1 : Lower block triangular structure in a block structured LP

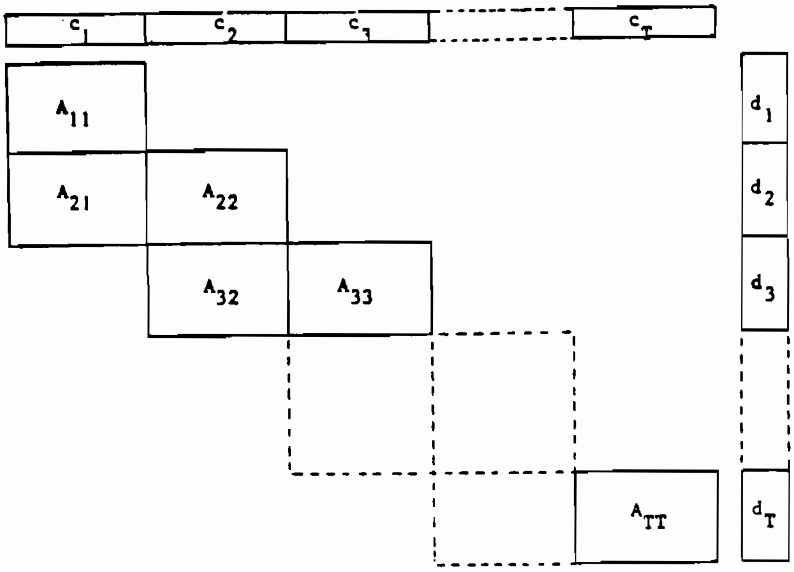


Figure 2 : Staircase Structure

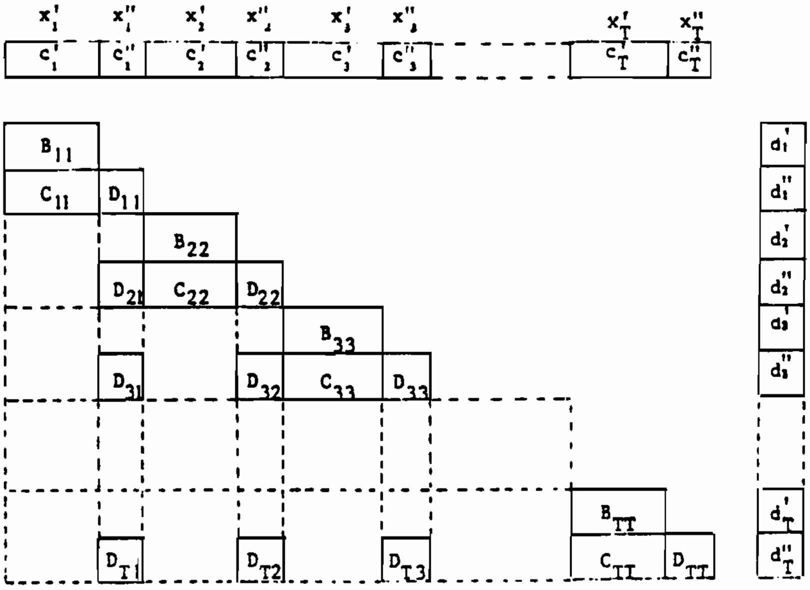


Figure 3 : Structure of an LP investment planning model

3. NESTED DECOMPOSITION ALGORITHM

The algorithm consists of solving a sequence of subproblems defined as follows.

$$SP_t^k \left| \begin{array}{lll} \min z_t^k = \left(p_t^k - \sum_{s=t+1}^{T} \pi_s^k Q_{st}^k\right) \lambda_t^k + \left(c_t - \sum_{s=t+1}^{T} \pi_s^k A_{st}\right) x_t & & (3) \\ \text{s.t.} \qquad Q_{tt}^k \lambda_t^k + A_{tt} x_t = d_t & & (4) \\ \qquad\qquad \delta_t^k \lambda_t^k = 1 & & (5) \\ \qquad\qquad \lambda_t^k , x_t > 0 & & (6) \end{array} \right.$$

where

p_t^k is an $1 \times k$ row vector

π_s^k is an $1 \times m_s$ row vector, $s=t+1, \ldots, T$

Q_{st}^k is an $m_s \times k$ matrix, $s=t, \ldots, T$

λ_t^k is an $k \times 1$ column vector

c_t is an $1 \times n_t$ row vector

x_t is an $n_t \times 1$ column vector

A_{st} is an $m_s \times n_t$ matrix, $s=t+1, \ldots, T$

d_t is an $m_t \times 1$ column vector

δ_t^k is an $1 \times k$ row vector

p_t^k, Q_{st}^k and δ_t^k are defined recursively as follows.

$$\left.\begin{array}{l}
\text{the jth component of } p_2^k\text{: } p_{2j}^k \equiv c_1 x_1^j,\ j=1,\ldots,k; \\
\text{the jth component of } p_t^k\text{: } p_{tj}^k \equiv c_{t-1}\, x_{t-1}^j + p_{t-1}^j\, \lambda_{t-1}^j \\
\qquad\qquad\qquad\qquad \text{for } j=1,\ldots,k \\
\qquad\qquad\qquad\qquad\quad t=3,\ldots,T;
\end{array}\right\} \quad (7)$$

$$\left.\begin{array}{l}
\text{the jth column of } Q_{s2}^k\text{: } q_{s2}^j \equiv A_{s1}^j\, x_1^j \\
\qquad\qquad\qquad\qquad \text{for } j=1,\ldots,k \\
\qquad\qquad\qquad\qquad\quad s=2,\ldots,T; \\
\text{the jth column of } Q_{st}^k\text{: } q_{st}^j \equiv Q_{st-1}^j\, \lambda_{t-1}^j + A_{st-1}\, x_{t-1}^j \\
\qquad\qquad\qquad\qquad \text{for } j=1,\ldots,k \\
\qquad\qquad\qquad\qquad\quad s=t,\ldots,T \\
\qquad\qquad\qquad\qquad\quad t=3,\ldots,T;
\end{array}\right\} \quad (8)$$

$$\left.\begin{array}{l}
\text{the jth component of } \delta_2^k\text{: } \delta_{2j} \equiv \begin{Bmatrix}1\\0\end{Bmatrix} \text{if } x_1^j \text{ is an extrem } \begin{Bmatrix}\text{point}\\\text{ray}\end{Bmatrix} \text{ solution,} \\
\text{for } j=1,\ldots,k; \\
\text{the jth component of } \delta_t^k\text{: } \delta_{tj} \equiv \begin{Bmatrix}1\\0\end{Bmatrix} \text{if } (\lambda_{t-1}^j, x_{t-1}^j) \text{ is an extreme } \begin{Bmatrix}\text{point}\\\text{ray}\end{Bmatrix} \text{ solution,} \\
\text{for } j=1,\ldots,k;\ t=3,\ldots,T.
\end{array}\right\} \quad (9)$$

We let SP_t denote the t-th subproblem with k unspecified. The index k denotes a cycle in the algorithm. SP_t^k is then read as subproblem t at cycle k. To simplify the notation, we assume that all subproblems at cycle k have the same number of proposals. For k=1, delete the terms involving λ and define $\left(x_t^1, \lambda_t^1\right) = (0,1)$ to be associated with the null proposal. We may assume without loss of generality that $\left(x_1^1\right)$, $\left(\lambda_t^1, x_t^1\right)$ for t=2, ..., T are feasible solutions so that $\delta_t^1 = (1)$, t=2, ..., T.

As a subproblem, SP_t^k uses prices from SP_{t+1}^k and generates a proposal (if any) for SP_{t+1}^{k+1}. As a restricted master problem, SP_t^k optimizes over the available proposals and generates prices for SP_s^k, s=t-1, ..., 1. Note that SP_T^k acts only as a master and SP_1^k only as a subproblem. The latter has the form

$$SP_1^k \left| \begin{array}{l} \min\left(c_1 - \sum_{s=2}^{T} \pi_s^k A_{s1}\right) x_1 \\ \quad A_{11} x_1 = d_1 \\ \quad\quad x_1 \geqslant 0 \end{array} \right.$$

Each proposal from SP_t consists of the following parts :

p_{t+1s}^k a scalar which is the actual unit cost for the proposal

$q_{s,t+1}^k$ an m_s x 1 column vector representing the coupling between periods t and $s (t < s \leqslant T)$, in fact $q_{t+1,t+1}^k$ represents an accumulated coupling between periods 1, ..., t and t+1. Only this coupling is explicitly constrained.

3.1. A Phase 1 Procedure

A phase 1 procedure is necessary to determine either a feasible starting basis for each subproblem SP_t, t=1, ..., T, or that the original problem (1) is infeasible.

Let $[SP_1, \ldots, SP_t]$ denote the system comprising the first t subproblems. Thus $[SP_1, \ldots, SP_T]$ represents the nested decomposition of (1). The following procedure is based on the observation that (1) is feasible if and only if each $[SP_1, \ldots, SP_t]$, t=1, ..., T is feasible.

Phase 1

```
Step (i)   : Set t=1. Find an extreme point solution to $SP_1$.
             If none exists, stop : (1) is infeasible.
Step (ii)  : If t=T, a feasible basis is available to $SP_t$, t=1, ..., T.
             Otherwise, form a proposal for $SP_{t+1}$.
             Set t=t+1.
Step (iii) : Start with an artificial basis for $SP_t$. Set the objective to be
             an infeasibility form i.e. a sum of artificial variables in $SP_t$,
             minimize this objective over $[SP_1, \ldots, SP_t]$, using the phase
             2 procedure in section 3.2.
             If the minimum is not zero, stop : (1) is infeasible.
             Otherwise go to step (ii).
```

3.2. A Phase 2 Procedure

Phase 2

```
Step (i)   : Set $k=1$.
Step (ii)  : Set $t=T$.
Step (iii) : Solve $SP_t^k$.
        If $SP_t^k$ has an optimal solution and
            a) $t < T$ : send extreme point proposal (if any) to $SP_{t+1}^{k+1}$;
            b) $t > 1$ : send prices to $SP_s^k$, $s=1, \ldots, t-1$;
        If $SP_t^k$ is unbounded from below and
            c) $t < T$ : send extreme ray proposal to $SP_{t+1}^{k+1}$ and go to step (ii).
            d) $t = T$ : stop, the problem is unbounded from below.
Step (iv)  : Set $t=t-1$. Return to step (iii) if $t > 0$.
Step (v)   : If no proposal is generated by any $SP_t^k$, i.e.
                 $z_t^k - \sigma_{t+1}^k = 0,\ t=1, \ldots, T-1$ :
             Stop, a minimum is achieved. Otherwise, set $k = k+1$ and return
             to step (ii).
```

3.3. A Phase 3 Procedure

We shall describe a procedure for the reconstruction of a feasible solution to the original problem at the end of cycle k (more precisely right after SP_T^k has been solved). We call this procedure Phase 3. We assume that

SP^k_T is bounded. Denote the feasible solution vectors by y_t and μ_t. Let $\mu_T = \lambda^{k+1}_T$ and $y_T = x^{k+1}_T$ where $\left(\lambda^{k+1}_T, x^{k+1}_T\right)$ is an extreme point optimal solution of SP^k_T. The remaining vectors are determined by solving a sequence of LP problems SY_t, t=T-1, ..., 1, defined as follows :

$$\min\ p_t\mu_t + c_t y_t$$

$$SY_t \left| \begin{array}{ll} \text{subject to} & Q_{tt}\mu_t + A_{tt}\, y_t = d_t \\ & Q_{st}\mu_t + A_{st}y_t = d_s - \Sigma^{s}_{r=t+1} A_{sr}y_r \\ & \qquad \text{for } s=t+1, \dots, T \\ & \delta_t\mu_t = 1 \\ & \mu_t, y_t \geqslant 0 \end{array}\right.$$

and

$$SY_1 \left| \begin{array}{ll} & \min\ c_1y_1 \\ \text{subject to} & A_1y_1 = d_1 \\ - & A_{t1}y_1 = d_t - \Sigma^{t}_{s=2} A_{ts}y_s, \quad t=2, \dots, T \\ & y_1 \geqslant 0 \end{array}\right.$$

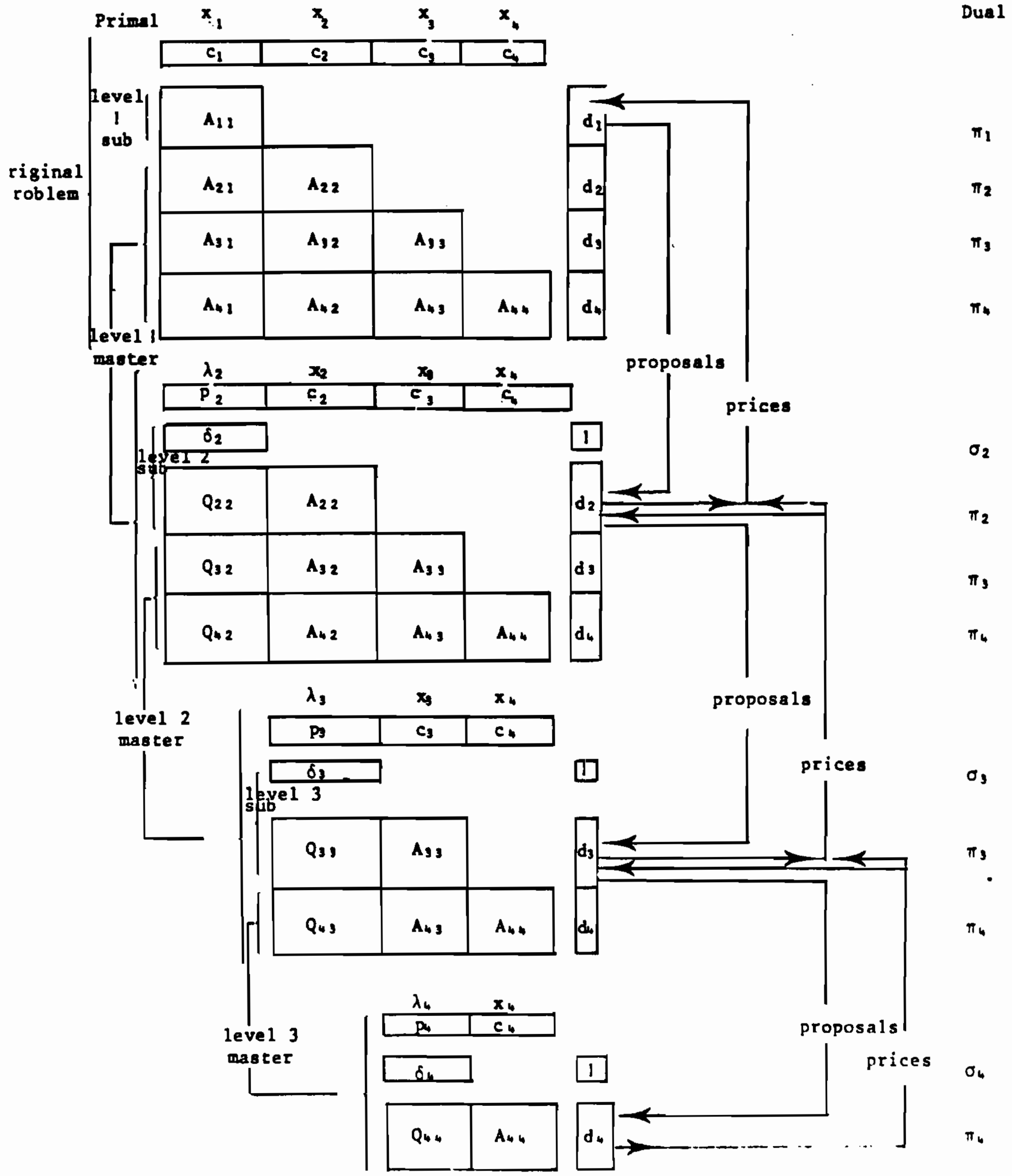


Figure 4 : Nesting of master problems and subproblems

4. AN IMPLEMENTATION BASED ON MPSX/370

The necessary data structure and computational strategies for an efficient implementation of decomposition algorithms have been discussed in [6] and [9]. Only straightforward adaptation or simple extensions are required for the algorithm in section 3. The advantages of the modularity of MPSX/370 are also studied in some detail in [9] and [14].We proceed directly to describe briefly LIFT, an implementation of the nested decomposition algorithm coded in PL/I, with major services provided by MPSX/370. Details are given in [2].

4.1. Procedures in LIFT

The services required and the MPSX/370 procedures used are listed in table 4.1. Procedures written in PL/I for LIFT are called external procedures. These are interfaced with MPSX/370 procedures by means of a bootstrap procedure called DPLBOOT. It loads the appropriate modules on the first call of any MPSX/370 procedure, thereby establishing all the links necessary for subsequent calls. This linkage is completely transparent to the user. LIFT consists of 11 external procedures (totalling 2 200 instructions) and two libraries of macros. One library is that of MPSX/370 ECL macros and the other contains LIFT specific macros.

The file organization is illustrated in Figure 5. Table 4.3. lists the purpose and frequency of use for each file. Two problem files are used in a flip-flop manner. Repeated problems ravision leads to a waste of storage space. When the current problem file is about to over-flow; it is compressed while copied to the other file.

SERVICE	FREQUENCY	MPSX/370 procedure
Reading and checking the subproblem data	once	CONVERT
Adding convexity row and Phase 3 objective row and right-hand-side	once	REVISE
Setting up and scaling the subproblem	cycle	SETUP
Forming the objective row	iteration$^+$	FORMC*
Solving the subproblem	cycle	PRIMAL*
Multiproposal generation tests	iteration$^+$	PRIMAL*
Forming an extreme point proposal	several times during a cycle	current solution in work region
Forming an extreme ray proposal	several times during a cycle	FTRANL followed by FTRANU*, MODIFY to fix the column at its bound
Adding new proposals	cycle	REVISE
Purging unprofitable proposals	several cycles	REVISE
Saving and restoring the basis	cycle	SAVE and RESTORE
Computing the dual solution	cycle	BTRANU followed by BTRANL
Setting up the subproblem right-hand-side in Phase 3	once	REVISE

Table 4.1

*These MPSX/370 procedures have been modified to service the subproblems (see [3]).

$^+$a simplex iteration.

LIFT major external procedures	function	frequency of use	called by	major calls to LIFT procedures	major calls to MPSX/370 procedures
CONTROL	main procedure	once	-	INITPAR SAVREST SETSUB NEST PHASE3	-
SAVREST	saves and restores a problem	once	CONTROL NEST	INITPAR	-
INITPAR	initializes controls and parameters	once	CONTROL SAVREST	-	-
SETSUB	reads the subproblem data, performs first setup, creates lists of coupling rows, insertsconvexity row,rhs and objective for phase 3	once for each subproblem	CONTROL	-	CONVERT SETUP INQUIRE SELIST REVISE FREECORE
NEST	driver procedure for phase 1 and phase 2 of the algorithm	once	CONTROL	SOLVSUB GARBCOL PROPSCL SAVREST	SETUP RESTORE REVISE ASSIGN COPY FREECORE

PROPSCL	computes scaling factor for extreme point proposals	once	NEST	-	-
GARBCOL	purges unprofitable proposals from sub-problem file	several cycles	NEST	-	FREECORE SETUP RESTORE SELIST PREMUL REVISE
SOLVSUB	solves subproblem sets up appropriate on conditions creates price vector for lower level subproblems	each subproblem at each cycle	NEST PHASE3	-	INVALUE PRIMAL BTRANU1 BTRANL1 GETVEC1 FTRANU1 FTRANL1 SAVE SOLUTION
PHASE3	driver procedure for phase 3 of the algorithm	once	CONTROL	SOLVSUB	SETSUP RESTORE REVISE ASSIGN COPY FREECORE

Table 4.2.

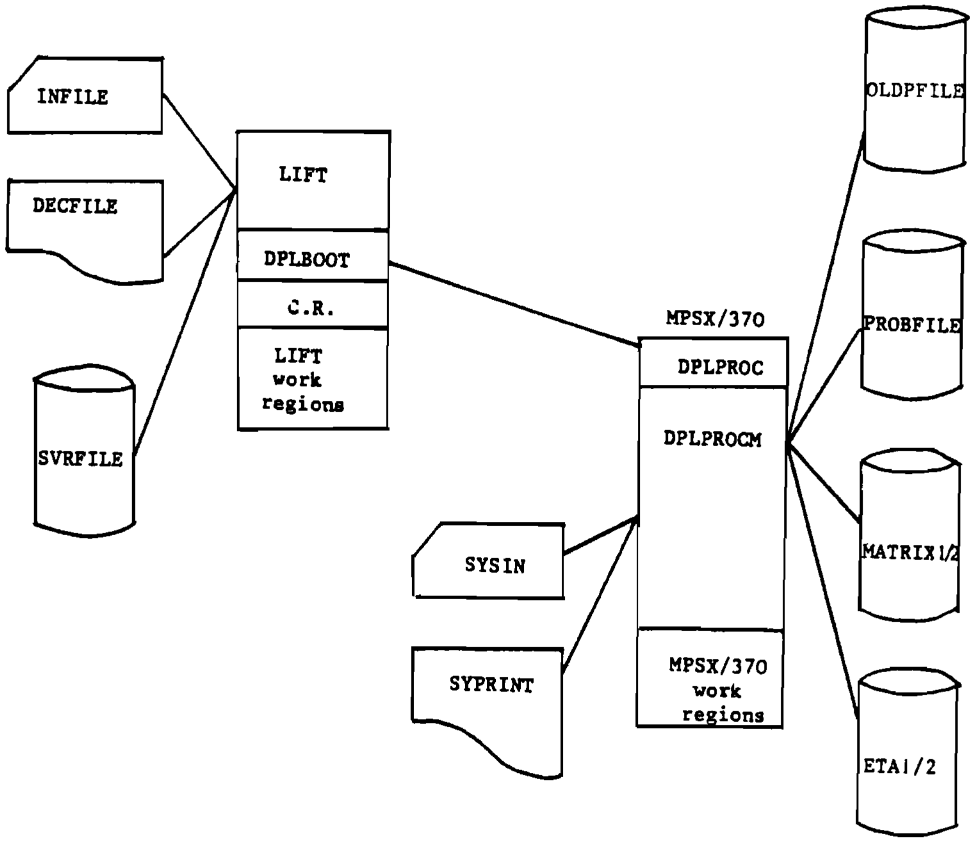

INFILE
DECFILE
SVRFILE
LIFT
DPLBOOT
C.R.
LIFT work regions
MPSX/370
DPLPROC
DPLPROCM
MPSX/370 work regions
SYSIN
SYPRINT
OLDPFILE
PROBFILE
MATRIX1/2
ETA1/2

Figure 5

File	Type	Purpose	Frequency of use
INFILE	sequential	used to specify the subproblems characteristics and control parameters	once
DECFILE	sequential	cycle log for the algorithm	cycle
SVRFILE	sequential	save and restore file for the algorithm	once
SYSIN	sequential	file containing the subproblems data in MPS format	once
SYSPRINT	sequential	iteration log of the simplex algorithm	simplex iteration
PROBFILE and OLDPFILE	direct access	problem files for the subproblems, used in a flip-flop way.	cycle
MATRIX1/2	direct access	work matrix files (internal file of MPSX/370)	each subproblem at each cycle
ETA1/2	direct access	eta files (U^{-1} and L^{-1}) of basis inverse representation (internal file of MPSX/370).	

Table 4.3.

LIFT has been designed to handle problems with up to 99 periods (i.e. a maximum of 99 subproblems). The maximum number of coupling rows for a subproblem is 1000. Both limits could be easily changed but the proposal buffer storage requirements may become prohibitive for a large number of coupling rows. The subproblems could theoretically have up to 16000 rows, but a practical limit is in the 2000-4000 range.

4.2. Input data for LIFT

For a problem with T periods, the data for the T subproblems comprise T contignous, complete data sets in the MPS format [11]. They are entered in the reversed period order (i.e; T,T-1,...,1). In the row section of each subproblem, the coupling rows that are active (i.e. having non zero coefficients in that subproblem) must be declared first, along with the objective row. Note that a subproblem SP_t inherits all active coupling rows of SP_{t-1} in periods t+1,...T. The date partitioning is very natural and allows independent generation of the subproblems as long as the names and units of the coupling rows are consistent.

It remains to specify the control parameters for LIFT in a separate file. The most important ones are the strategic parameters which,

i) indicate the subproblems are to be solved backwards or alternately backwards and forwards;

ii) specify the percentage improvement, frequency and maximum number of proposal in the mechanism for multi-proposal generation;

iii) control the proposal purging mechanism ;

iv) determine whether the prices or coupling rows of a subproblem have changed significantly to call for its resolution; and

v) control printing.

5. COMPUTATIONAL EXPERIENCE

We have experimented with LIFT on a series of test problems. Table 5.1 summarizes the statistics of the test problems. The nature and origin of all the test problems (except MODGLOB and BEDYOO) are given in [10]. SCAGR7/25 (group A) are agricultural planning models. SCSD1/6/8 (group B) are structural design optimization problems. SCFXM1/2/3 (group C) are production scheduling problems. SCTAP1/2/3 (group D) are dynamic traffic assignment problems. SCORPION, SCRS8, MODGLOB and BEDYOO (group E) are energy models. Note that only the last two have a truly lower block-triangular structure.

All test problems were solved with standard MPSX/370. The parameters were set to their default values and the macro OPTIMIZE [11] was called in these runs. Table 5.2 reports the corresponding statistics. All runs were made under VM/CMS on an IBM/370 model 158 in a 1000K virtual machine (1500K for BEDYOO).

We solved all the test problems with LIFT in a 1000K virtual machine (1200K for BEDYOO). The same control parameters were used in all the experiments. All runs were stopped with a primal-dual gap of less than 0.1%. Table 5.3 summarizes the statistics of the runs. Phase 3 always terminated in one cycle, because of little purging (if any) in phase 1 and 2. In addition to input, phases 1,2 and 3: the total CPU time accounts for table construction for the program, initializations, computing proposal scaling factors, saving the problem, etc.

Our experiments were aimed at testing the robustness of LIFT and efficiency as compared to standard MPSX/370 but not at validating its computational strategies. This has been done by J. Ho in [6].

Problem	Type (*)	Number of Periods	Number of			Density
			Rows	Columns (**)	Nonzeros	
SCAGR7	S	7	130	270	683	1.94
SCAGR27	S	25	472	972	2501	0.55
SCSD1	S	3	78	838	3226	4.94
SCSD6	S	7	148	1498	5814	2.62
SCSD8	S	39	398	3148	11732	0.94
SCFXM1	S	4	331	788	2943	1.13
SCFXM2	S	8	661	1575	5890	0.57
SCFXM3	S	12	991	2362	8837	0.38
SCTAP1	S	10	311	791	3683	1.50
SCTAP2	S	10	1101	2981	14395	0.44
SCTAP3	S	10	1491	3971	19045	0.32
SCORPION	S	6	389	747	2097	0.72
SCRS8	S	16	491	1660	4520	0.55
MODGLOB	T	6	1387	4547	17828	0.28
BEDYOO	T	6	5898	12048	25184	0.03

Table 5.1

(*) 'S' means 'staircase'
'T' means 'lower block_triangular'

(**) Including slacks

		SCAGR7	SCAGR25	SCSD1	SCSD6	SCSD8	SCFXM1	SCFXM2	SCFXM3	SCTAP1	SCTAP2	SCTAP3	SCORPION	SCRS8	MODGLOB	BEDYOO
	Phase 1	130	527	155	313	1252	254	535	820	231	601	795	316	468	1759	4761
Iterations	Phase 2	60	369	195	584	1361	178	355	489	51	186	256	82	303	1396	4311
	Total	190	896	350	897	2613	432	890	1309	282	787	1051	398	771	3155	9072
	Input	0.02	0.06	0.10	0.17	0.32	0.06	0.12	0.18	0.11	0.41	0.54	0.07	0.14	0.37	0.59
CPU	Phase 1	0.03	0.31	0.08	0.30	3.76	0.18	0.46	0.98	0.12	0.43	0.71	0.12	0.22	2.12	8.50
time	Phase 2	0.04	0.75	0.19	0.75	3.88	0.17	0.71	1.50	0.7	0.57	1.02	0.12	0.64	5.75	58.63
(min)	Phase 1+2	0.07	1.06	0.27	1.05	7.64	0.35	1.17	2.48	0.20	1.00	1.73	0.24	0.86	7.87	67.13
	Total (incl solution)	0.11	1.18	0.41	1.29	8.12	0.45	1.38	2.80	0.34	1.58	2.54	0.35	1.08	8.54	69.42
I/O		1073	3011	2107	5388	21008	1801	4088	7926	1824	6972	10417	1529	3792	22572	232343

Table 5.2

GROUP		A		B			C			D			E			
PROBLEM		SCAGR7	SCAGR25	SCSD1	SCSD6	SCSD8	SCFXM1	SCFXM2	SCFXM3	SCTAP1	SCTAP2	SCTAP3	SCORPION	SCRS8	MODGLOB	BEDYOO
NUMBER OF PERIODS		3	6	3	3	6	4	4	6	5	5	5	6	4	6	6
CYCLES	PHASE 1	4	11	2	2	5	11	7	13	4	4	4	5	8	5	5
	PHASE 2	6	12	4	8	3	15	12	16	6	7	10	4	7	6	6
	TOTAL	10	23	6	10	8	26	19	29	10	11	14	9	15	11	11
CPU TIME (MIN)	INPUT	0.03	0.11	0.13	0.23	0.96	0.09	0.18	0.28	0.16	0.58	0.77	0.12	0.19	0.49	0.75
	PHASE 1	0.12	0.90	0.03	0.22	0.83	0.90	1.11	1.93	0.12	0.72	1.15	0.15	0.99	2.14	8.74
	PHASE 2	0.28	1.71	0.30	1.01	1.23	1.52	2.38	4.96	0.68	2.22	3.87	0.35	1.05	3.62	6.81
	PHASE 3	0.08	0.19	0.13	0.21	0.63	0.28	0.25	0.47	0.19	0.64	0.96	0.18	0.42	0.87	3.76
	PHASE 1 + 2 + 3	0.48	2.80	0.46	1.44	2.69	2.70	4.23	7.36	0.99	4.16	5.98	0.68	2.46	6.63	20.06
TOTAL CPU TIME (MIN)		0.57	3.22	0.64	1.74	3.62	2.91	4.40	8.41	1.20	4.89	7.94	0.97	2.88	7.80	22.14
CPU TIME IN MPSX/370 PROCEDURES	SETUP	0.10	0.56	0.12	0.35	0.30	0.53	0.61	1.37	0.21	0.72	1.23	0.11	0.50	0.80	1.61
	PRIMAL	0.16	1.28	0.19	0.75	1.63	1.21	2.28	4.32	0.35	1.82	3.20	0.23	1.27	4.51	14.65
	REVISE	0.04	0.28	0.04	0.08	0.13	0.35	0.31	0.66	0.04	0.30	0.37	0.02	0.20	0.35	0.51
	TOTAL	0.30	2.12	0.35	1.12	2.06	2.09	3.20	6.35	0.60	2.84	4.80	0.36	1.97	5.66	16.77
I/O		5915	28861	6750	13547	17525	24299	26400	52418	9648	30492	46865	8061	22744	48680	95071

Table 5.3

We observe that for some classes of test problems (group B and E) LIFT tends to be more efficient than MPSX/370 as the problem size increases.

It can be observed in table 5.3 that only a small portion of CPU time is spent on LIFT's external procedures. The rest is taken up by MPSX/370 setup, revise and simplex iterations. This leads us to believe that improvement of the efficiency of LIFT should come primarily from strategic manipulation aimed at reducing the number of cycles.

We believe that LIFT is a valuable addition to a mathematical programming system such as MPSX/370. It provides, for some classes of problems, a way to solve large problems more economically. Problems that are impractical to solve by standard software may now be accomodated.

REFERENCES

[1] Ament D., Loute E., Remmelswaal M., "LIFT User's Manual", in preparation.

[2] Ament D., Loute E., Remmelswaal M., "LIFT System Manual", in preparation.

[3] Culot B., Loute E., "DECOMPSX System Manual", (in French), CORE computing report 80-B-02, C.O.R.E., Université Catholique de Louvain, Belgium, 1980.

[4] Dantzig G.B., Wolfe P., "Decomposition Principle for Linear Programs". Operations Research, vol.8, pp. 101-11, 1960.

[5] Ho J.K., Manne A.S., "Nested Decomposition for Dynamic Models", Mathematical Programming 6, 1974, pp.121-140.

[6] Ho J.K., "Implementation and Application of a Nested Decomposition Algorithm", in Computers and Mathematical Programming, ed. W.W. White, NBS 1978, pp.21-30.

[7] Ho J.K., Loute E., Smeers Y. and Van de Voort E., "The Use of Decomposition Techniques for Large Scale Linear Programming Energy Models", Energy Policy, special issue on "Energy Models for the European Community", ed. A. Strub, June 1979, pp. 94-101.

[8] Ho J.K., Loute E., "A Comparative Study of Two Methods for Staircase Linear Programs", Transactions on Mathematical Software, ACM, 6, 1980, pp. 17-30.

[9] Ho J.K., Loute E., "An Advanced Implementation of the Dantzig Wolfe Decomposition Algorithm", CORE DP 8014, Université Catholique de Louvain, Belgium, 1980.

[10] Ho J.K., Loute E., "A Set of Staircase Linear Programming Test Problems", CORE Computing Report 80-B-05, C.O.R.E., Université Catholique de Louvain, Belgium, 1980.

[11] IBM, "IBM Mathematical Programming System Extended/370 (MPSX/370) Program Reference Manual", SH19-1095-3, December 1979.

[12] IBM, "IBM Mathematical Programming System Extended/370 (MPSX/370)Logic Manual", (licensed material) LY19-1024-0, January 1975.

[13] Simmonard M., "Programmation Linéaire", vol. 1 et 2, Dunod, Paris, 1972.

[14] Slate L., Spielberg K., "The Extended Control Language of MPSX/370 and Possible Applications", IBM Systems Journal, 17, 1978, pp. 64-81.

SOLVING LINEAR PROGRAMMING PROBLEMS BY RESOURCE ALLOCATION METHODS

Klaus Beer

Mathematical Programming Department
Technische Hochschule
Karl-Marx-Stadt

We describe a method of feasible directions for the resource-allocation approach (see Kornai/Liptak, Geoffrion, and others) to solving LP problems. We compute the direction by application of the ϵ-subdifferential. The method generates primal feasible solutions which improve the objective function by at least ϵ at each iteration. The numerical experiments indicate that as a rule a solution near to the optimum is reached after about 10 iterations. The method converges to the exact value of the objective function only at a linear rate.

1. INTRODUCTION

In 1977 we published a book (see Beer [1]) on solving large-scale LP problems. We considered four types of methods:

(a) simplex methods for general problems using sparse matrix techniques such as LU-decomposition of the inverse basis matrix;

(b) simplex methods with basis matrix factorization for problems with special structures;

(c) iteration methods, such as penalty methods or augmented Lagrangian methods;

(d) decomposition methods.

In the book we made the following conclusions:

- The simplex method is competitive and is not obsolete for large problems. For normal cases we would use such software. In the GDR this is the PS OPSI, which is comparable with the MPS from IBM.
- Simplex methods for special structures are more effective, but are only applicable for the selected structure. In the GDR we do not have such software (except for the transportation problem).

- Iteration methods use simple computer programs and rapidly give solutions near to the optimum. We applied the penalty method for a large problem on a small computer.
- Decomposition methods are not recommended. The rate of convergence is too slow.

However, in the last four years we have dealt with some questions of realization of decomposition methods for the blockangular structure (coupled rows) and for the linked structure (coupled columns). We have carried out some experiments (see Käschel [4 ,5]).

If we want to get a solution that is only close to the optimum, then our decomposition methods equal the simplex method with respect to the solution time. The slow convergence in the essential part of the method (from a practical point of view the first iterations) is improved. An important improvement in the efficiency of the method was to use the ε-subdifferential instead of the usual subdifferential, which we recommended in Beer [1] for resource allocation methods. The decomposition method not only works for the case of a blockangular structure but also for a linked structure with primal feasible solutions.

2. THE GENERAL PRINCIPLE

In our opinion dealing with decomposition methods implies dealing with methods for subdifferentiable problems. The programming problems which are connected with decomposition have the following properties:

- The objective function is given only implicitly. This implies that to compute this function at a point, one must solve a programming problem.
- We may compute the subdifferential of the objective function by a generation technique.
- Usually, some of the restrictions are given only implicitly.

We conclude from this situation that we must apply subgradient or feasible direction methods.

Before we report about resource allocation methods it will be useful to discuss briefly the background of such an approach, specifically the method of feasible directions for nondifferentiable problems. Let us consider the problem (1)

$$(1) \qquad \left.\begin{array}{l} f(u) \to \max \\ u \in U \end{array}\right\}$$

where f is a subdifferentiable concave function and $f(u) = -\infty$ at some points $u \in U$ is possible. In LP-decomposition f is a piece-wise linear concave function. $U \in R^n$ is a closed convex set. For theoretical simplification we replace problem (1) by the equivalent problem (2)

$$(2) \qquad \left.\begin{array}{l} F(u) \to \max \\ u \in R^n \end{array}\right\}$$

with

$$F(u) = \begin{cases} f(u), & u \in U \\ -\infty, & u \notin U \end{cases} .$$

We suppose further that we know the subdifferential $\partial F(u)$ and that for all $\varepsilon > 0$ we know an extension $P_\varepsilon(u)$ of $\partial F(u)$, i.e., $P_\varepsilon(u) \supset \partial F(u)$, $u \in R^n$.

At each iteration of the method of feasible directions (FDM) we have to realize two operations:

- We must calculate a direction r with the property $F'(u,r) > 0$. We do it by solving the problem

  $$\max_{\|r\| \leq 1} \min \{ (r,v,) : v \in P_\varepsilon(u)\} \qquad \text{(DRP)} .$$

  Here(.,.) denotes the scalar product in R^n. (DRP) is a linear programming problem, if we use the sum or maximum norm. By using the Euclidean norm we get a quadratic programming problem. These LP or quadratic problems we may solve by column generation (see Beer [1], and Käschel [2]).

- The second operation is the determination of the steplength in the direction r. We must determine such a scalar λ that

  $$F(u+\lambda r) = \max \{F(u+\gamma r) : \gamma \geq 0\} \quad . \qquad \text{(SLP)}$$

  In decomposition methods (SLP) leads to parametric programming with one parameter, which is common for all subproblems.

Now we can formulate the FDM method in outline.

Step 1: Given a point $u^{\mathrm{o}} \in \mathrm{dom}\, F$, $\varepsilon_{\mathrm{o}} > 0$, $\gamma_{\mathrm{o}} \geq 0$, $\rho \in (0,1)$, we put $k := 0$, $t := 0$.

Step 2: We solve (DRP) by $\varepsilon = \varepsilon_k$, $u = u^t$. If we obtain during the solving process for (DRP) a solution (r^t, v^t) with the property

$$(r^t, v^t) = \min \{(r^t, v) : v \in P_{\varepsilon_k}(u^t)\} > \gamma_k \quad ,$$

then we go to step 3, otherwise we go to step 4.

Step 3: We solve (SLP) by $r = r^t$, $u = u^t$. We choose the steplength λ_t and put

$$u^{t+1} := u^t + \lambda_t r^t \, , \quad t := t+1$$

and return to step 2.

Step 4: We put

$$\varepsilon_{k+1} := \rho \cdot \varepsilon_k \, , \ \gamma_{k+1} := \rho \cdot \gamma_k \, , \ k := k+1$$

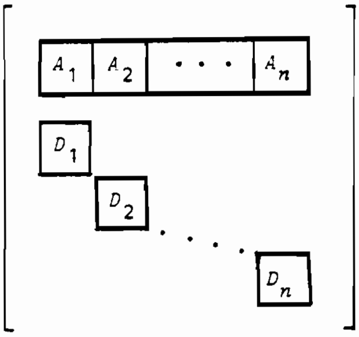
and go to step 2.

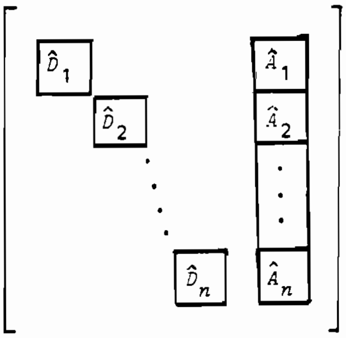
For details and proofs of convergence of FDM see Schwartz [8].

3. THEORY OF RESOURCE ALLOCATION

After this background let us now turn to the core of this paper. We consider problem *(A)*, and simultaneously problem *(Â)*, with block-structure:

$$(A)\begin{cases} \sum\limits_{i=1}^{n} (c_i, x_i) \rightarrow \max \\ \sum\limits_{i=1}^{n} A_i\ x_i \leq b \\ D_i\ x_i = d_i \\ x_i \geq 0\ ,\ i = 1(1)n \end{cases} \qquad (\hat{A})\begin{cases} (\hat{c}_0, \hat{x}_0) + \sum\limits_{i=1}^{n} (\hat{c}_i, \hat{x}_i) \rightarrow \max \\ \hat{A}_i\ \hat{x}_0 + \hat{D}_i\ \hat{x}_i = \hat{d}_i\ ,\ i = 1(1)n \\ \hat{x}_i \geq 0\ ,\ i = 1(1)n \end{cases}$$





We shall explain an improvement of the decomposition algorithms described in Beer [1], pp 188-202.

For $u_i \in R^m$ and $u \in R^{n}{}_{o}$ we put, respectively,

$$\phi_i(u_i) = \begin{cases} \max\ \{(c_i, x_i) : A_i\ x_i \leq u_i\ ,\ D_i\ x_i = d_i\ ,\ x_i \geq 0\ \},\{\ldots\} \neq \emptyset \\ -\infty \qquad\qquad ,\ \{\ldots\} = \emptyset \end{cases}$$

$$\hat{\phi}_i(u) = \begin{cases} \max\ \{(\hat{c}_i, \hat{x}_i) : \hat{D}_i\ \hat{x}_i = \hat{d}_i - \hat{A}_i\ u\ ,\ x_i \geq 0\ \},\{\ldots\} = \emptyset \\ -\infty \qquad\qquad ,\ \{\ldots\} = \emptyset \end{cases}$$

For our subproblems, we put

$$\Phi(u_1,u_2,\dots,u_n) = \sum_{i=1}^{n} \phi_i(u_i)$$

$$\hat{\Phi}(u) = (c_0,u) + \sum_{i=1}^{n} \phi_i(u)$$

where Φ and $\hat{\Phi}$ are concave, subdifferentiable, piece-wise functions.

We replace problems (A) and $(\hat{A})$ by (3) and (4), respectively.

$$F(u_1,u_2,\dots,u_n) = \Phi(u\ ,u\ ,\dots,u_n) + \delta(u/U) \to \max \tag{3}$$

$$\hat{F}(u) = \hat{\Phi}(u) + \delta(u/\hat{U}) \to \max \tag{4}$$

where

$$U = \{(u_1,u_2,\dots,u_n) \in R^{nm} : \sum_{i=1}^{n} u_i = b\}$$

$$U = \{u : u \geq 0\} \quad .$$

It is well known (for instance, from Kornai and Liptak [6], Lasdon [7], and Geoffrion [3]) that problems (3) and (4) are equivalent to (A) and $(\hat{A})$, respectively.

For the subdifferential we have the formulas (see Beer [1])

$$\partial\phi_i(u_i) = \{u_i^* : u_i^* A_i + y_i D_i \geq c_i\ ,\ u_i^* \geq 0\ ,$$
$$(u_i,u_i^*) + (y_i,d_i) \leq \phi_i(u_i)\}$$

$$\partial\hat{\phi}_i(u_i) = \{-u_i^* \hat{A}_i : u_i^* \hat{D}_i \geq \hat{c}_i\ ,\ (u_i^*\ ,\ d_i - \hat{A}_i u) \geq \hat{\phi}_i(u)\}\ ,$$

i.e., $\partial\phi_i$ is a projection of the set of optimal dual solutions from the subproblem $\phi_i(u)$ and therefore is a convex polyhedral set. The same is true for $\partial\hat{\phi}_i$.

Also we have

$$\partial F(u_1,u_2,\ldots,u_n) = \{u_1^* + w, u_2^* + w, \ldots, u_n^* + w) : \quad u_i^* \in \partial\phi_i(u_i)\ ,\ w \in R^m\}$$

$$\partial\hat{F}(u) = \{c_0 + \sum_{i=1}^{n} \lambda_i + \mu : \quad \lambda_i \in \partial\hat{\phi}_i(u)\ ,\ i = 1(1)n\ ,\ \mu \in K^*(u)\}$$

where $K^*(u)$ is a convex polyhedral cone (see Beer [1]).

Now we may formulate the basic results.

Theorem 1. If we denote by $\partial_\varepsilon\phi_i(u_i)$ the ε-subdifferential (after R.T. Rockafellar)

$$\partial_\varepsilon\phi_i(u) := \{u_i^* : \phi_i(w) - \phi_i(u_i) \leq (u_i^*\ ,\ w - u_i) + \varepsilon\ ,\ \forall w \in R^m\}$$

then the following formula is true

$$\partial_\varepsilon\phi_i(u_i) = \{u_i^* : u_i^* A_i + y_i D_i \geq c_i\ ,\ \ u_i^* \geq 0 \quad (u_i^*, u_i) + (y_i, d_i) \geq \phi_i(u_i) + \varepsilon\} \quad .$$

Thus $\partial_\varepsilon\phi_i(u_i)$ is a weak extension of the set of optimal dual solutions from the subproblem $\phi_i(u_i)$.

Theorem 2. If we put

$$P_\varepsilon(u_1,u_2,\ldots,u_n) := \{(u_1^* + w\ ,\ u_2^* + w, \ldots, u_n^* + w) : \quad u_i^* \in \partial_\varepsilon\phi_i(u_i)\ ,\ w \in R^m\}$$

then we have

$$\partial_{\varepsilon}F(u_1,u_2,\dots,u_n) \subset P_{\varepsilon}(u_1,u_2,\dots,u_n) \subset \partial_{\varepsilon n}F(u_1,u_2,\dots,u_n).$$

Analogous results to theorems 1 and 2 are also true for $\partial_{\varepsilon}\hat{\Phi}(u)$ and $\partial_{\varepsilon}\hat{F}$.

Theorems 1 and 2 permit us to use the FDM described above to solve the problems (3) and (4) respectively. This completes the requirements for the decomposition algorithms for the original problems (A) and $(\hat{A})$. The decomposition algorithms deal only with the linear subproblems $\phi_i(u_i)$ and the (DRP), which is also an LP-problem. Therefore, we can realize the decomposition algorithm on the basis of an existing LP-routine for a computer. At the Technische Hochschule, Karl-Marx-Stadt, we have realized it on the basis of PS OPSI. The practical implementation of the algorithms can not be described here (see [1],[2],[4] and [5]).

The convergence and some important practical aspects follow from the next theorem.

Theorem 3. We denote the optimal value of the objective function of problem (A) by F^*, and the vectors $u^t := (u^t, u^t, \dots, u_n^t) \in R^{mn}$ and $r^t := (r^t, r^t, \dots, r_n^t) \in R^{mn}$ by u^t and r^t respectively. We suppose that $F^* < \infty$.

Thus we have:

(a) From $0 \in P_{\varepsilon}(u^t)$ it follows that

$$F^* \geq F(u^t) \geq F^* - n\cdot\varepsilon \tag{5}$$

(b) If the (DRP) yields an improving direction, i.e., we go in the FDM from step 2 to step 3, then

$$F(u^{t+1}) \geq F(u^t) + \varepsilon$$

(c) There exists in the sequence $\{u^t\}$, which the algorithm FDM generates, an index t_0 such that for $t \geq t_0$ we have the inequalities

(c) There exists in the sequence $\{u^t\}$, which the algorithm FDM generates, an index t_o such that for $t \geq t_o$ we have the inequalities

$$0 \leq F^* - F(u^{t+1}) \leq (1 - \frac{\rho}{n+1})^{n+1} (F^* - F(u^t))$$

(first proved by Schwartz).

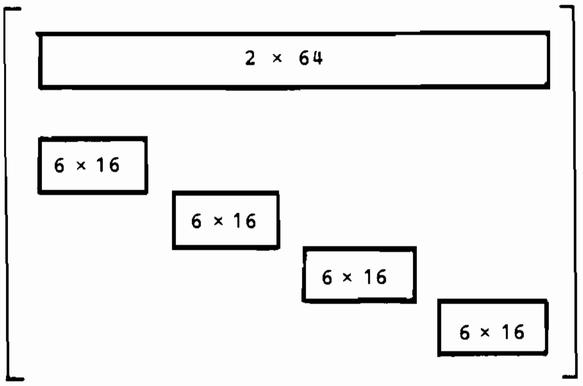
This means:

- If we solve (DRP) exactly, i.e., in the algorithm FDM $\gamma_o = 0$, then the sequence of steps 2 and 3 of the algorithm leads to a near optimal solution of problem *(A)*. Thereby, the algorithm converges. If $\gamma_c > 0$, then convergence may also be proved (see Schwartz [8]).
- At step 3 in the algorithm FDM the objective function increases at least by ε. Thereby, after a finite number of steps 2 and 3 the algorithm comes to step 4. However, we may select the value of ε as we like and thus control the approximation to the optimal value. The results in experiments are encouraging and follow in the sequel.
- The algorithm FDM has a linear rate $q = 1 - \frac{\rho}{n+1}$ of convergence, which explains the observation of slow convergence near the optimum in numerical experiments. Here ρ is the factor that we apply in step 4 to scale down the ε and n is the number of diagonal blocks.

An analogue of theorem 3 for the problem $(\hat{A})$ may also be formulated.

4. RESULTS OF SOME NUMERICAL EXPERIMENTS BY J. KÄSCHEL AND W. REMKE

For the blockangular structure shown below we have computed only small-scale examples. We consider an example with the following size:



If we start the algorithm FDM with different values of ε_o (and $\gamma_o = 0$, $\rho = 0.33$) we get :

Table 1

t	k	ε_k	$F(u^t)$
0	0	200	7,452.28
1	0	200	13,773.40
2	0	200	14,240.49
3	1	66.67	15,188.89
4	2	22.22	15,698.35
5	3	7.41	15,698.35
6	3	7.41	15,715.01

Table 2

t	k	ε_k	$F(u^t)$
0	0	400	7,452.28
1	0	400	7,575.48
2	0	400	9,739.88
3	0	400	12,010.52
4	0	400	15,194.24
5	1	133.3	15,194.24
6	2	44.4	15,373.18
7	2	44.4	15,719.96
8	3	14.8	15,719.96

Table 3

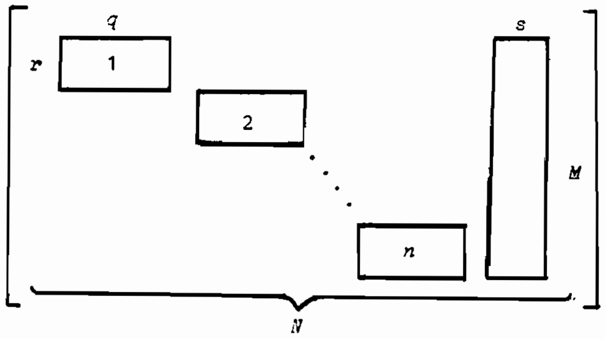
t	k	ε_k	$F(u^t)$
0	0	300	7,452.28
1	0	300	7,475.05
2	1	100	9,346.89
3	2	33.3	10,433.11
4	2	33.3	12,632.88
5	2	33.3	13,579.37
6	2	33.3	13,773.10
7	2	33.3	14,747.19
8	2	33.3	15,284.08
9	2	33.3	15,600.42

Table 4

Iteration number	$F(u^t)$
38	9,231
46	13,627
47	14,287
50	14,796
51	15,648
52	15,721 = optimal value

For comparison purposes we have computed the same example on the same computer with the simplex method. We obtained the results in Table 4.

Next we consider the linked structure



$p \cdot q$ is the size of each diagonal block
s is. the number of coupled variables
M is the total number of rows
N is the total number of columns .

In Table 5 we compare the number of simplex steps, added over all subproblems in the decomposition method (simplex tables with p rows), with the number of simplex steps needed if we apply the simplex method (simplex tables with M rows) to the whole problem.

Table 5 Results for nine examples with linked structures.

n	s	p	q	M	N	Total number of simplex steps, added over all n+1 subproblems, in the decomposition method	Number of simplex steps in the simplex method
3	3	4	10	12	33	35	30
3	3	4	10	12	33	91	24
3	3	4	10	12	33	255	6
4	6	5	20	20	86	355	45
4	6	5	20	20	86	435	82
5	4	4	10	20	54	453	35
6	8	10	40	60	248	548	233
6	8	10	40	60	248	863	151
8	15	20	49	160	407	1,664	505

Table 6 shows the behavior during the solution process for the last example from Table 5 (8 blocks and 15 coupled variables).

Table 6

Decomposition method		Simplex method	
t	$F(u^t)$	Number of iteration t	$F(u^t)$
0	667	420	0
1	722	430	367
2	745	440	608
3	751	450	697
4	756	470	761
5	813.3	490	820
6	813.4	505	843

Our conclusion from the examples is that with the decomposition method we obtain a value near the optimum at an earlier stage of the solution process than with the simplex method. Usually, the first 10 iterations in the decomposition method leads to a solution that differs from the optimal value by no more then 3%. We hope that the same effect will be true for very large LP-problems.

REFERENCES

[1] Beer, K., Lösung grosser linearer Optimierungsaufgaben, VEB Deutscher Verlag der Wissenschaften, Berlin 1977.

[2] Beer, K., J. Käschel, Column generation in quadratic programming, Mathem. Operationsforsch. Statistik, Ser. Optimization, vol. 10 (1979), no.2, pp 179-184.

[3] Geoffrion, A.M., Primal resource-directive approaches for optimizing nonlinear decomposable systems, Oper. Res., vol. 18 (1970), no. 3, pp 375-403.

[4] Käschel, J., Zur numerischen Realisierung von Dekompositionsverfahren der Linearen Optimierung, Berichte der 11. Jahrestagung "Mathematische Optimierung", Vitte (Hiddensee), 1979, pp 87-90.

[5] Käschel, J., Zu ersten numerischen Ergebnissen bei der Realisierung von Dekompositionsverfahren mit Hilfe von PS OPSI, Jahrestagung "Mathematische Optimierung", Vitte (Hiddensee), 1980, pp 55-59.

[6] Lasdon, L.S., Optimization theory for large systems, The Macmillan Co., New York 1970.

[7] Kornai, J., T. Liptak, Two level planning, Econometrica, vol. 33 (1965), no. 1, pp 141-169.

[8] Schwartz, B. Untersuchungen zur Konvergenz der Verfahren der zulässigen Richtungen für Optimierungsaufgaben in endlichdimensionalen Räumen, B-Dissertation an der Technischen Hochschule Karl-Marx-Stadt, 1979.

AN ADVANCED IMPLEMENTATION OF THE DANTZIG–WOLFE DECOMPOSITION ALGORITHM FOR LINEAR PROGRAMMING*

James K. Ho**
Applied Mathematics Department
Brookhaven National Laboratory
Upton, N.Y.

Etienne Loute***
CORE
Université Catholique de Louvain
Louvain-la-Neuve

Since the original work of Dantzig and Wolfe in 1960, the idea of decomposition has persisted as an attractive approach to large-scale linear programming. However, empirical experience reported in the literature over the years has not been encouraging enough to stimulate practical application. Recent experiments indicate that much improvement is possible through advanced implementations and careful selection of computational strategies. This paper describes such an effort based on state-of-the-art, modular linear programming software (IBM's MPSX/370).

The first part of the paper is devoted to a summary of the algorithm and to specifications for an advanced implementation (LP routine, data handling, and computational strategies). A code, named DECOMPSX, meeting these specifications is presented in the second part. Computational experiments on a sample of test problems are reported.

*The authors wish to thank Benoît Culot, who undertook a major part of the programming task described in this paper as a Master's thesis at UCL; M. Benichou, J.M. Gauthier and G. Ribière of IBM France; and Mr. Utens of IBM Belgium, for helpful information and discussion on MPSX/370. The Belgian Ministry of Scientific Policy and the EEC DGXII are gratefully acknowledged for their support, respectively through energy contracts E/I/3 and 339-78-1 EMB, to CORE under the supervision of Y. Smeers.

**The submitted manuscript has been coauthored under contract DE-AC02-76CH0016 with the U.S. Department of Energy. Accordingly, the U.S. Government retains a non-exclusive royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

***Research supported by the Belgian National Research Program in Energy, contract E/I/3.

1. *INTRODUCTION*

Linear programs (LP) with the block-angular structure arise from the optimization of systems consisting of coupled subsystems. For example, the subsystems may be geographical regions each with its own production operations while the coupling is due to common resource constraints. Or the subsystems may be national economies coupled by international trade. In any case, it would be easy to imagine that when the number of subsystems is large or when each subsystem is already very complex, the resulting problems may become very difficult to solve. In 1960, Dantzig and Wolfe [10] introduced the elegant decomposition principle for linear programs. Applied to the block-angular structure, this principle implies that the entire problem can be solved by solving a coordinated sequence of independent subproblems corresponding to the subsystems. Although the process of coordination is shown to be finite, little is known of its convergence properties [1]. Subsequent attempts in implementation, application and refinement ([2], [26], [32], [33]; see [13] for a summary) produced inconclusive and often less than encouraging results in terms of computational efficiency. The status of the decomposition approach, twenty years after its inception, can be summarized as follows.

1) With advances in sparsity techniques for variants of the simplex method (see e.g. [29]), it is unlikely that decomposition can, in general, be significantly more efficient than the simplex approach whenever the latter can produce a solution at an acceptable cost.
2) There is no general purpose, easily accessible, easy to use and computationally robust implementation of decomposition compatible with state-of-the-art LP software. Software manufacturers tend to cater to the average user whose problems are seldom large enough to call for decomposition. In turn, the average user is unlikely to formulate problems too complex for available software.
3) From time to time, users faced with really large problems do attempt decomposition. This is usually done in an ad hoc fashion for the application at hand. Even if it works, there remains always the question of justifying such a nontrivial software project which seldom profits from continuing or even repeated use.

Motivated by Dantzig's insight [12], recent results in large scale techniques for structured LP's ([20], [21]) have identified the following directions for further development of the decomposition approach.

i) The aim of decomposition is not to compete with the simplex approach in its domain of application, but to extend its capability for truly large problems.

ii) For a definitive demonstration of its potential usefulness, it is necessary to attempt implementations at the level of state-of-the-art LP software.

iii) To promote understanding and practice of these concepts, decomposition must be available as a simple to use option in commercial LP codes.

This paper describes an advanced implementation, henceforth called DECOMPSX, of the Dantzig-Wolfe decomposition algorithm. Much as it is a logical progression in our long term research interest, this effort has been precipitated by

a) the growing interest in employing multi-regional and multi-national energy system optimization models ([15], [20]) in energy policy analysis which is expected to give rise to problems exceeding practical limits of existing software; and

b) the advent of modular commercial LP software, such as IBM's MPSX/370 [23], which facilitates a state-of-the-art implementation.

The idea of building upon existing LP software is nothing new. In fact it is the only logical choice because decomposition involves repeated solution of LP's. For example the LP codes LP/90/94, XDLA (ICL), LPM1 (Tomlin) and MPSX (IBM) were used in [2], [32], [20], and [33] respectively. However, the modularity of MPSX/370 allows the full exploitation of its advanced algorithmic features in the design of DECOMPSX.

After a brief summary of the Dantzig-Wolfe algorithm in section 2, the specifications for an efficient implementations in terms of data handling and solution strategies are given in section 3. Section 4 describes the design of DECOMPSX based on MPSX/370. The results on a sample of test problems are included in section 5 for the sole purpose of indicating the robustness of DECOMPSX. Extensive experimental, comparative and benchmark results will be reported in a subsequent paper.

2. *SUMMARY OF THE ALGORITHM*

A block-angular LP with R blocks has the form :

$$\text{minimize} \quad z \equiv \sum_{r=0}^{R} c_r x_r \tag{2.1}$$

$$\text{subject to} \quad \sum_{r=0}^{R} A_r x_r = d_o \tag{2.2}$$

$$B_r x_r = d_r, \; r=1,\ldots R \tag{2.3}$$

$$x_r \geqslant 0 \; , \; r=0,\ldots R \tag{2.4}$$

where c_r is $1 \times n_r$, d_r is $m_r \times 1$ and all other vectors and matrices are of suitable dimensions. The Dantzig-Wolfe algorithm is described very briefly here, primarily to recall concepts assumed to be familiar to the reader and to present notations used in subsequent sections. The reader is referred to [10], [11] and [4] for details.

Assuming that each block of constraints (2.2) and (2.3) is feasible, the m_o constraints in (2.2), called the coupling constraints, will be used to define a master problem which, at cycle k of the algorithm, determines prices π_o^k on these constraints. With these prices, a subproblem from block r at cycle k (SP_r^k) can be defined.

$$SP_r^k \left| \begin{array}{lll} \text{minimize} & v_r^k \equiv (c_r - \pi_o^k A_r)x_r & (2.5) \\ \text{subject to} & B_r x_r = d_r & (2.6) \\ & x_r \geqslant 0 & (2.7) \end{array} \right.$$

If SP_r^k is bounded, then it has an extreme point optimal solution.Otherwise, a solution corresponding to an extreme ray (henceforth called simply an extreme ray solution) is obtained. In either case, denote the solution by x_r^k, and if it passes a test of candidacy to be described shortly, a proposal for the master problem is generated. The proposal from SP_r^k , defined as

$$\begin{pmatrix} p_{rk_r} \\ q_{rk_r} \end{pmatrix} = \begin{pmatrix} c_r \; x_r^k \\ A_r \; x_r^k \end{pmatrix} \tag{2.8}$$

is the $(k_r)^{th}$ proposal submitted by block r through cycle k. Representing all the proposals from block r as a matrix, we have

$$\begin{bmatrix} P_r^{k_r} \\ \\ Q_r^{k_r} \end{bmatrix} \equiv \begin{bmatrix} p_{r1}, \ldots, p_{rk_r} \\ \\ q_{r1}, \ldots, q_{rk_r} \end{bmatrix} . \qquad (2.9)$$

The master problem E^k is then obtained from (2.1) and (2.2) by substituting x_r, r=1, ..., R with convex combinations of the proposals, which by definition are feasible in their respective block constraints (2.3) and (2.4). Hence

$$E^k \left| \begin{array}{lrr} \text{minimize} & z^k \equiv c_o x_o + \sum\limits_{r=1} P_r^{k_r} \lambda_r^{k_r} & (2.10) \\ \text{subject to} & A_o x_o + \sum\limits_{r=1} Q_r^{k_r} \lambda_r^{k_r} = d_o & (2.11) \\ & \delta_r^{k_r} \lambda_r^{k_r} = 1 \ ; \ r = 1, \ldots, R & (2.12) \\ & x_o \geq 0 \ , \ \lambda_r^{k_r} \geq 0 \ ; \ r = 1, \ldots, R & (2.13) \end{array} \right.$$

where $\lambda_r^{k_r} \equiv (\lambda_{r1}, \ldots, \lambda_{rk_r})$ with $\lambda_{r\ell}$ being the weight on the ℓ^{th} proposalfrom block r; and $\delta_r^{k_r} \equiv (\delta_{r1}, \ldots, \delta_{rk_r})$ with $\delta_{r\ell}$ being one if the ℓ^{th} proposal correspondsto an extreme solution, and zero otherwise. Finally, denote the vector of dual variables in E_k by $(\pi_o^k, \sigma_1^k, \ldots, \sigma_R^k)$ where π_o^k corresponds to the coupling constraints (2.11) and σ_r^k to the r^{th} convexity constraint in (2.12).

The algorithm, consisting of three phases, can now be described.

<u>Phase I</u> (For feasibility)

<u>Step 1</u>. (Feasibility of each block)
Solve subproblem SP_r^o for r = 1, ..., R. These are defined in (2.5)-(2.7) with $\pi_o^k = 0$. If one of the subproblems is infeasible, stop : the whole problem is infeasible. Generatea proposal from each subproblem for the master problem E^1.

Step 2. (Feasibility of the coupling constraints)
Use phase II to minimize the sum of infeasibilities in E^1. If the optimal value is positive, stop : the original problem is not feasible. Otherwise, restore original objective function and proceed to phase II.

Phase II (For optimality)

Step 0.

. Set k to 1 if objective function is sum of infeasibilities.

. Set $\underline{z}^{k-1}$, the lower bound on optimal value of objective function, to $-\infty$ if objective function is the original one.

Step 1.
Solve current master problem E^k. If it is unbounded, stop; the original problem is unbounded. Otherwise, send vector of dual variables (π_o^k, σ_r^k) to subproblem r for $r = 1, \dots, R$.

Step 2.
Solve subproblem SP_r^k for $r = 1, \dots, R$.

. If SP_r^k is bounded, denote the optimal solution by x_r^k .

Test of candidacy :

$$\text{if } v_r^k \equiv (c_r - \pi_o^k A_r)\, x_r^k < \sigma_r^k \,, \qquad (2.14)$$

form an extreme point proposal accroding to (2.8).

. If SP_r^k is unbounded, form an extreme ray proposal according to (2.8).

Step 3. (Stopping criteria)

. If no proposal has been generated in step 2, stop; if objective function is sum of infeasibilities, terminate step 2 of phase I . Otherwise, proceed to phase III.

. If all subproblems have been optimized and the objective function is the original one, update lower bound on optimsl value according to :

$$\underline{z}^k = \max\,(\underline{z}^{k-1},\; z^k + \sum_{r=1}^{R} (v_r^k - \sigma_r^k)).$$

If $z^k - \underline{z}^k < \varepsilon$,where ε is some small positive user supplied tolerance proceed to phase III. Otherwise, set k to k+1 and go back to step 1.

Phase III (For reconstruction of a primal solution)

Step 1. (Compute allocation)

Let the optimal values in E^k be $(\hat{z}^k ; \hat{x}_o , \hat{\lambda}_1^k , \dots , \hat{\lambda}_R^{kR})$

Compute $y_r = Q_r^{k_r} \hat{\lambda}_r^{k_r}$, for $r = 1, \dots E$

and define for $r=1, \dots, R$

$$SY_r \left\{ \begin{array}{l} \text{minimize} \quad c_r x_r \\ \text{subject to } A_r x_r = y_r \\ \qquad\qquad B_r x_r = d_r \\ \qquad\qquad\quad x_r \geq 0 \end{array} \right. \tag{2.15}$$

Step 2. (Reconstruction)

Solve SY_r for $\hat{x}_r$, $r = 1, \dots, R$.

Then $(\hat{x}_o, \hat{x}_1, \dots, \hat{x}_R)$ constitute a solution to the original problem with objective value $\hat{z}^k$.

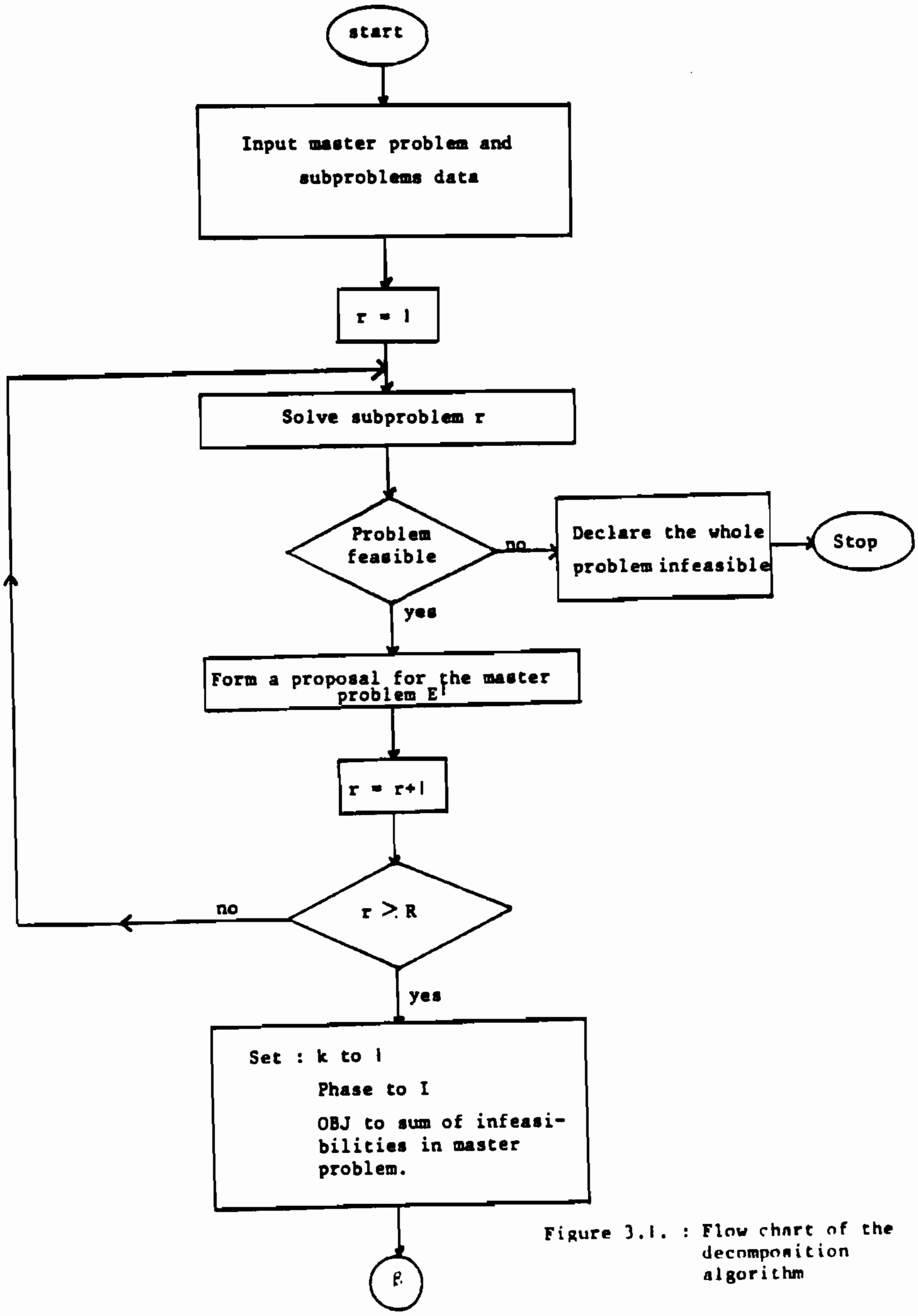

start
Input master problem and subproblems data
r = 1
Solve subproblem r
Problem feasible
no
Declare the whole problem infeasible
Stop
yes
Form a proposal for the master problem E^r
r = r+1
r > R
no
yes
Set : k to 1
Phase to I
OBJ to sum of infeasi-bilities in master problem.
β

Figure 3.1. : Flow chart of the decomposition algorithm

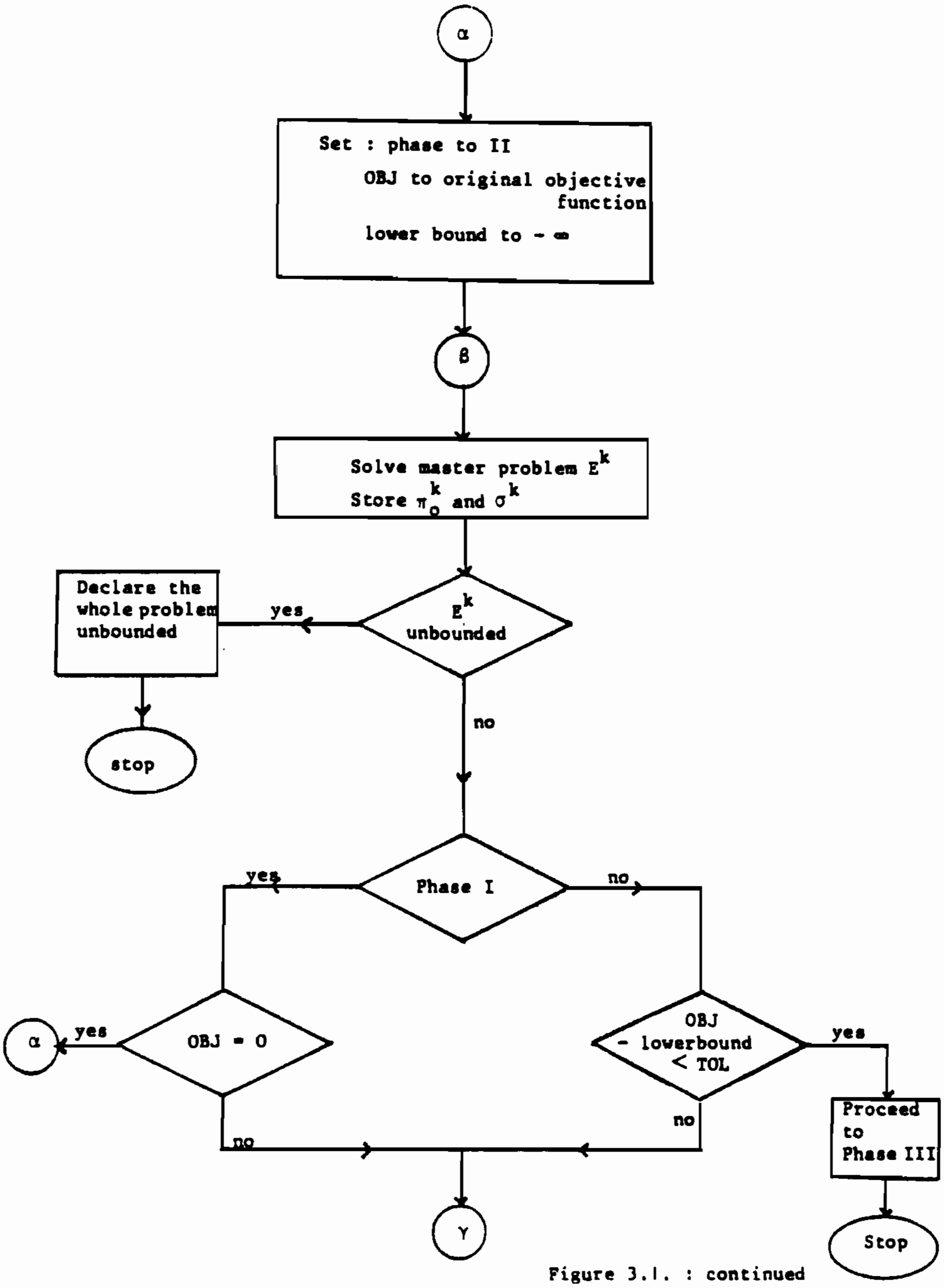


Figure 3.1. : continued

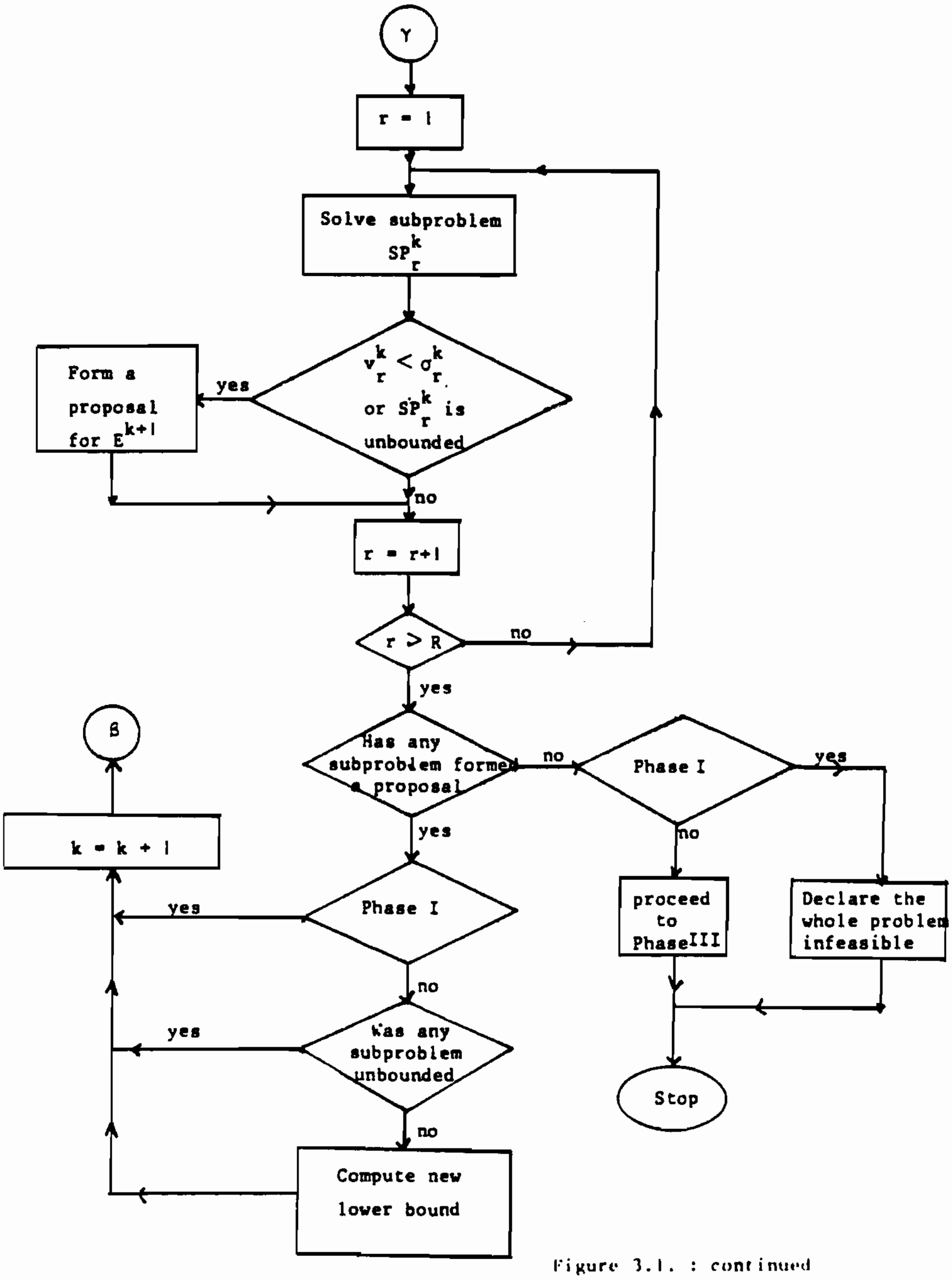

γ
r = 1
Solve subproblem SP_r^k
$v_r^k < \sigma_r^k$ or SP_r^k is unbounded
yes
Form a proposal for E^{k+1}
no
r = r+1
r > R
no
yes
Has any subproblem formed a proposal
no
Phase I
yes
no
proceed to PhaseIII
Declare the whole problem infeasible
Stop
yes
Phase I
yes
no
Was any subproblem unbounded
yes
no
Compute new lower bound
k = k + 1
β

Figure 3.1. : continued

3. *SPECIFICATIONS FOR AN ADVANCED IMPLEMENTATION*

There are three major aspects in the design of a decomposition code :

a) solving the master and subproblems,

b) data handling to update and process the master and subproblem; and

c) computational strategies

3.1. LP routine

Since the master as well as the subproblems are linear-programs, the efficiency of an implementation of the decomposition algorithm depends primarily on the LP routine used. At present, the revised simplex method [9] is still the only approach that has proved to be computationally efficient and robust in practice. By current standards, any efficient revised simplex code would have the following features.

i) Sparse storage : usually a packing scheme is used to store the nonzero elements of matrices by column.

ii) Product form of inverse [29]: the basis inverse is updated by appending an elementary matrix represented by a vector determined by the pivot column.

iii) Basis inversion : the basis is reinverted periodically to maintain sparsity and numerical accuracy.

In addition, an advanced revised simplex code may have most of the following sophisticated features.

iv) Supersparsity [25]: an element pool is used to store only unique values of nonzero elements.

v) Dynamic storage allocation.

vi) Either the Hellerman-Rarick invert procedure [18] or a LU factorization of the inverse [16] (also known as elimination form of the inverse [28]).

vii) Forrest-Tomlin update of LU inverse [14].

viii) Harris 'DEVEX' methods for pivot selection [17].

ix) Accuracy checking by scaling and dynamic tolerances [3].

3.2. Data handling

Access to an advanced LP code does not automatically cover this aspect of the design. In particular, the master and subproblems should be accessible for solution and updating in the internal format for the LP code without conversion to and from intermediate input and output data. This is relatively straightforward if the LP code already has provisions for such data and solution management. Otherwise, a formidable programming task will be required to interface the LP code with commands of the decomposition algorithm. Simply "batching" the individual LP's and revising them in external format would incur so much overhead in data processing as to preclude any hope of overall efficiency in decomposition.

In terms of the generation and utilization of the coordinating information, i.e. prices and proposals, it is possible to specify an efficient design for the data structure of the subproblem independently of the LP code used.

In one of the few implementations of the decomposition algorithm mentioned in the literature, Beale [2] suggests keeping subproblems in the tabular form depicted in figure (3.2.).

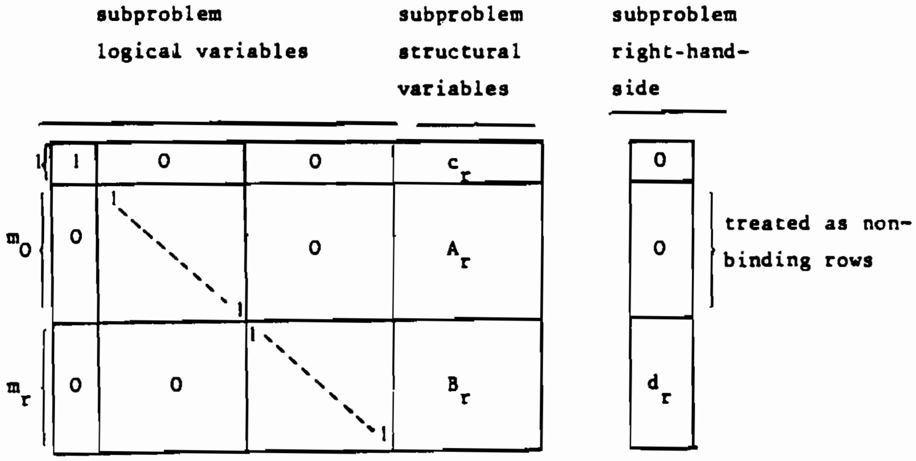

subproblem logical variables
subproblem structural variables
subproblem right-hand-side
1
1
0
0
c_r
0
m_0
0
1
1
0
A_r
0
treated as non-binding rows
m_r
0
0
1
1
B_r
d_r

Figure 3.2. : Tabular representation of subproblem data.

Note that the coupling rows are present in the subproblem. They are declared as nonbinding rows. To simplify notation we omit the subproblem subscript. Note that the first m_o+1 logical variables are always basic because they correspond to nonbinding rows. Therefore a subproblem basis matrix M will exhibit the following structure :

$$\left[\begin{array}{c|c|c} 1 & 0 & \bar{c} \\ \hline 0 & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & \bar{A} \\ \hline 0 & 0 & \bar{B} \end{array}\right]$$

Figure 3.3. : Subproblem basis matrix structure.

The basis inverse is then as in figure 3.4.

$$M^{-1} = \left[\begin{array}{c|c|c} 1 & 0 & -\bar{c}\,\bar{B}^{-1} \\ \hline 0 & \begin{matrix}1 & & \\ & & \\ & & 1\end{matrix} & -\bar{A}\,\bar{B}^{-1} \\ \hline Q & 0 & \bar{B}^{-1} \end{array}\right]$$

Figure 3.4. : Subprotlem basis inverse

Recall from (2.5) that the subproblem objective function at cycle k is :

$$v^k = (c - \pi_o^k A)\ x$$

For data management considerations it is not desirable to compute and use this objective function explicitly. Indeed, introducing new coefficients is not easy with the standard packing scheme used in LP codes. Moreover, one would have to make the distinction in the data structure between the A and B matrices. This would preclude the use of the standard columnwise packing scheme. A last drawback is that one would have to keep a copy of the vector c to construct the cost vector at each cycle.

Generation of the subproblem objective function is facilitated by the subproblem structure in figure 3.2. Recall that the vector of simplex multipliers is obtained by multiplying a unit row vector by the basis inverse. In the case of the subproblem :

$$\underbrace{(1,\ 0,\ \dots\ ,0)}_{m_0\ +\ m_r\ +\ 1\ \text{components}}\ M^{-1} = (\overbrace{1,0,\ \dots,0,}^{m_o+1}\ \overbrace{-\ \bar{c}\ \bar{B}^{-1}}^{m_r\qquad \text{components}}) . \tag{3.1}$$

The 1 x m_r row vector $\bar{c}\ \bar{B}^{-1}$ is the vector of simplex multipliers corresponding to the original objective function ($\pi_o^o = 0$) in the subproblem. If , instead of a unit vector, we take the vector :

$$(1,\ -\ \pi_o^k,\ 0,\ \dots,\ 0), \tag{3.2}$$

and postmultiply by M^{-1}, we get the 1 x (1 + m_o + m_r) row vector :

$$(1,\ -\ \pi_o^k,\ -\ (c\ -\ \pi_o^k\ \bar{A})\ \bar{B}^{-1}) = (1,\ -\ \pi_o^k,\ -\ \pi_r^k) \tag{3.3}$$

so that the vector of simplex multipliers corresponding to the modified objective function is given, modulo a minus sign, by the third term.

The reduced cost of column j of the subproblem is by definition

$$c_j - \pi_o^k A_j - \pi_r^k B_j \qquad (3.4)$$

which is simply the scalar product of the row vector (3.3) by the j^{th} column of the matrix in figure 3.3. Thus with this data structure there is no need to form the objective function explicitly provided that one uses the vector (3.2) to compute the simplex multipliers.

Proposals corresponding to extreme points are generated as follows. The current basic solution is given by premultiplying the right-hand-side vector by the basis inverse. This yields in our case :

$$\begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} = \begin{pmatrix} - \bar{c}\, \bar{B}^{-1} d \\ - \bar{A}\, \bar{B}^{-1} d \\ \bar{B}^{-1} d \end{pmatrix}$$

Now $\bar{x} = B^{-1}d$ is the basic solution of the subproblem, giving all nonzero values of x^k. Therefore, $cx^k = \bar{c}\,\bar{x}$ and $Ax^k = \bar{A}\,\bar{x}$, and the proposal defined in (2.8) is given by the first two components of (3.5), modulo a sign.
Proposals corresponding to extreme rays are generated as follows.
Suppose that column j of the subproblem prices out negatively. To find its update, i.e. its expression in terms of the current basis, we must premultiply by the basis inverse

$$M^{-1} \begin{pmatrix} c_j \\ A_j \\ B_j \end{pmatrix} = \begin{pmatrix} c_j - \bar{c}\, \bar{B}^{-1} B_j \\ A_j - \bar{A}\, \bar{B}^{-1} B_j \\ \bar{B}^{-1} B_j \end{pmatrix}$$

If $\hat{B}_{,} = \bar{B}^{-1} B_j < 0$, then the subproblem is unbounded. An extreme ray y, and n_r x 1 colum vector, is constructed by setting (cf Simmonard [30] , vol 2, pp 9-10) :

$$\begin{cases} y_j = 1 \\ y_s = - \hat{B}_{ji}, \text{ if } x_s \text{ is basic in row } i, \\ y_s = 0, \text{ otherwise.} \end{cases}$$

The extreme ray proposal is then :

$$\begin{pmatrix} cy \\ Ay \end{pmatrix} = \begin{pmatrix} c_j - \bar{c}\,\hat{B}_j \\ A_j - \bar{A}\,\hat{B}_j \end{pmatrix} = \begin{pmatrix} c_j - \bar{c}\,\bar{B}^{-1} B_j \\ A_j - \bar{A}\,\bar{B}^{-1} B_j \end{pmatrix}$$

The data structure proposed above allows very handy generation of extreme point and extreme ray proposals. Moreover it is quite convenient for implementing the phase III procedure. Indeed, forming the phase III subproblems amounts to revising the right-hand-side and declaring the coupling rows as binding in figure 3.2.

3.3. Computational strategies

This third aspect concerns refinements of the basic algorithm aimed at accelarting convergence and reducing core storage requirements and the amount of data handling. Most of these refinements are motivated by heuristics and their benefits cannot be guaranteed on theoretical grounds. However, an advanced implementation should provide such instruments for fine tuning in actual applications.

i) Partial cycles

The convergence of the decomposition algorithm is not lost if in some cycles, only a few of the subproblem s are solved. Such cycles are called partial cycles.If in the course of the algorithm, a few subproblems turn out to be more active in the proposal generation mechanism, it may be useful to treat only these subproblems during a few cycles. Such partial cycles can save CPU time and peripheral activities because less subproblems are set up and solved. Subproblems to be solved in these partialcycles should be selected according to some criterion such as the number of their proposals that have entered the master basis, etc ... Some knowledge of the nature of the problem may indicate subproblems which are more important : for example in a multi-national energy model a subproblem corresponding to a country with an important energy deficit or surplus, is likely to be more active in the algorithmic process. Note also that a subproblem that is not affected by the latest price changes in the master problem needs not be resolved.

ii) multi proposal generation

Since any solution of a subproblem that satisfies the candidacy test (2.14) may lead to an improvement in the master problem, it can be used to generate a proposal. This allows the possibility of obtaining a multitude of proposals from a subproblem during one cycle. Empirical experience ([2], [27]) showed that this strategy tends to accelerate convergence. An example of a mechanism to control proposal generation, used by Ho [19], depends on three factors :

a) frequency control;

b) relative improvement of the reduced cost of the candidate proposal;

and c) total number of proposals generated.

The user specifies four parameters : q_{freq}, q_{perc}, q_{max} and q_{term}. The proposal generation procedure is triggered by the first feasible solution x to the subproblem r that satisfies :

$$\gamma_r = (c_r - \pi_o A_r) x - \sigma_r < 0$$

The corresponding proposal is generated. Then a proposal is generated every q_{freq} iterations or whenever γ_r is decreased by q_{perc}. No more than q_{max} proposals are kept for transmission. The procedure is terminated once q_{term} proposals have been generated. The last q_{max} proposals generated are then sent to the master problem.

When a subproblem is unbounded two companion proposals are generated : one from the extreme ray giving rise to unboundedness and the other from the extreme point corresponding to the current basic feasible solution. Experience indicates that unbounded solutions may occur after only a few iterations. Nonetheless, more proposals can be generated if the column giving rise to unboundedness is fixed at zero and the optimization of the restricted subproblem is pursued until possibly another extreme ray is met, in which case the procedure is repeated.

iii) Proposal purging

As the size of the master problem grows with the addition of new proposals, the inactive ones must be purged periodically to keep storage requirements within acceptable limits.

4. *AN IMPLEMENTATION BASED ON MPSX/370*

From the description of the Dantzig-Wolfe decomposition algorithm, we identify the services required by the algorithm. We now consider how modern LP codes may be used to provide these services.

Clearly, the important logic of the algorithm as well as the manipulation of the prices and proposals cannot be performed by the usual procedures in conventional LP packages. A control procedure must be designed to call the LP routine repeatedly to solve the appropriate problems. Different levels of interface between user written procedures (in FORTRAN, PL/I, etc) and the LP procedures can be considered.

1) User procedures can be called in a control program (acting as a main procedure) written in the control language of the LP package. Communications of data is done by means of files. The MPSX/READCOMM interface [22] is an example.
2) The macro services of the LP code (e.g. primal simplex) can be called in a main procedure written by the user in a high level language. Data is communicated primarily by files. An example is CDC's APEX III [5].
3) Not only macro services, but modular algorithmic tools for LP, e.g. pricing, pivot selection, updating, are user callable. Data can be communicated through arrays or structures rather than files. Moreover, internal computational data for the LP code (the work region) is made accessible. The MPSX/370 LP software offers these possibilities through an extended control language called ECL [23], [31].

The ability to solve and modify LP problems repeatedly within a high level language is a major requirement of the Dantzig-Wolfe algorithm. The control language of traditional mathematical programming systems do not provide enough flexibility. All the power of a high level language, as well as that of an advanced LP code are necessary. As MPSX/370 is the first commercially available LP code that has all the features listed in §3.1, except supersparsity, in addition to provisions for a level 3) interface, it is used as the basis of our implementation.

4.1. The MPSX/370 extended control language (ECL)

The MPSX/370 extended control language (ECL) is PL/I supplemented by a series of macro instructions to be used with the PL/I preprocessor. Examples of the macros are those that declare :

i) the control parameters stored in the communication region (CR);
ii) the MPSX/370 user callable procedures;
iii) PL/I data structures to access internal computational data.

Conditions reflection various status of the problem, such as infeasibility, optimality, and unboundedness, are defined in many MPSX/370 procedures. Whenever a condition is met in a procedure, a demand is issued and control is passed to the macro which is designated to act on that demand. Employing the ON CONDITION facility in PL/I, the user may override the macro defined by the MPSX/370 and substitute other actions on that demand. For example, MPSX/370 terminates the primal simplex algorithm when the condition of unboundedness occurs in the PRIMAL procedure. However, in decomposition, this condition should lead to the generation of an extreme ray proposal if PRIMAL is being used to solve a subproblem.

The MPSX/370 procedures are interfaced with user written ECL procedures by means of a bootstrap procedure called DPLBOOT. It loads the appropriate modules on the first call of any MPSX/370 procedure, thereby establishing all the links necessary for subsequent calls. This linkage is completely transparent to the user.

4.2. Using MPSX/370 procedures to service the master problem

The services required and the MPSX/370 procedures used are listed in Table 4.1. Note that similar procedures exist and are usually callable in other LP codes (e.g. [5]). However, unless data is passed in internal format instead of by external files, such services would not be adequately efficient for an advanced implementation.

We mention here some adjustements that are necessary. Empirical experience indicates that the scaling procedure used by MPSX/370 is inappropriate for the proposal data,which may contain elements with large magnitudes relative to the unit element in the convexity row. Therefore, a separate external procedure will be used for this purpose. Also, all the computations are performed in double precision (8 bytes) in MPSX/370 procedures. However the data is kept on a file, named PROBFILE, by packing each nonzero element with its row index in 8 bytes : 6 bytes for the coefficient and 2 bytes for the index. This intermediate precision limits the accuracy of the coordination between the master problem and the subprcblems. Default tolerances have been adjusted in order to avoid numerical difficulties when solving the master problem.

Service	Frequency	MPSX/370 procedure
Reading and checking the data	1	CONVERT
Adding convexity rows	1	REVISE
Adding new proposals	cycle	REVISE
Purging unprofitable proposals	cycle	REVISE
Setting up an scaling the problem	cycle	SETUP
Saving and restoring the basis	cycle	SAVE/RESTORE
Solving the problem	cycle	PRIMAL or OPTIMIZE
Saving the dual solution	cycle	SOLUTION
Setting up right-hand-side for sub-problems in Phase III	1	POSTMUL

Table 4.1 MPSX/370 procedures serving the master problem.

4.3. Using MPSX/370 procedures to service the subproblems

The services required and the MPSX/370 procedures used are listed in Table 4.2. Three of these, and two others called by them, have to be modified for this purpose.

In order to use the implicit form of the objective function as described in § 3.2, the following MPSX/370 procedures involving the objective value require modification :

FORMC, which initializes the vector used to compute simplex prices;
FIXVEC, which computes the current solution after a reinversion;
FTRANU, which updates candidate columns; and
ELIMINLU, which updates the problem at the end of a simplex iteration.

The MPSX/370 procedure PRIMAL is used to solve a subproblem. It has to be modified to incorporate the mechanism for proposal generation described in § 3.3. The tests for proposal generation are performed after a major simplex iteration to benefit from multiple pricing. If one of the tests is passed, the demand XDOFREQ3 is issued. The corresponding action is to copy the proposal to a buffer for the next revision of the master problem. The execution of PRIMAL is then resumed. A similar action is taken when the XDOUNB demand is issued in PRIMAL, signalling an unbounded solution. In this case however, PRIMAL is resumed after copying the proposal to the buffer and fixing the column that caused unboundedness to zero value.

To summarize, five MPSX/370 procedures have been modified. About 120 assembler statements have been added. These modifications were easily implemented with the aid of the MPSX/370 logic manual [5], as well as comments in the source code. We could have chosen not to modify the MPSX/370 source at the cost of :

1) revising the data of the objective row at each cycle (using REVISE on the problem file or better still, MODIFY on the work matrix);
2) less efficient implementation of the multiproposal generation scheme (more frequent calls to and returns from PRIMAL; and interruption of major iterations of the multiple pricing strategy).

All the modifications are fully described in [6].

Service	Frequency	MPSX/370 procedure
Reading and checking the data	1	CONVERT
Setting up the problem	cycle	SETUP
Forming the objective row	iteration†	FORMC*
Multiproposal generation tests	iteration†	PRIMAL*
Forming an extreme point proposal	several times/cycle	current solution in work region
Forming an extreme ray proposal	several times/cycle	FTRANL followed by FTRANU *
Solving the subproblem	cycle	PRIMAL*
Saving and restoring basis	cycle	SAVE and RESTORE
Setting up the subproblem for Phase III	1	MODIFY (on work matrix)
Solving the subproblem in Phase II	1	PRIMAL* or OPTIMIZE

Table 4.2. MPSX/370 procedures serving the subproblems.

* These MPSX/370 procedures have been modified to service the subproblems

† A simplex iteration

4.4. Organization of DECOMPSX

We are now ready to describe briefly DECOMPSX, an implementation of the Dantzig-Wolfe decomposition algorithm coded in PL/I, with major services provided by MPSX/370. A full description can be found in [7] and [8].

DECOMPSX consists of 17 external procedures (totaling about 2000 instructions) and 2 libraries of macros. One library is that of MPSX/370 ECL macros and the other contains 9 declaration macros with 300 instructions. The name, function, frequency of use, calling procedure and procedures called for the major external procedures of DECOMPSX and MPSX/370 are summarized in Table 4.3.

The file organization is illustrated in Figure 4.1. Table 4.4. lists the type, purpose and frequency of use for each file. For the master problem, two problem files are used in a flip-flop manner. Repeated problem revisions lead to a waste of storage space. When the current problem file is about to overflow; it is compressed while copied to the other file.

DECOMPSX has been designed to handle up to 99 subproblems with up to 1000 coupling rows in the master problem. Both limits could be easily changed but the proposal buffer storage requirements may become prohibitive for a large number of coupling rows. The subproblems could theoretically have up to 16000 rows, but a practical limit is in the range of 2000-4000 rows.

DECOMPSX external procedure	function	frequency of use	called by	major calls to	
				DECOMPSX procedures	MPSX/370 procedures
CONTROL	main procedure	1	-	SAVREST INITPAR MASTER PHASE3	CONVERT
INITPAR	initializes controls and parameters	1	CONTROL	SAVREST	-
SAVREST	saves and restores a problem	1	CONTROL INITPAR	-	-
MASTER	driver procedure for phase I and II of decomposition algorithm	1	CONTROL	OPTZE POLICY SETSUBP SOLVSUB PURGE GARBCOL PROPSCL	ASSIGN COPY CRASH FREECORE RESTORE REVISE SETUP
PROPSCL	computes a scaling factor for proposals based on first batch of proposals	1	MASTER	-	-
POLICY	selects subproblems to be treated in a cycle	cycle	MASTER	-	-
SETSUBP	performs first set up of suproblems, creates tables for price transfers between master and subproblems	#subproblems	MASTER	-	INQUIRE SETUP SELIST

Table 4.3. DECOMPSX external procedures.

DECOMPSX external procedure	function	frequency of use	called by	major calls to	
				DECOMPSX procedure	MPSX/370 procedure
OPTZE	solves the master problem, solves the subproblems in phase III.	cycle + # subproblems	MASTER PHASE3	-	CRASH RESTORE DUAL SAVE INVERT SOLUTION PRIMAL
SOLVSUB	solves subproblems in Phase I and II	cycle	MASTER	PURGE	FTRANL1 FTRANU1 GETVEC1 INVALUE PRIMAL MODIFY SAVE
GARBCOL	purges unprofitable proposals from master problem file	n cycles	MASTER	-	SELIST BTRANU1 BTRANL1 PREMUL
PURGE	purges the proposal buffer on an auxiliary file when it is full	cycle	MASTER SOLVSUB	-	-
PHASE3	driver procedure for Phase III of decomposition algorithm	1	CONTROL	OPTZE FILESOL	INVALUE SETUP MODIFY RESTORE POSTMUL SELIST
FILESOL	files the solution in standard MPS format	# subproblems	PHASE 3	-	SOLUTION

Table 4.3 (Continued)

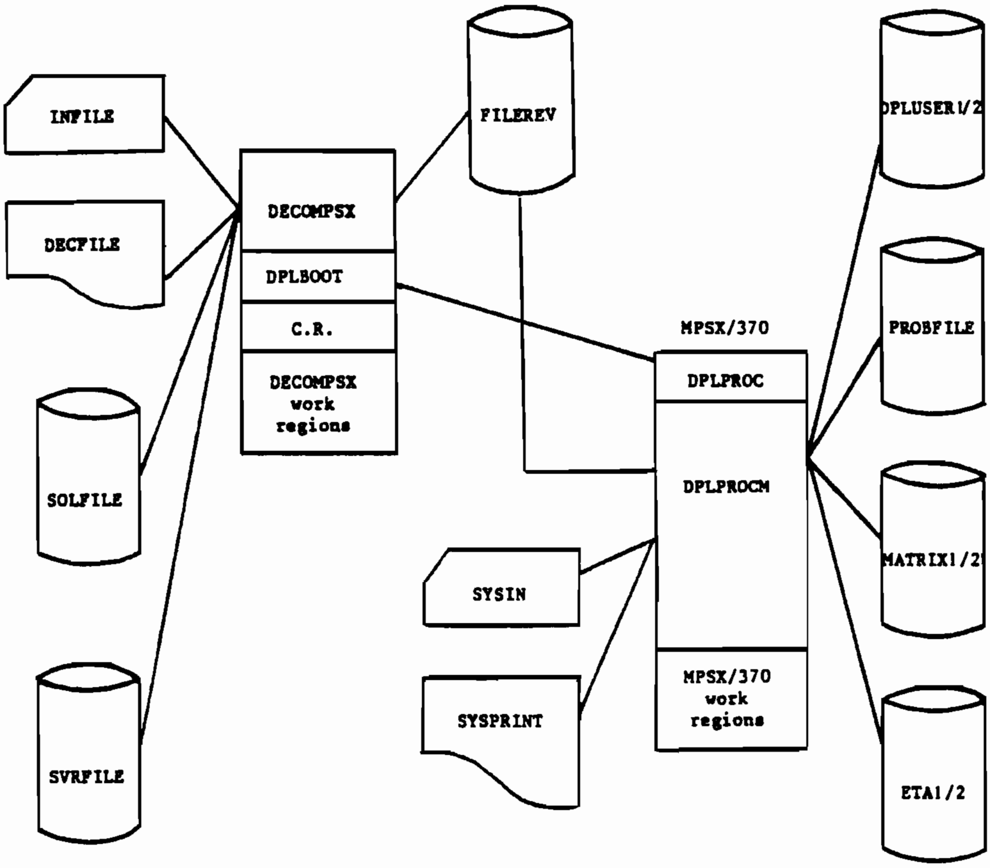


Figure 4.1. File organization of DECOMPSX.

File	type	purpose	frequency of use
INFILE	sequential	used to specify tha master problem and sub-problems characteristics, and control parameters	1
DECFILE	sequential	iteration log for the decomposition algorithm	cycle
SOLFILE	sequential	solution file in standard MPS format	1
SURFILE	sequential	save and restore file for the decomposition algorithm	1
FILEREV	sequential	file used to keep proposal data for master problem revise	cycle
SYSIN	sequential	file containing master problem and subproblem data	1
SYSPRINT	sequential	iteration log of the simplex algorithm	simplex iteration
DPLUSER1/2	direct access	problem files for master problem, used in a flip-flop way	cycle
PROBFILE	direct access	problem file for subproblems	cycle
MATRIX1/2	direct access	work matrix files (internal file of MPSX/370)	cycle x problems
ETA1/2	direct access	eta files (U^{-1} and L^{-1}) of basis inverse representation (internal file of MPSX/370)	cycle x problems

Table 4.4. Files used by DECOMPSX.

4.5. Input data for DECOMPSX

For a problem with R blocks, the data for the master and subproblems comprise R+1 contiguous, complete data sets in the MPS format [23]. In the row section of each subproblem, the coupling rows that are active (i.e. having non zero coefficients in that subproblem) must be declared first, along with the objective row. The data partitioning is very natural and allows independent generation of the subproblems as long as the names and units of the common constraints are consistent.

It remains to specify certain control parameters for DECOMPSX. The most important ones are the strategic parameters which,

i) indicate whether the master or the subproblems are to be solved first in the first cycle;

ii) indicate whether the proposal buffer is to be shared by all subproblems (and copied on to a file when overflow occurs) or to be used by one subproblem at a time;

iii) indicate whether subproblems that have not been sending proposals are to be skipped;

iv) specify the percentage improvement, frequency and total number generated used in the mechanism for multi-proposal generation;

v) determine whether the prices from the master problem have changed significantly to call for resolution of individual subproblems; and

vi) control the printing and filing of the solution.

5. *SAMPLE OF EXPERIMENTS*

We have experimented with DECOMPSX on a series of test problems. Table 5.1. summarizes the statistics of the test problems. All runs were performed under VM/CMS on an IBM/370 model 158. A 1000 K virtual machine was used for the runs. Table 5.2. indicates the number of cycles of the decomposition algorithm and the total CPU time in minutes. In Table 5.3. we indicate the CPU time in the major MPSX/370 procedures during phases I and II : SETUP, REVISE and PRIMAL for the master problem, SETUP and PRIMAL for the subproblems. The last column gives the total of the preceeding five and is to be compared with the last column of Table 5.2. The difference is the processing time needed for converting the problem data, for prices and proposals transfer and for the reconstruction phase (phase III).

We conclude with a few remarks on the computational results presented in Tables 5.2. and 5.3. From Table 5.2., we note that the convergence of the algorithm depends on the problem as much as on its dimensions. It is therefore important to identify convergence properties for various classes of applications. The problems used include all the truly block-angular models we have access to at this writting as well as models that can be transformed somewhat artificially, to have this structure. For that reason we believe that DECOMPSX is robust. It can be observec from Table 5.3. that only a small portion of the total CPU time is spent on DECOMPSX external procedures. The rest is taken up by MPSX/370 simplex operations. This justifies our claims to an efficient implementation. It also means that future improvement would have to come primarily from strategie manipulations aimed at reducing the number of cycles. The omission of any comparative performance data is deliberate. This is because not all the problems tested are truly block-angular. Some are decomposable in other, possibly more efficient ways. Sketchy comparative statistics would therefore be most misleading. Instead, the detailed results of experiments and comparaisons will be reported in a subsequent paper.

	Problem name	SCORPION	FIXMAR	FORESTRY	FOREST	REGENT	ORESTE	DYNA	BLDYOO
Number of	(rows, columns)	(389,747)	(325,758)	(402,1006)	(521,1123)	(1299,3924)	(1537,3373)	(4711,10231)	(5904,12054)
	coupling rows	53	18	11	19	45	39	596	594
	subproblems	6	4	6	2	9	3	5	6
	non-zero elements	2097	2925	4197	2652	38839	6238	21274	25196
	Density (%)	0.7	1.2	1.0	0.45	0.7	0.12	0.04	0.03
	Initial master problem	(59,59) 1.7	(22,23) 4.0	(17,18) 5.9	(21,22) 4.76	(54,369) 4.9	(42,43) 2.38	(601,1082) 0.42	(598,1044) 0.38
Number of (rows, columns) and density	Subproblem 1	(109,165) 1.7	(110,206) 3.8	(84,187) 3.8	(261,522) 0.85	(183,433) 4.7	(537,1150) 0.34	(943,1952) 0.20	(1588,2614) 0.12
	" 2	(109,170) 2.2	(91,185) 3.3	(56,127) 7.7	(281,622 0.85	(185,442) 4.5	(537,1150) 0.34	(943,1952) 0.20	(992,1929) 0.19
	" 3	(109,175) 2.2	(75,200) 2.7	(80,189) 5.6		(185,442) 4.5	(541,1154' 0.33	(943,1952) 0.20	(988,1925) 0.20
	" 4	(109,175) 2.2	(103,221) 5.0	(83,217) 4.0		(185,442) 4.5		(943,1952) 0.20	(982,1919) 0.20
	" 5	(109,165) 2.1		(74,162) 7.2		(185,442) 4.5		(943,1952) 0.20	(977,1914) 0.20
	" 6	(109,165) 2.1		(80,177) 4.6		(184,439) 4.6			(973,1910) 0.20
	" 7					(184,439) 4.6			
	" 8					(184,439) 4.6			
	" 9					(185,442) 4.5			
	Average number of rows in a sub-problem	109	95	76	271	184	538	943	1083

Table 5.1. Dimensions of the test problems.

Problem name	number of cycles	Total CPU time (minutes)
SCORPION	6	1.30
FIXMAR	22	2.35
FORESTRY	12	2.35
FOREST	7	1.30
REGEMT	6	5.65
ORESTE	12	4.24
DYNA	25	38.46
BLDYOO	21	71.85 (†)

(†) All problems are stopped with primal-dual gap less than 0,01 % except the last one (9 %).

Table 5.2. Solution statistics with DECOMPSX.

Problem name	CPU time (minutes)					Total
	Master			Subproblems		
	SETUP	REVISE	PRIMAL	SETUP	PRIMAL	
SCORPION	0.01	0.01	0.09	0.20	0.54	0.85
FIXMAR	0.16	0.05	0.32	0.34	0.94	1.81
FORESTRY	0.06	0.08	0.15	0.33	1.21	1.83
FOREST	0.05	0.01	0.05	0.17	0.69	0.97
REGEMT	0.03	0.04	0.11	2.13	1.29	3.60
ORESTE	0.01	0.04	0.11	0.83	2.41	3.40
DYNA	0.94	1.19	9.58	4.17	16.97	32.85
BLDYOO	1.32	0.85	14.15	4.47	43.25	64.44

Table 5.3. Time spent in MPSX/370 procedures.

References

[1] Alder, I., Ülkücü, A., "On the number of iterations in Dantzig-Wolfe decomposition", in *"Decomposition of Large Scale Problems"*, ed. D.M. Himmelblau, North-Holland/American Elsevier, Amsterdam-New York, 1973, pp. 181-187.

[2] Beale, E., Hughes, P. and R. Small, "Experiences in using a decomposition program", *Computer Journal,* 8, pp. 13-18, 1965.

[3] Benichou et al., "The efficient solution of large-scale linear programming problems-some algorithmic techniques and computational results", *Mathematical Programming,* 13, 1977, pp. 280-332.

[4] Broise, P., Huard, P. and Sentenac, J., *"Decomposition des Programmes Mathématiques",* Dunod, Paris, 1968.

[5] CDC, "APEX-III Reference Manual", Publication n° 76070000, 1974.

[6] Culot, B., Loute, E. and Ho, J.K., "DECOMPSX user's manual", (in French), CORE computing report 80-B-01, C.O.R.E., Université Catholique de Louvain, Belgium, 1980.

[7] Culot, B., Loute, E., "DECOMPSX system manual", (in French), CORE computing report 80-B-02, C.O.R.E., Université Catholique de Louvain, Belgium, 1980.

[8] Culot, B., Loute, E., "DECOMPSX source", CORE computing report 80-B-03, C.O.R.E., Université Catholique de Louvain, Belgium, 1980.

[9] Dantzig, G.B., *"Linear Programming and Extensions",* Princeton University Press, Princeton, 1963.

[10] Dantzig, G.B., and P. Wolfe, " Decomposition Principle for Linear Programs", *Operations Research,* vol. 8, pp. 101-11, 1960.

[11] Dantzig, G.B. and P. Wolfe, "The Decomposition Algorithm for Linear Programs", *Econometrica,* 29, pp. 767-778, 1961.

[12] Dantzig, G.B., "On the need for a system optimization laboratory", in *"Optimisation Methods for Resource Allocation",* eds. R. Cottle, J. Krarup, Crane-Russack, New York, 1974, pp. 3-24.

[13] Dirickx, Y.M.I., Jennergren, L.P., *"Systems Analysis by Multilevel Methods",* Wiley, New York, 1979.

[14] Forrest, J. and J. Tomlin, "Updating Triangular Factors of the Basis in the Product Form Simplex Method", *Mathematical Programming,* 2, pp. 263-278, 1972.

[15] Geottle, R., Cherniavsky, E. and R. Tessmer : "An Integrated Multi-Regional Energy and Interindustry Model of the United States", BNL 22728, Brookhaven, National Laboratory, New York, May 1977.

[16] Gustavson, F.G., "Some basic techniques for solving sparse systems of linear equations", in *"Sparse Matrices and their Applications*, eds. D.J. Rose, R.A. Willoughby, Plenum, New York, 1972, pp. 41-52.

[17] Harris, P., "Pivot Selection Rules of the Devex LP Code", Mathematical Programming Study n° 4, "Computational Practice in Mathematical Programming", 1975, pp. 30-57.

[18] Hellerman, E., Rarick, D., "The partitioned preassigned pivot procedure (P^4)", in *"Sparse Matrices and their Applications"*, eds. D.J. Rose, R.A. Willoughby, Plenum, New York, 1972, pp. 67-76.

[19] Ho, J.K., "Implementation and Application of a Nested Decomposition Algorithm", in *"Computers and Mathematical Programming"*, ed. W.W. White, NBS 1978, pp. 21-30.

[20] Ho, J.K., Loute, E. Smeers, Y. and E. Van de Voort, "The Use of Decomposition Techniques for Large Scale Linear Programming Energy Models", *Energy Policy*, special issue on "Energy Models for the European Community", ed. A. Strub, June 1979, pp. 94-101.

[21] Ho, J.K. and Loute, E., "A comparative study of two methods for staircase linear programs", ACM Transactions on Mathematical Software 6 (1980)17-30.

[22] IBM, "Mathematical Programming System Extended (MPSX) Read Communication Format (READCOMM) Program Description Manual". SH20-0960-0, January 1971.

[23] IBM, "IBM Mathematical Programming System Extended/370 (MPSX/370) Program Reference Manual", SH19-1095-3, December 1979.

[24] IBM, "IBM Mathematical Programming System Extended/370 (MPSX/370) Logic Manual", (licensed material) LY19-1024-0, January 1975.

[25] Kalan, J.E., "Aspects of large-scale in core linear programming", in *Proceedings of A.C.M. Annual Conference*, pp. 304-313, 1971.

[26] Kutcher, G.P., "On the decomposition price endogenous models", in *"Multi-level Planning : Case Studies in Mexico"*. eds. L.M. Goreux, A.S. Manne, North Holland/American Elsevier, Amsterdam/New York, 1973, pp. 499-519.

[27] Lasdon, L.S., *"Optimization Theory for Large Systems"*, McMillan, London, 1970.

[28] Markowitz, H.M., "The elimination form of the inverse and its application to linear programming", *Management Science*, 3, 1957, pp. 255-269.

[29] Orchard-Hays, W., *"Advanced Linear-Programming Computing Techniques"*, McGraw-Hill, New York, 1968.

[30] Simmonard, M., *"Programmation Linéaire"*, vol. 1 et 2, Dunod, Paris, 1972.

[31] Slate, L. and Spielberg, K., "The extended control language of MPSX/370 and possible applications", *IBM Systems Journal,* 17, 1978, pp. 64-81.

[32] Von Schiefer, G., "Lösung grosser linear Regionalplannungsprobleme mit der Methode von Dantzig and Wolfe", *Zeitschrift für Operations Research,* 20, pp. B1-B16, 1976.

[33] Williams, H.P. and Redwood, A.C., "A structured programming model in the food industry", *Operational Research Quaterly,* 25, 1974, pp. 517-527.

ON A DECOMPOSITION PROCEDURE FOR DOUBLY COUPLED LP'S

J. Stahl

Institute for Applied Computer Techniques
Budapest

The first part of this paper presents a decomposition procedure for solving the doubly coupled LP(1). Problem (1) is solved by treating the two master problems (3), (4) and the subproblem (5). The resulting procedure is not a generalization of either Dantzig–Wolfe (D–W)'s or Benders' well-known decomposition algorithms. Though the method can be derived using ideas similar to those of the D–W decomposition procedure, neither the D–W nor the Benders' algorithm is a special case. Nevertheless there are cases where the same algorithm can be obtained from our procedure as from D–W decomposition and Benders' algorithm respectively. The second part of the paper discusses the possibility of applying a dynamic scaling strategy to solve the master problems.

INTRODUCTION

An LP of the following form

$$\begin{aligned} &By + \sum_i A_i x_i \leq b \\ &D_i y + B_i x_i \leq b \qquad\qquad (i=1,2,...) \\ &y, x_1, x_2, ... \geq 0 \\ &\max\ (cy + \sum_i c_i x_i) \end{aligned}$$

can be the mathematical model of several interesting economic problems. For example, the model of a system consisting of several subsystems, or the model of a system over several periods, may be of this form. In the first case x_i represents the subsystems' activities. The coupling variable y can be interpreted as a centrally controlled activity of direct influence on the subsystems' activities, and the first group of constraints (coupling constraints) may be balances. An obvious interpretation at the same level can also be given for the second case.

Keeping in mind these interpretations or thinking of the Dantzig-Wolfe and/or Benders decomposition procedure, a decomposition scheme is suggested, where the subproblems are of the form

$$\begin{aligned} B_i x_i &\le b_i - D_i \tilde{y} \\ x_i &\ge 0 \\ \max\ (c_i - \tilde{p} A_i)\, x_i & \quad . \end{aligned}$$

Here the multipliers $\tilde{p}$ assigned to the coupling rows and the values $\tilde{y}$ assigned to the coupling variables can be obtained by solving master problems similar to that of the mentioned procedures. A more formal derivation of such a procedure can also be given (Stahl 1976).

In Section 1 of the present paper we summarize the procedure. It consists mainly of earlier results that have been published mostly in Hungarian (Stahl 1978). Section 2 deals with a variant of the procedure. Since we have at present little computational experience , we consider our paper as dealing with *possibilities* to increase the scope and effectiveness of decomposition.

1. DECOMPOSITION PROCEDURE FOR DOUBLY COUPLED LPs

Let us consider the following LP problems:

$$\begin{aligned} A_{01} x_1 \qquad\qquad &\le b_0 \\ A_{10} x_0 + A_{11} x_1 + A_{12} x_2 &\le b_1 \\ A_{21} x_1 + A_{22} x_2 &\le b_2 \\ x_0 , x_1 , x_2 \qquad &\ge 0 \\ \max\ (c_0 x_0 + c_1 x_1 + c_2 x_2) & \end{aligned} \tag{1}$$

and its dual

$$\begin{aligned} p_1A_{10} \qquad\qquad &\geq c_0 \\ p_0A_{01} + p_1A_{11} + p_2A_{21} &\geq c_1 \qquad\qquad (2) \\ p_1A_{12} + p_2A_{22} &\geq c_2 \\ p_0, p_1, p_2 \qquad &\geq 0 \\ \min\ (p_0b_0 + p_1b_1 + p_2b_2) & \end{aligned}$$

To simplify the notation we did not take into account explicitly the possible block-diagonal structure of A_{22}.

Now we assume that the polyhedral sets

$$X_1 = \{x_1 : A_{01}x_1 \leq b_0,\ x_1 \geq 0\}$$

and

$$P_1 = \{p_1 : p_1A_{10} \geq c_0,\ p_1 \geq 0\}$$

are bounded.

To solve (1) and (2) we use the following iterative procedure.

In each step, one has to solve two master problems. The first is an LP problem of the form

$$\begin{aligned} A_{10}x_0 + \sum_i \lambda_i(A_{11}\tilde{x}_{1i} + A_{12}\tilde{x}_{2i}) + \sum_j \mu_j A_{12}\tilde{\tilde{x}}_{2j} &\leq b_1 \\ \sum_i \lambda_i &= 1 \qquad\qquad (3) \\ x_0, \lambda_i, \mu_j &\geq 0 \\ \max\ (c_0x_0 + \sum_i \lambda_i(c_1\tilde{x}_{1i} + c_2\tilde{x}_{2i}) + \sum_j \mu_j c_2\tilde{\tilde{x}}_{2j}) & \end{aligned}$$

where $(\tilde{x}_{1i},\tilde{x}_{2i})$ is an element of the set

$$X = \{(x_1,x_2) : A_{01}x_1 \le b_0,\ A_{21}x_1 + A_{22}x_2 \le b_2, x_1,x_2 \ge 0\}$$

and $\tilde{\tilde{x}}_{2j}$ is an extremal direction of the set

$$X_2 = \{x_2 : A_{22}x_2 \le 0,\ x_2 \ge 0\} \quad .$$

The problem does not necessarily contain λ and μ variables, and in the first case there is no need for the equality constraint. Due to the boundness of X_1, the set

$$\{(x_1,x_2) : A_{01}x_1 \le b_0,\ A_{21}x_1 + A_{22}x_2 \le 0,\ x_1,x_2 \ge 0\}$$

can be replaced by X_2.

The other master problem is similar and of the form

$$\begin{aligned} p_0A_{01} + \sum_i \sigma_i(\tilde{p}_{1i}A_{11} + \tilde{p}_{2i}A_{21}) + \sum_j \tau_j\tilde{\tilde{p}}_{2j}A21 &\ge c_1 \\ \sum_i \sigma_i &= 1 \qquad (4) \\ p_0,\sigma_i,\tau_j &\ge 0 \\ \min\ (p_0b_0 + \sum_i \sigma_i(\tilde{p}_{1i}b_1 + \tilde{p}_{2i}b_2) + \sum_j \tau_j\tilde{\tilde{p}}_{2j}b_2) \end{aligned}$$

where $(\tilde{p}_{1i},\tilde{p}_{2i})$ is an element of the set

$$P_1 = \{(p_1,p_2) : p_1A_{10} \ge c_0,\ p_1A_{12} + p_2A_{22} \ge c_2, p_1,p_2 \ge 0\}$$

and the $\tilde{\tilde{p}}_{2j}$s are extremal directions of the set

$$P_2 = \{p_2 : p_2A_{22} \ge 0,\ p_2 \ge 0\} \quad .$$

We call these problems the p_1-problem and the x_1-problem respectively.

Again, due to the boundness of X_1 and P_1, both the masters always have feasible solutions. It is easy to see that, if a p_1-problem (an x_1-problem) is unbounded, then problem (2) (problem 1) has no feasible solution. In such a case the procedure terminates.

Let $\tilde{\phi}$ be the optimal value of the actual p_1-problem if this problem contains a λ variable and $-\infty$ otherwise. Similarly $\tilde{\Psi}$ is the optimal value of the actual x_1-problem or ∞.

If $\tilde{\phi} = \tilde{\Psi}$, the procedure terminates, otherwise let $(\tilde{p}_1,\tilde{\pi}_0)$ and $(\tilde{x}_1,\tilde{\xi}_0)$ denote optimal solutions to the duals of the master problems. Now to complete the iteration steps, one has to deal with the subproblem

$$\begin{aligned} A_{22}x_2 &\leq b_2 - A_{21}\tilde{x}_1 \\ x_2 &\geq 0 \\ \max(c_2-\tilde{p}_1A_{12})x_2 & \end{aligned} \tag{5}$$

If this problem has no feasible solution one must introduce a new τ variable into problem (4). This variable corresponds to that extremal element $\tilde{\tilde{p}}_2$ of P_2 for which $\tilde{\tilde{p}}_2(b_2-A_{21}\tilde{x}_1) < 0$.

Similarly, if (5) is unbounded, a new μ-variable will be introduced into problem (3). The variable corresponds to that extremal element $\tilde{\tilde{x}}_2$ of X_2, for which $(c_2-\tilde{p}_1A_{12})\tilde{\tilde{x}}_2 > 0$.

If (5) has an optimal solution then one has to augment the master problems with a λ or σ variable corresponding to $(\tilde{x}_1,\tilde{x}_2)$ and $(\tilde{p}_1,\tilde{p}_2)$ respectively, where $\tilde{x}_2$ and $\tilde{p}_2$ are extremal optimal solutions to problem (5) and its dual, respectively.

In any case, one has to begin a new iteration step.

The following theorem contains the main features of the procedure.

THEOREM 1. *If (1) has no optimal solution, the procedure terminates after a finite number of steps. Otherwise the values of $\tilde{\Phi}$ and $\tilde{\Psi}$ obtained converge to the optimal value (1).*

If problem (1) has an optimal solution, then the procedure may be finite or infinite. If $\tilde{\Phi} = \tilde{\Psi}$, the procedure can indeed be completed,since from the actual p_1-problem and x-problem one can easily obtain feasible solutions to (1) and to the dual problem (2) such that the corresponding objective values are $\tilde{\Phi}$ and $\tilde{\Psi}$ respectively. Both of the sequences of the $\tilde{\Phi}$ and $\tilde{\Psi}$ values are obviously monotonic.

Now we omit the assumptions about the boundness of X_1 and P_1. (However, with respect to the economic interpretation which was given, these are in any event not too restrictive in some important practical cases.)

Let us augment problem (1) in the following way

$$\begin{array}{rcl} e\,x_1 & \leq & \beta \\ A_{01}x_1 & \leq & b_0 \\ -\xi e + A_{10}x_0 + A_{11}x_1 + A_{12}x_2 & \leq & b_1 \\ A_{21}x_1 + A_{22}x_2 & \leq & b_2 \\ \xi, x_0, x_1, x_2 & \geq & 0 \\ \max(-\gamma\xi + c_0x_0 + c_1x_1 + c_2x_2) & & \end{array} \qquad (6)$$

where each of the components of the e vectors is unity,and β and γ are non-negative.

Now we fix $\varepsilon>0$ arbitrarily and choose two sequences $\{\beta_h\}$ and $\{\gamma_h\}$ in such a way that $0 < \beta < \beta_2 < \ldots, 0 < \gamma < \gamma_2 < \ldots,$ $\beta_h \to \infty$, $\gamma_h \to \infty$.

For a fixed pair (β_h, γ_h), one can apply the procedure for solving (6). After a finite number of steps one of the following cases will occur:

(a) (6) has no feasible solution;

(b) the dual problem of (6) has no feasible solution;

(c) one has a feasible solution $(\xi_h, x_{0h}, x_{1h}, x_{2h})$ to (6), a feasible solution $(\pi_h, p_{0h}, p_{1h}, p_{2h})$ to its dual, and the absolute value of the difference of the corresponding objective values is at most ε.

Then one can choose the next pair $(\beta_{h+1}, \gamma_{h+1})$ and do the same, etc. If the sets X and P are not empty and both β and γ are large enough, then case (c) is the only possibility. We have the following theorem.

THEOREM 2. *The sequences* $\{\xi_h\}$ *and* $\{\pi_h\}$ *converge. Problem (1)/(2) has a feasible solution if and only if* $\lim\limits_{h\to\infty} \xi_h (= \lim\limits_{h\to\infty} \pi_h) = 0$, *and in this case also* $\lim\limits_{h\to\infty} |\gamma_h \xi_h - \varepsilon| (= \lim\limits_{h\to\infty} \beta_h \pi_h = \varepsilon) = 0$ *is valid.*

This means that by applying the procedure in the above manner in the case when problem (1) has an optimal solution, both ξ_h and π_h will be arbitrarily small after a finite number of steps. Then the variables ξ and π can be omitted. Now there are two ways to proceed.

The first possibility is to terminate the procedure. According to the last theorem, (x_{0h}, x_{1h}, x_{2h}) and (p_{0h}, p_{1h}, p_{2h}) are almost feasible and almost optimal solutions to problems (1) and (2) respectively.

The other possibility is to continue the procedure in the same way as in the bounded case. (Actually, we have somewhat changed the original problem (1).) In the new problem we have $b_1 + \xi_h e$ on the right-hand side, and $c_1 - \pi_h e$ in the objective function instead of b_1 and c_1 respectively. The following theorem of Dantzig justifies why we can proceed now as in the bounded case.

THEOREM 3. *If both problems (3) and (4) have feasible solutions, and if both these have non-degenerate basic feasible solutions, then applying the procedure as in the bounded case, the obtained* $\tilde{p}_1$ *belongs to a bounded set, and the same is valid for the obtained* $\tilde{x}_1$.

We can thus apply the same argument as in Theorem 1 to establish the convergence of this part of the procedure for solving (1) without the boundedness assumptions.

If in the course of the above procedure one of the variables ξ and π is already small, the variable and the corresponding sequence can be omitted, and one can proceed similarly. Namely, if ξ_h is small, we omit the variable ξ and the sequence $\{\gamma_h\}$, and we continue with π and $\{\beta_h\}$; then $\lim\limits_{h\to\infty} \pi_h$ exists. If $\lim\limits_{h\to\infty} \pi_h = 0$, then problem (1) is an optimal solution; if $\lim\limits_{h\to\infty} \pi_h > 0$, then problem (1) is unbounded. (Now the sequence $\{\pi_h\}$ is obviously monotonic.) All these can be justified by applying the last theorem and ideas similarly to those of Theorem 2.

With respect to computations, we have no results on choosing and changing efficiently the ε and the sequences $\{\beta_h\}$ and $\{\gamma_h\}$. That is to say we do not know anything about the rate of the convergence of the procedure. The introduction of the assumption on boundness of X_1 and P_1 has assured the feasibility of the masters and made the presentation much more simple, i.e. we can avoid the usual phase-I procedures, which are now a bit complicated because of the two separate masters. There is similarly no problem with the presentation if at least one of the sets X_1 and P_1 is bounded. The boundness of X_1 is not unrealistic if one thinks of real, large LPs.

We have two further remarks.

One can think of *nested* decomposition based on this procedure.

Secondly, in the polar case, the LP problem

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \\ \max\ & cx \end{aligned}$$

where

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix}$$

$$b = (b_1, b_2, \dots, b_m)$$

$$c = (c_1, c_2, \dots, c_n)$$

and

$$x = (x_1, x_2, \dots, x_n)$$

has the equivalent LP-problem

$$\begin{aligned} \sum_j y_{ij} & \leq b_i && (i=1,2,\dots,m) \\ x_j - x_{ij} & = 0 && (i=1,2,\dots,m;\ j=1,2,\dots,n) \\ -y_{ij} + A_{ij}x_{ij} & \leq 0 && (i=1,2,\dots,m;\ j=1,2,\dots,n) \\ \max\ (\sum_j c_j x_j) & . \end{aligned}$$

Now applying the procedure for solving this problem we shall have subproblems of the form

$$A_{ij}\ x_{ij} \leq \tilde{y}_{ij}$$

$$x_{ij} \geq 0$$

$$\max(\tilde{q}_{ij}\ x_{ij})$$

where $\tilde{y}_{ij}$ and $\tilde{q}_{ij}$ are given vectors at each occurence of these subproblems. In this case the matrices of the subproblems are arbitrary parts of the matrix of the original problem.

To augment the masters it is not neccesary to use the variables corresponding to the optimal solutions of the subproblem. In fact, other solutions of (5) and its dual can be used. The procedure to be considered in the next part is such a variant.

2. A VARIANT OF THE PROCEDURE

To apply the simplex method there are several strategies to choose the incoming variables. To solve the masters using the procedure in Part 1, we simply choose a variable for which the relative cost is of appropriate sign. We denote the optimal solutions to the subproblem and its dual respectively. Since by $\tilde{x}_2$ and $\tilde{p}_2$ respectively. Since

$$c_1\tilde{x}_1 + c_2\tilde{x}_2 - \tilde{p}_1(A_{11}\tilde{x}_1 + A_{12}\tilde{x}_2) - \tilde{\pi}_0$$

$$= c_1\tilde{x}_1 - \tilde{p}_1 A_{11}\tilde{x}_1 + (c_2 - \tilde{p}_1 A_{12})\tilde{x}_2 + \tilde{p}_1 b_1 - \tilde{\phi}$$

$$= \tilde{\Psi} - \tilde{\xi}_0 - \tilde{p}_1 A_{11}\tilde{x}_1 + \tilde{p}_2(b_2 - A_{21}\tilde{x}_2) + \tilde{p}_1 b_1 - \tilde{\phi}$$

$$= \tilde{p}_1 b_1 + \tilde{p}_2 b_2 - (\tilde{p}_1 A_{11} + \tilde{p}_2 A_{21})\tilde{x}_1 - \tilde{\xi}_0 + \tilde{\Psi} - \tilde{\phi}$$

and $\tilde{\Psi} - \tilde{\phi} > 0$, it follows that at least one of the relations

$$c_1\tilde{x}_1 + c_2\tilde{x}_2 - \tilde{p}_1(A_{11}\tilde{x}_1 + A_{12}\tilde{x}_2) - \tilde{\pi}_0 > 0$$

or

$$\tilde{p}_1 b_1 + \tilde{p}_2 b_2 - (\tilde{p}_1 A_{11} + \tilde{p}_2 A_{21})\tilde{x}_1 - \tilde{\xi}_0 < 0$$

is valid.

Using a more efficient variable selection method, one can hope for better progress in the masters. The method to be investigated is dynamic scaling, i.e. the relative cost divided by a norm of the column is taken into account. This ratio is independent of variable scaling. Depending on the norm there are several further possibilities. The main point here is to choose an easy-to-calculate norm (or an approximation to the Euclidean norm) which still results in efficient variable selection. We will consider such a form, that was suggested and successfully tested in (1).

Denoting a column by a and a basis by B one can choose

$$||a|| = \max\{1, |\Sigma x_j|\}$$

where the summation is over those α_j components of B for which the jth basic variable is structural. It is obvious that the above $||a||$ can be easily calculated. Namely,

$$\tilde{\delta}_j = \begin{cases} 1 & \text{if the j}^{th}\text{ basic variable is structural} \\ 0 & \text{otherwise} \end{cases} .$$

defines the components of a vector $\tilde{d}$. The vector $\tilde{d}B^{-1}$ can be calculated parallel to the prices, and $|\Sigma x_j| = \tilde{d}B^{-1}a$ can be calculated parallel to the relative costs.

When choosing an incoming variable in a p_1-problem, this means that, instead of the subproblem (5), one has to solve a problem of the form

$$A_{22}x_2 \leq b_2 - A_{21}\tilde{x}_1$$

$$x_2 \geq 0 \tag{7}$$

$$\max\left\{\frac{c_1\tilde{x}_1 + c_2x_2 - \tilde{p}_1\,(A_{11}\tilde{x}_1 + A_{12}x_2) - \tilde{\pi}_0}{\max\{1, |\tilde{d}(A_{11}\tilde{x}_1 + A_{12}x_2) + \tilde{\delta}|\}}\right\} .$$

Applying the usual transformations, (7) can be reduced to a linear program. Under the condition

$$c_1\tilde{x}_1 + c_2x_2 - \tilde{p}_1(A_{11}\tilde{x}_1+A_{12}x_2) -\tilde{\pi}_0 > 0$$

the objective in (7) is equivalent to

$$\min \left\{ \frac{\max\{1, |\tilde{d}(A_{11}\tilde{x}_1 + A_{12}x_2) + \tilde{\delta}|\}}{c_1\tilde{x}_1 + c_2x_2 - \tilde{p}_1(A_{11}\tilde{x}_1 + A_{12}x_2) - \tilde{\pi}_0} \right\} \quad . \tag{8}$$

We let

$$\zeta = \frac{1}{c_1\tilde{x}_1 + c_2x_2 - \tilde{p}_1(A_{11}\tilde{x}_1 + A_{12}x_2) - \tilde{\pi}_0}$$

$$\xi_{2i} = \frac{\xi_{2i}}{c_1\tilde{x}_1 + c_2x_2 - \tilde{p}_1(A_{11}\tilde{x}_1 + A_{12}x_2) - \tilde{\pi}_0} \quad (i=\ldots)$$

where ξ_{2i} is the component of x_2. Denoting by z_2 the vector consisting of the ζ_{2i} (8), (7) becomes equivalent to the LP-problem

$$\begin{aligned}
\zeta + \xi &\leq 0 \\
dA_{12}z_2 + (\tilde{d}A_{11}\tilde{x}_1+\tilde{\delta})\zeta + \xi &\leq 0 \\
-dA_{12}z_2 - (\tilde{d}A_{11}\tilde{x}_1+\tilde{\delta})\zeta + \xi &\leq 0 \\
(c_2 - \tilde{p}_1A_{12})z_2 + (c_1\tilde{x}_1 - \tilde{p}_1A_{11}\tilde{x}_1 - \tilde{\pi}_0)\zeta &= 1 \\
A_{22}z_2 - (b_2 - A_{21}\tilde{x}_1)\zeta \qquad &\leq 0 \\
z_2, \zeta \geq 0 \qquad & \\
\max \xi \quad . \qquad &
\end{aligned} \tag{9}$$

It is easy to see that (9) cannot be unbounded and that it always has a feasible solution if there exists a variable in the p_1-problem for which the relative cost is greater than zero. If in an optimal solution to (9) $\zeta>0$, then one obtains a λ variable, and for $\zeta=0$ one has μ variable. (To be precise, one has to describe a separate problem similar to (7) for the μ variables. But the solution to this problem can be obtained from (9) in the case where $\zeta=0$.)

A similar possibility for the Dantzig-Wolfe procedure was discussed in Somos (1980).

A further point here is that from the dual solution of (9) one can obtain a σ variable for the x_1-problem.

One can apply the same argument for the x_1-problem. The dual of the problem corresponding to (9) is of the form

$$
\begin{aligned}
-(c_2-p_1A_{12})x_2 - \lambda_1 - \lambda_2(\tilde{p}_1A_{11}\tilde{e}+\tilde{\varepsilon}) + \lambda_3(\tilde{p}_1A_{11}\tilde{e}+\tilde{\varepsilon}) \\
- \lambda_4(\tilde{p}_1\tilde{b}_1-\tilde{p}_1A_{11}\tilde{x}_1-\tilde{\xi}_0) \leq 0 \\
\lambda_1+\lambda_2+\lambda_3 = 1 \\
A_{22}x_2 - \lambda_2A_{21}\tilde{e} + \lambda_3A_{21}\tilde{e} - \lambda_4(b_2-A_{21}\tilde{x}_1) \leq 0 \\
x_2, \lambda_1, \lambda_2, \lambda_3 \geq 0 \\
\max \ \lambda_4 \ .
\end{aligned}
$$

Both problem (9) and (10) require just a little more updating compared with the procedure of the previous part. They are not "far" from each other, so they can be solved at the same time. (Having solved these problems, one has two variables to each of the masters.)

None of these statements are valid if we try to use the greatest-change-in-the-objective column selection strategy for the masters. Furthermore, we could not prove in this case a theorem corresponding to the following.

THEOREM 4. *If in the case of $\tilde{\Phi}<\tilde{\Psi}$ one generates new variables for the masters by solving (9) and (10)(instead of solving (5)), then either (1) has no optimal solution and the procedure terminates after a finite number of steps or the $\tilde{\Phi}$ and $\tilde{\Psi}$ values converge to the optimal value of (1).*

REFERENCES

Maros, I., 1979. Dynamic scaling in the LP package LIPROS. Információ-Elektrónika(in Hungarian).

Somos, E., 1980. Dynamic scaling in the Dantzig-Wolfe decomposition. Információ-Elektrónika (in Hungarian)

Stahl, J., 1976. Decomposition procedures for convex programs. Optimization and Operations Research. Lecture Notes in Economics and Mathematical Systems 117. Springer Verlag.

Stahl, J., 1978. Decomposition procedures for LPs. Közgazdasági és Jogi Konyvkiado.

THE ELLIPSOIDAL ALGORITHM

AN EQUIVALENCE BETWEEN THE SUBGRADIENT AND ELLIPSOIDAL ALGORITHMS

Marshall L. Fisher*

Department of Decision Sciences
The Wharton School
University of Pennsylvania

We show that the ellipsoidal algorithm is equivalent to the subgradient algorithm executed in a transformed problem space. This suggests that the ellipsoidal algorithm will perform well on problems for which the subgradient algorithm is known already to perform well. The best example of this is the dual problem arising in the Lagrangian relaxation approach to integer programming, on which the subgradient algorithm has generally out-performed the simplex method.

*This paper was supported in part by NSF Grant ENG 78-26500 to the University of Pennsylvania.

The subgradient and ellipsoidal algorithms are iterative methods for finding $x \varepsilon R^n$ satisfying

$$a_i^T x \leq b_i \quad (i=1,\ldots, m, \ a_i \ \varepsilon \ Z^n, \ b_i \ \varepsilon \ Z) \qquad \text{(P)}$$

In (P), x and a_i are column vectors and T denotes transpose.

Both methods begin at an arbitrary point $x^0 \ \varepsilon \ R^n$ and generate a sequence $\{x^k\}$ of trial solutions. The methods differ in the rule for obtaining x^{k+1} from x^k. If x^k solves (P), both methods stop. Otherwise, let i_k denote the index of a constraint violated by x^k, (i.e. $a_{i_k}^T \ x^k > b_{i_k}$) and set $a=a_{i_k}$.

The subgradient rule is

$$x^{k+1} = x^k - t_k a \qquad \text{(SG)}$$

where t_k is a positive scalar step size. Several approaches for setting t_k have been proposed and observed to perform well in practice. The fundamental theoretical result is the $\{x^k\}$ converges to a solution to a solution of (P) if one exists and if

$$\lim_{k\to\infty} t_k = 0 \text{ and } \lim_{k\to\infty} \sum_{i=0}^{k} t_i = \infty.$$

This method was first proposed by Agmon [1] and Motzkin and Schoenburg [6]. Subsequent study has included a substantial Russian literature, principally Poljak [7] and Shor [8] and rate of convergence studies such as Goffin [3]. Held, Wolfe and Crowder [4] reported successful computational experience and Fisher [2] has surveyed the use of the subgradient method in conjunction with Lagrangian relaxation.

The ellipsoidal rule is

$$x^{k+1} = x^k - \frac{1}{(n + 1)||Q_k^T a||} Q_k Q_k^T a \qquad (E)$$

where Q_k is an n x n nonsingular matrix. This method appears to have first been suggested by Shor [9], [10]. Recently, Khachian [5] achieved a fundamental breakthrough in complexity theory by showing how to specify $\{Q_k\}$ so as to solve (P) in polynomial time.

The subgradient and ellipsoidal rules are obviously quite similar. If we take $t_k = \dfrac{1}{(n + 1)||Q_k^t a||}$ in the subgradient rule, then they differ only in their direction vectors. The direction -a of the subgradient method is normal to the hyperplane $ax = b_{i_k}$ and points into the set of points satisfying $ax \leq b_{i_k}$. In the ellipsoidal algorithm, the subgradient direction is deflected to $-Q_k Q_k^T a$, which forms an acute angle with -a.

Comparison of the two methods can be sharpened by employing a statement of (P) in a transformed space. Let Q be a given n x n nonsingular matrix. We transform the original problem variables by $y = Q^{-1}x$ to obtain

$$a_i^T Qy \leq b_i \qquad (i = 1,\ldots, m,\ a_i \in Z^n,\ b_i \in Z) \qquad (P_Q)$$

Problem (P_Q) is equivalent to (P) in the sense that y solves (P_Q) if and only if x = Qy solves (P). However, (P_Q) and (P) are not equivalent from the algorithmic viewpoint in that the performance of the subgradient method is generally not the same on (P) and (P_Q). Thus one might be able to speed up convergence of the subgradient method by an appropriate choice of Q.

In a sense, this is exactly what the ellipsoidal algorithm does. Precisely, iteration k of the ellipsoidal algorithm applied to (P) is identical to iteration k of the subgradient method on (P_{Q_k}) with $t_k = \dfrac{1}{(n+1)||Q_k^T a||}$.

To verify this, apply (SG) to (P_{Q_k}).

$$y^{k+1} = y^k - \frac{1}{(n+1)||Q_k^T a||} (a^T Q_k)^T$$

$$= Q_k^{-1} x^k - \frac{1}{(n+1)||Q_k^T a||} Q_k^T a.$$

Transforming y^{k+1} to $x^{k+1} = Q_k y^{k+1}$ gives

$$x^{k+1} = Q_k y^{k+1}$$

$$= x^k - \frac{1}{(n+1)||Q_k^T a||} Q_k Q_k^T a$$

which is identical with the point determined by (E).

What are the implications of this result? A number of researchers have conducted computational experiments to determine what, if any, is the practical value of the ellipsoidal algorithm for linear programming. To date this effort has failed to discover a class of problems on which the ellipsoidal algorithm outperforms the simplex method. The result of this paper suggests that the ellipsoidal algorithm is likely to perform well on problems for which the subgradient method is known to outperform the simplex method. The most spectacular example of this is the success of the subgradient method in solving the dual problems which arise in the Langrangian relaxation approach to integer programming. As discussed in Fisher [2], this approach involves dualizing a set of complicating constraints of an integer program to obtain a Lagrangian problem whose optimal value is a lower bound (for minimization problems) on the optimal value of the original problem. The Lagrangian problem can thus be used in place of a linear programming relaxation to provide bounds in a branch and bound algorithm. This approach has lead to dramatically improved algorithms for a number of important problems in the areas of routing, location, scheduling, assignment and set covering.

The dual problem in Lagrangian relaxation is the problem of finding values for the dual variables that maximize the lower bound. This problem is a linear program with a vast number of constraints that are implicitly generated as solutions of the Lagrangian problem. Fisher [2] surveys the computational experience with the solution of Lagrangian duals and concludes that the subgradient method has generally outperformed the simplex method.

This suggest that the ellipsoidal algorithm would also do well on these problems. Moreover, the ellipsoidal algorithm can accomodate the large number of implicitly generated constraints much more easily than the simplex method. Finally, when one uses a dual problem within branch and bound, it is not neccessary to obtain an optimal solution to the dual. The question of interest is whether there are dual variable values that give a lower bound large enough to fathom the current node of the branch and bound tree. Since this problem has exactly the same form as (P), any awkwardness in converting an optimization problem to (P) is avoided.

Acknowledgement

This paper was supported in part by NSF Grant ENG 78-26500 to the University of Pennsylvania.

REFERENCES

1. Agmon, S., "The relaxation method for linear inequalities." Canadian Journal of Mathematics 6 (1954) 382-392.

2. Fisher, M.L., "The Lagrangian Relaxation Method for Solving Integer Programming Problems," forthcoming in Management Science.

3. Goffin, J.L., "On the Convergence Rates of Subgradient Optimization Methods." Math. Prog., 13 (1977), 329-347.

4. Held, M., Wolfe, P., & Crowder, H.D., "Validation of Subgradient Optimization." Mathematical Programming, 6 (1974), 62-88.

5. Khachian, L.G., "A polynomial algorithm in linear programming." Doklady Akademiia Nauk USSR 244 (No. 5, February, 1979) 1093-96 (translated as Soviet Mathematics Doklady 20, 191-194).

6. Motzkin, T. and Schoenberg, I.J., "The relaxation method for linear inequalities." Canadian Journal of Mathematics 6 (1964) 393-404.

7. Poljak, B.T., "A General Method for Solving Extremum Problems". Soviet Mathematics Doklady, 8 (1967), 593-597.

8. Shor, N.Z., "On the structure of algorithms for the numerical solution of optimal planning and design problems," Dissertation, Cybernetics Institute, Academy of Sciences U.S.S.R. (1964).

9. Shor, N.Z., "Utilization of the operation of space dilation in the minimization of convex functions." Kibernetika 6 (No. 1, Jan.-Feb., 1970), 6-12. Translated as Cybernetics 6, 102-108.

10. Shor, N.Z., "Convergence rate of the gradient descent method with dilation of the space," Kibernetika 6 (No. 2, March-April, 1970), 80-85. Translated as Cybernetics 6, 102-108.

A NUMERICAL INVESTIGATION OF ELLIPSOID ALGORITHMS FOR LARGE-SCALE LINEAR PROGRAMMING*

Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright

Systems Optimization Laboratory
Department of Operations Research
Stanford University

The ellipsoid algorithm associated with Khachiyan and others has certain theoretical properties that suggest its use as a linear programming algorithm. Some of the practical difficulties are investigated here. A variant of the ellipsoid update is first developed, to take advantage of the range constraints that often occur in linear programs (i.e. constraints of the form $l \leqslant a^T x \leqslant u$, where $u - l$ is reasonably small). Methods for storing the ellipsoid matrix are then discussed for both dense and sparse problems. In the large-scale case, a major difficulty is that the desired ellipsoid cannot be represented compactly throughout an arbitrary number of iterations. Some schemes are suggested for economizing on storage, but any guarantee of convergence is effectively lost. At this stage there remains little room for optimism that an ellipsoid-based algorithm could compete with the simplex method on problems with a large number of variables.

*This research was supported by the U.S. Department of Energy Contract DE-AC03-76SF00326, PA No. DE-AT03-76ER72018; National Science Foundation Grants MCS-7926009 and ENG77-06761; the Office of Naval Research Contract N00014-75-C-0267; and the U.S. Army Research Office Contract DAAG29-79-C-0110. Reproduction in whole or in part is permitted for any purposes of the United States Government.

1. INTRODUCTION

Considerable interest has been generated by the publication (Khachiyan, 1979) of a so-called ellipsoid algorithm that can theoretically find a feasible point for certain sets of linear inequalities in polynomial time. The algorithm defines an initial ellipsoid that encloses a finite volume of the feasible region, and proceeds by defining a sequence of shrinking ellipsoids, each of which contains this feasible region. The center of one of the ellipsoids must eventually be a feasible point, for if not, the volume of the ellipsoids will ultimately be smaller than that of the feasible region they contain (a contradiction).

It is known that a feasible-point algorithm can be adapted to solve the linear programming problem, and conversely. It is also well known that the simplex method (Dantzig, 1963), the standard technique for solving linear programs, is potentially an exponential-time algorithm, and that simple examples exist that elicit the algorithm's worst-case performance. It is therefore natural for the question to have been raised: for the solution of linear programs, could Khachiyan's algorithm prove to be superior to the simplex method?

There are several practical difficulties with the original ellipsoid algorithm, and with various derivatives that have since been proposed. For large-scale problems, perhaps the most obvious difficulty is that the matrix defining the required ellipsoid is far too large to be stored. The corresponding basis matrix in the simplex method is smaller in dimension and is invariably very sparse. Also, the known practical performance of the simplex method (except on contrived examples) is exceedingly good; in fact it is safe to regard it as a *linear*-time algorithm. Therefore, for problems involving more than 100 variables (say), the hope that an ellipsoid algorithm might prove superior should never have been high.

A redeeming feature of ellipsoid algorithms is that they do not require the solution of any large systems of equations each iteration, and therefore do not require a matrix factorization for that purpose. Also, there is a degree of arbitrariness about the definition of the ellipsoid, and the general flavor is that of an iterative procedure. These properties we hope to exploit. Although the bound on the number of iterations for Khachiyan's algorithm is low compared to that of the simplex method, it is still far too large to be meaningful. In abandoning the polynomial-time features of the algorithm, little of practical value will be lost.

No attempt is made here to review the literature concerning ellipsoid algorithms (most of which is theoretical in nature). For chronological details we refer the reader to Lawler (1980) and Wolfe (1980a,b).

2. SOLVING LINEAR PROGRAMS

The primal simplex method is usually implemented to solve linear programming problems in the following standard form:

$$\text{LP1:} \qquad \begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax + y = b \\ & l \le x \le u \\ & l' \le y \le u' \end{array}$$

where A is $m \times n$. There are no restrictions on m and n, but it is usually true that $m < n$. A typical ratio is $m \approx \frac{1}{2}n$.

The slack variables y are present to allow the simplex method to be implemented using column operations only. The upper and lower bounds on x and y allow for equality and inequality constraints of all types. Many of the bounds could be infinite, but a user would usually be able to assign reasonable values to them if pressed to do so.

It is important to note that in most real-life examples a large proportion of the general constraints are equalities, so that many components of y should be zero at a solution. (Thus, for perhaps half of the constraints, the associated bounds will be $l'_i = u'_i = 0$.) It has been suggested that for each such i, some variable x_j could be eliminated. However, when so many equality constraints are present, this would duplicate much of the computational effort involved in the simplex method. Furthermore, the number of general constraints would not be reduced unless the bounds on x_j were both infinite. (For example, the simple lower bound $x_j \ge l_j$ would be transformed into a general constraint that would have to be retained, unless $l_j = -\infty$.)

Conversion to a feasible-point problem.

In order to convert a linear program to a feasible-point problem, various suggestions have been made concerning the dual of the linear program (e.g., Aspvall and Stone, 1979). Because of the inclusion upper *and* lower bounds above, the dual of LP1 is complex, and for the sake of brevity we will not give it here. It is sufficient to note that the combined primal-dual system involves a much larger number of constraints and (more significantly) a much larger number of variables than in the primal problem alone. Also, the volume of the feasible region is essentially zero. At this stage of development, there seems little point in considering the primal-dual system.

Returning to LP1, we shall instead take the obvious approach of imposing a "target value" on the objective function. The aim will be to find a point x that achieves this value and also satisfies the primal constraints. With an ellipsoid algorithm there is no need to introduce slack variables. We shall therefore consider the problem

$$\boxed{\begin{array}{lll} \text{LP2:} & \text{find a point} & x \\ & \text{that satisfies} & c^T x \leq t \\ & & b_l \leq Ax \leq b_u \\ & & l \leq \ x \ \leq u \end{array}}$$

This is exactly equivalent to LP1 if the bounds on Ax are suitably defined and if the target value t happens to be the optimal value for the objective.

Naturally we still wish to avoid treating a problem whose feasible region has zero volume. For test purposes we choose to perturb all of the bounds to obtain the following relaxed problem:

$$\boxed{\begin{array}{lll} \text{LP3:} & \text{find a point} & x \\ & \text{that satisfies} & c^T x \leq t + \delta t \\ & & b_l - \delta b_l \leq Ax \leq b_u + \delta b_u \\ & & l - \delta l \leq \ x \ \leq u + \delta u \end{array}}$$

where the perturbations are positive vectors and will be substantial (rather than near rounding level). Some of the upper and lower bounds could still be $+\infty$ or $-\infty$ respectively, but any general equality constraints will now be range constraints.

In fact, it would be sufficient to relax just the bounds on equality constraints by a nontrivial amount, as long as the objective perturbation δt is also significant.

Clearly, if LP1 is well posed and has a known optimal objective t, then the feasible region for LP3 has nonzero volume. The larger the perturbations the larger the feasible region. Our reason for dealing with such problems is that they can be derived naturally from existing LP models with known solutions. Furthermore, if an ellipsoid algorithm is *not* able to solve such problems satisfactorily, then it is unlikely to be of any use on a problem whose solution is unknown.

3. AN ELLIPSOID FOR RANGE CONSTRAINTS

Since range constraints arise naturally, it is worthwhile developing an ellipsoid that takes advantage of them. It will then be possible to accomplish in one iteration an effect that would take many consecutive iterations with the earlier ellipsoid algorithms.

In order to define the necessary notation, we shall first derive the usual updating process in a fairly general way.

The linear transformation.
An ellipsoid may be represented in the form

$$(x - x_k)^T(R^TR)^{-1}(x - x_k) \leq \sigma^2, \tag{1}$$

where x_k is its center, σ is a scalar, and

$$B \equiv R^TR$$

is a positive-definite matrix. Given any nonzero vector a, we can transform this ellipsoid into a hypersphere as follows. Let the vector Ra be normalized to have unit Euclidean length, and then choose an orthonormal matrix Q ($Q^TQ = I$) that reduces it to the first column of the identity matrix:

$$\begin{aligned} \nu &= \|Ra\|_2, \\ q &= \frac{1}{\nu}Ra, \\ Qq &= e_1. \end{aligned} \tag{2}$$

(The matrix Q will not be needed in practice.) Now define some new variables z according to the linear transformation

$$x - x_k = \sigma R^TQ^Tz.$$

It is easy to see that the original ellipsoid reduces to $z^Tz \leq 1$, a sphere in n dimensions with center at the point $z_k = 0$ (i.e., the origin).

If $a^Tx \geq l_i$ happens to be the i-th constraint on x, we can define a scalar ρ to be the "scaled residual",

$$\rho = (l_i - a^Tx_k)/\sigma\nu,$$

and the corresponding constraint on z will become $z \geq \rho e_1$. If x_k violates the constraint, and if the constraint cuts the original ellipsoid, ρ will lie within the range $0 < \rho < 1$. The transformed ellipsoid and constraint are shown as the circle and the left-most vertical line in Figures 1, 2 and 3.

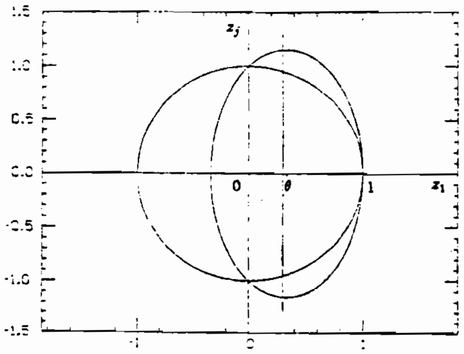


Figure 1. Khachiyan's original ellipsoid.

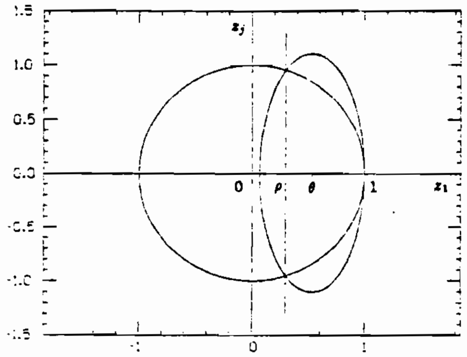


Figure 2. The deep-cut ellipsoid.

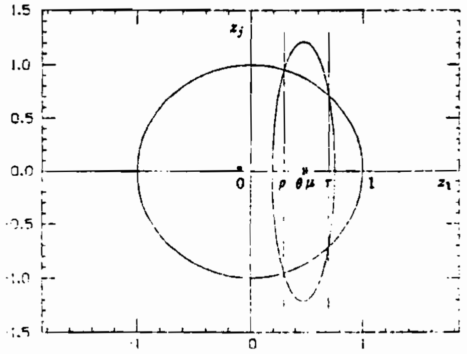


Figure 3. The range ellipsoid.

An updated ellipsoid.
A new ellipsoid with center at the point $z_{k+1} = \theta e_1$, and with all major axes equal except for the first, must take the form

$$\alpha(z_1 - \theta)^2 + \sum_{j=2}^{n} \beta z_j^2 \leq 1.$$

(The three quantities θ, α and β have yet to be specified.) In matrix notation, this is

$$(z - z_{k+1})^T \begin{pmatrix} \alpha & \\ & \beta I_{n-1} \end{pmatrix} (z - z_{k+1}) \leq 1,$$

which becomes

$$(x - x_{k+1})^T R^{-1} Q^T \begin{pmatrix} \alpha/\beta & \\ & I_{n-1} \end{pmatrix} Q R^{-T} (x - x_{k+1}) \leq \frac{\sigma^2}{\beta}$$

in the original coordinates. We wish to write this in a form analogous to equation (1), namely

$$(x - x_{k+1})^T (\bar{R}^T \bar{R})^{-1} (x - x_{k+1}) \leq \bar{\sigma}^2, \tag{3}$$

where

$$\bar{B} \equiv \bar{R}^T \bar{R} \quad \text{and} \quad \bar{\sigma}^2 \equiv \sigma^2/\beta.$$

It follows that

$$\begin{aligned} \bar{B} &= R^T Q^T (I - (1 - \frac{\beta}{\alpha}) e_1 e_1^T) Q R \\ &= R^T (I - \delta q q^T) R \\ &= R^T (I - \xi q q^T)^2 R, \end{aligned}$$

and so

$$\bar{B} = B - \delta p p^T, \tag{4}$$

$$\bar{R} = (I - \xi q q^T) R, \tag{5}$$

$$\bar{\sigma} = \sigma \gamma, \tag{6}$$

$$x_{k+1} = x_k + \sigma \theta p, \tag{7}$$

where

$$p = R^T q, \tag{8}$$

$$\gamma^2 = 1/\beta, \tag{9}$$

$$\phi^2 = \beta/\alpha, \tag{10}$$

$$\delta = 1 - \phi^2, \tag{11}$$

$$\xi = \delta/(1 + \phi). \tag{12}$$

The range ellipsoid.
We must now choose θ, α and β in some optimal way. To do this, we first specify where the surface of the new ellipsoid should be, by imposing two simple constraints. This determines α and β in terms of θ. The volume of the new ellipsoid can then be minimized with respect to θ.

Figure 1 illustrates where Khachiyan's updated ellipsoid was chosen to lie, and Figure 2 shows the so-called deep-cut ellipsoid that was later proposed by many authors. Both cases can be obtained from the "range ellipsoid", which we shall now derive.

In general, the constraint defining the above transformations will take the form

$$l_i \leq a^T x \leq u_i.$$

The lower bound gave rise to the scaled residual ρ, and the upper bound defines a similar quantity τ as follows:

$$\begin{aligned} \rho &= (l_i - a^T x_k)/\sigma\nu, \\ \tau &= (u_i - a^T x_k)/\sigma\nu, \\ \mu &= \tfrac{1}{2}(\rho + \tau), \\ \psi &= (1 - \rho^2) + (1 - \tau^2), \end{aligned} \tag{13}$$

where μ is the mid-point of the scaled range, and ψ will be useful below. The transformed range constraint is

$$\rho e_1 \leq z \leq \tau e_1, \qquad \text{or} \qquad \rho \leq z_1 \leq \tau.$$

This is shown as the two vertical lines in Figure 3. (We are considering the case where u_i is small enough for the second line to pass through the hypersphere, i.e., the inequality $\tau \leq 1$ is satisfied. Otherwise, we simply set $\tau = 1$.)

Clearly the ellipsoid in Figure 3 will contain all of the relevant feasible region if it cuts the hypersphere at the points $z_1 = \rho$ and $z_1 = \tau$. Assuming $\rho < \tau$, this gives the two conditions

$$\alpha(\rho - \theta)^2 + \beta(1 - \rho^2) = 1, \tag{14}$$

$$\alpha(\tau - \theta)^2 + \beta(1 - \tau^2) = 1, \tag{15}$$

from which we can deduce expressions for α and β in terms of θ:

$$\begin{aligned} \frac{\beta}{\alpha} &= 1 - \frac{\theta}{\mu}, \\ \frac{1}{\alpha} &= \theta^2 - \frac{1 + \rho\tau}{\mu}\theta + 1, \\ \frac{1}{\beta} &= 1 - \mu\theta + \frac{\theta}{\mu - \theta}\frac{(\tau - \rho)^2}{4}. \end{aligned} \tag{16}$$

Now the volume of the new ellipsoid relative to the hypersphere is $1/\sqrt{\alpha\beta^{n-1}}$. Hence, the volume can be minimized by choosing θ to satisfy

$$\frac{\partial}{\partial\theta}\left(\frac{1}{\alpha\beta^{n-1}}\right) = 0.$$

This leads to the quadratic equation

$$(n+1)\theta^2 - \left(2n\mu + \frac{1+\rho\tau}{\mu}\right)\theta + (n\rho\tau + 1) = 0,$$

and since it is clear from the figure that θ must lie to the left of the mid-point μ, the required root of the quadratic is given by

$$\begin{aligned} \theta &= \mu - \Delta, \\ \Delta &= \frac{1}{4\mu(n+1)}\left(\sqrt{(n^2-1)(\tau^2-\rho^2)^2+\psi^2} - \psi\right). \end{aligned} \tag{17}$$

This completes the derivation. The optimal range ellipsoid is defined by equations (2)–(13) and (16)–(17).

Discussion.
Setting $\tau = 1$ gives the deep-cut ellipsoid in Figure 2, and $\rho = 0$, $\tau = 1$ gives the original (Khachiyan) ellipsoid in Figure 1.

If $\rho = \tau$, i.e., if the original range constraint was actually an equality constraint, equations (14) and (15) will be dependent. In this case the updated ellipsoid reduces to a subspace, and in equations (4)–(12) we would take $\theta = \rho$, $1/\alpha = 0$, $1/\beta = 1 - \rho^2$, $\phi = 0$, and $\delta = \xi = 1$.

4. STORING THE ELLIPSOID

For derivation purposes we represented the ellipsoid matrix above by $B = R^TR$. The best way to store B in practice is naturally open to question. Although we are primarily concerned with large problems, it is worthwhile considering small problems first.

The dense case.
One pleasing feature of an ellipsoid algorithm is that it can be "super stable", provided a little care is taken with its implementation. By super stable we mean that rounding errors may slightly *retard* convergence but will not *prevent* it. This statement may come as a surprise, since the original algorithm has been

criticized on numerical grounds. It is important to note that the mere fact that the matrix B can become ill-conditioned does not necessarily imply that the algorithm is unstable. Indeed it is quite possible for B to be singular without incurring any ill effects, provided it is represented in an appropriate manner.

The main operation in question is the recurrence of a positive-definite matrix given a rank-one modification:

$$\bar{B} = B - \delta pp^T.$$

This problem occurs in many algorithms and a fully satisfactory solution is known. In other situations we usually wish to solve some linear equations with the updated matrix, and hence it has proved beneficial to recur the Cholesky factorization $B = LDL^T$, where L is lower triangular and D is diagonal. In the present context we only need to form products of the form Ba, so "invertibility" of B or its factors is not relevant. Nevertheless, it can be helpful to recur the Cholesky factors, because there are procedures for doing so that guarantee B will retain numerical positive definiteness (e.g., Gill, Murray and Saunders, 1975). Any rounding errors incurred simply cause the computed ellipsoid to be slightly larger than it would be analytically.

If equality constraints are admitted, B will become singular and we can again update the Cholesky factors in a way that guarantees positive semidefiniteness. (Some of the diagonals of D will be exactly zero.) This would be preferable to updating B itself, but it is not safe to assume that subsequent iterates will satisfy the equality constraints to within working accuracy. To avoid unreasonable loss of feasibility with respect to the equality constraints it would be necessary to continue checking them. If one of them is not satisfied to within the required tolerance, it would be necessary to repeat the rank-reduction process. The net effect is therefore almost identical to treating equalities as range constraints.

An alternative to $B = LDL^T$ is the factorization $B = R^TR$ with R held as a dense, square matrix. On numerical grounds there is little to choose between the two approaches, even if singularity arises owing to the presence of equality constraints. (For example, Wolfe (1980b) has implemented the deep-cut ellipsoid using $\sigma^2 B = J^TJ$ with square J, and mentions performing 30,000 updates without numerical difficulty.) However, the Cholesky factorization requires less work per iteration and only half the storage, i.e., essentially the same requirements as if B itself were maintained as a symmetric matrix.

The sparse case.

For large n it is clearly not practical to represent B using dense matrices. The only obvious approach is to start with $B = I$ or some diagonal matrix (requiring minimal storage) and then update B or its factors in some kind of product form.

It is known how to update Cholesky factors in the form

$$L = L_k L_{k-1} \dots L_0,$$

where each factor L_j is triangular and can be stored compactly using two n-vectors. However, there are more efficient alternatives. From equation (4) above we see that the factorization $B = R^T R$ would give the product form

$$R = (I - \xi_k q_k q_k^T) \dots (I - \xi_1 q_1 q_1^T) R_0, \tag{18}$$

requiring storage of a scalar ξ_j and one n-vector q_j per iteration. Even more simply, we have

$$B = B_0 - \delta_1 p_1 p_1^T - \dots - \delta_k p_k p_k^T, \tag{19}$$

a direct summation that requires the same storage as the product form of R, but about half the work per iteration.

One possible advantage of using (18) rather than (19) is that the vectors $\{q_j\}$ are almost certainly more sparse than the vectors $\{p_j\}$, at least initially. However, since both sets of vectors rapidly become dense (particularly if the violated constraint vector a is chosen as an aggregation of several violated constraints), this advantage is of little significance. The fact that only one pass is needed through the vectors $\{p_j\}$ each iteration, weighs heavily in favor of (19).

In the dense case we argued against working with B itself. This was because the Cholesky factorization offered superior numerical reliability at no cost in terms of storage or work. In fact, it would be safe to work with B as long as there are no equality constraints or very narrow ranges, and as long as the scaled residual ρ is prevented from being very close to one.

A fundamental difficulty with both the product form (18) and the summation form (19) is that the storage required grows steadily with each iteration. This is analogous to sparse implementations of the simplex method, in which the factorization of the basis matrix tends to occupy more and more storage each time it is updated. A vital difference is that the basis matrix can be refactorized periodically in a compact form that seldom requires storage for more than $5m$ nonzeros. In other words, the simplex method's workspace can be condensed when necessary *without any loss of ground*; the next iteration will not be materially different from what it would have been had the condensation not taken place. Unfortunately this does not appear to be true for the ellipsoid algorithm.

The need to compactify storage is discussed further below under the heading of "resetting" and "cycling" strategies.

5. IMPLEMENTATION ASPECTS

An ellipsoid algorithm for solving the feasible-point problem (such as problem LP3) would ideally take the following form:

1. (Initialize.) Set $k = 0$ and choose an initial ellipsoid defined by x_0, B and σ, such that B is positive definite and at least part of the feasible region satisfies $(x - x_0)^T B^{-1}(x - x_0) \leq \sigma^2$.
2. (Terminate?) If x_k satisfies the constraints to within a required tolerance, accept it as a feasible solution and terminate.
3. (Choose a constraint.) Select, or construct, a constraint of the form $l_i \leq a^T x \leq u_i$ that is not satisfied to the desired accuracy.
4. (Update.) Obtain new quantities x_{k+1}, $\bar{B}$ and $\bar{\sigma}$ by a process such as the one described in section 3. Set $k = k + 1$ and return to step 2, using the new quantities in place of the old.

We need to consider how each step of such an algorithm might be implemented on a machine with finite precision and finite storage. It will be necessary to introduce certain changes to the algorithm, and some of these will unfortunately invalidate the proof of convergence.

Choice of initial ellipsoid.

Since the method proceeds by shrinking the volume of an ellipsoid enclosing a solution, the efficiency of the method will be doubly enhanced if the initial ellipsoid has a small volume. (Over-estimating the initial size of the ellipsoid permits x_k to move far away from x_0. It also slows the initial rate of reduction in volume.)

In some cases the user may be able provide an estimate of the distance from x_0 to the feasible region. Alternatively, it would not be unreasonable to ask the user to place sensible lower *and* upper bounds on all variables. An initial ellipsoid could then be defined from a diagonal matrix with assurance that it contained some feasible points. However, it would probably be a gross over-estimate of the dimensions that would in reality suffice.

In practice, some general procedure for choosing an initial ellipsoid is required, whether sensible bounds on the variables are available or not. Barring use of the simplex method, it seems that any procedure that is *guaranteed* to enclose part of the feasible region is likely to give an initial ellipsoid that is much too large for the subsequent algorithm to be efficient. An estimate without such guarantees may be adequate, provided we have some means of increasing the size of the ellipsoid should the estimate prove to have been too small.

The method we have used is to set the initial $\sigma^2 B$ equal to the hypersphere $\sigma^2 I$, where the radius σ is chosen so that the initial scaled residual ρ takes a specified value, such as 0.5 or 0.1. The smaller the value of ρ the more likely the ellipsoid will be sufficiently large. However, the initial rate of convergence will be correspondingly slower.

Expanding and shrinking the ellipsoid.

It is worth noting that it is not essential for there to be a feasible point within the initial ellipsoid. Subsequent ellipsoids will always contain regions that were not contained in their predecessors. Also, at every iteration we will consider altering the current radius σ to ensure that the scaled residual satisfies

$$0 < \rho_{min} \leq \rho \leq \rho_{max} < 1.$$

(Typical values are $\rho_{min} = 0.01$, $\rho_{max} = 0.9$.) These are heuristic values to prevent the current ellipsoid from being "too big" or "too small", respectively. In particular, if $\rho > \rho_{max}$, we assume that the ellipsoid is too small and increase σ accordingly.

Unfortunately, a value of $\rho > 1$ will ultimately arise in the case where no feasible point exists. The strategy of expanding the ellipsoid therefore eliminates any hope of confirming the non-existence of a feasible point. (On the other hand, such confirmation was never a practical reality with the original algorithm either.) More seriously, if ρ_{max} becomes active and forces an increase in σ, the proof of convergence is invalidated because the ellipsoids are no longer continually shrinking.

The interpretation of a small value of ρ is that all the violated constraints pass close to the center of the ellipsoid. Under such circumstances it seems reasonable to shrink the size of the ellipsoid by reducing σ. This should not interfere too seriously with the proof of convergence. In any event, not to do so would result in a long succession of small ρ values, and the corresponding volume reduction would be negligible.

Choice of constraint.

The proof of convergence does not depend on which violated constraint is chosen to construct the next ellipsoid, but it seems reasonable to suppose that the *rate* of convergence may depend critically on the choice made. The reduction in volume of succeeding ellipsoids is greater the larger the value of ρ. Therefore it may appear that ρ should be maximized. This could be done if the quantities $a_i^T B a_i$ were recurred for each constraint vector a_i. However, additional work would be required, and in reality it would be a poor strategy. The reason is that the ellipsoid may be very oblate. (This would be certain if the problem has narrow range constraints.) The scaled residual $\rho_i = (i\text{-th violation})/(\sigma(a_i^T B a_i)^{\frac{1}{2}})$ may be very large just because $a_i^T B a_i$ is very small. It is not the volume of the ellipsoid that is of overriding concern, since the volume can be arbitrarily small if the ellipsoid tends towards a subspace, and this can happen at any stage.

Instead of the above, we adopted the obvious strategy of choosing constraints with the largest violations. For consistency the general constraints were always scaled so that $\sum_{j=1}^{n} |a_{ij}| = 1$.

Intuitively, choosing just a single violated constraint would seem to be a myopic strategy, and indeed it was found to give a poor rate of convergence in the early iterations. It is worth noting that when B is represented in summation form, the product

$$Ba = (B_0 - \delta_1 p_1 p_1^T - \cdots - \delta_k p_k p_k^T)a$$

is computed by setting $w \leftarrow B_0 a$ and then performing the operations

$$\pi \leftarrow a^T p_j, \qquad w \leftarrow w - \pi \delta_j p_j$$

for j from 1 to k. The p_j's are stored as dense vectors, but if a is just one row a_i from the original constraint matrix it will be very sparse, and so the scalars π can be computed at negligible cost. Hence in the sparse case, an iteration can be performed almost twice as fast if violated constraints are not aggregated.

In spite of the previous comment, overall performance is usually much improved if a violated constraint is constructed from the set of all violated constraints, according to

$$a = \sum \omega_i a_i$$

for appropriate indices i and weights ω_i. The weights we have experimented with are $\omega_i = r_i$, $\sqrt{r_i}$, and 1, where r_i is the violation for the i-th constraint.

A disadvantage of this aggregated constraint (apart from giving a dense a) is that unless all of the constraints involved have reasonable upper and lower bounds, the aggregated range is unlikely to be small. Most of the advantage of the range ellipsoid will therefore be lost. One way of avoiding this drawback would be to form three separate aggregated constraints, one from any narrow range constraints (e.g., perturbed equalities), one from normal range constraints, and one from the remainder. At each iteration, one of these aggregated constraints would be chosen. However, this variation was not tried.

Resetting strategies.

Even with B represented in product or summation form, the most serious implementation difficulty is still the amount of storage required. After k updates to an initial ellipsoid we need to store k dense vectors p_j, each of length n. For large n it is clearly not practical to allow k to exceed 100 (say), and the work per iteration for such a large k would far exceed that involved in a typical iteration of the simplex method.

Ideally we would like to define a new ellipsoid at some stage, with the same center x_k and the following three properties:

1. it should have a sparse representation;
2. it should be similar in volume;
3. it should enclose the original ellipsoid.

For example, the current $\sigma^2 B$ could conceivably be replaced by $\sigma^2 \lambda I$ where λ is the largesı eigenvalue of B. This would obviously satisfy properties 1 and 3. However, the new volume is likely to be considerably larger than before. We have been unable to define an ellipsoid that has all three properties, and it is probable that no such ellipsoid exists. Instead, since we have some means of increasing the size of the ellipsoid should it prove to be too small, it may be sufficient to satisfy properties 1 and 2.

The resetting strategy we have used is as follows. At some specified frequency (every K iterations where $K = 20$, say), the current ellipsoid

$$(x - x_k)^T B^{-1} (x - x_k) \leq \sigma^2$$

is replaced by

$$(x - x_k)^T D^{-1} (x - x_k) \leq (\varphi\sigma)^2,$$

where $D = \text{diag}(B)$ and $\varphi = 0.9$, provided the choice of 0.9 leads to a satisfactory value of ρ on the next iteration. It can be shown that if $\varphi = 1$, the new ellipsoid would enclose the old one along its smallest axis, but not along its largest axis. From Figure 3 we see that the volume at the fringe of the ellipsoid along the largest axis may not even be within the initial ellipsoid, so its omission may not prove to be crucial. The "reduction factor" φ is an attempt to compensate for the over-estimate that the diagonal ellipsoid makes along the shortest axis. Little can be said about how the new ellipsoid compares to the old along intermediate axes, but we would expect the shorter ones to be enclosed and the longer ones not to be.

Cycling strategies.

"Resetting" amounts to discarding all modification vectors p_j every K iterations. An alternative is to retain K modifications throughout. At each iteration a new update is added but the one from K iterations earlier is discarded. (We call this "cycling" because the new update simply overwrites the old one in storage; the point at which the replacement occurs cycles around a workspace of fixed size.)

It can be shown that at each iteration, this cyclic update gives an ellipsoid that encloses both the ellipsoid from the previous iteration *and* the ellipsoid that would be present had no discards ever been made.

Note that although updates are discarded from B, their effect on x_k and σ is not. This latter point is of some importance, since the volume of the current ellipsoid would otherwise reflect only the last K updates. Also, it is vital in this variation of the algorithm that σ decrease every iteration. This will occur only if ρ is larger than $1/n$. (Hence the introduction of ρ_{min} earlier.)

6. RESULTS AND OBSERVATIONS

Most of the ideas discussed here have been implemented in a Fortran program on an IBM 370/168. Some existing LP models were used as test problems. These were input in standard MPS format and stored in single precision. Since access is required to the rows of the constraint matrix, a row list of its nonzero elements was formed from the usual column list. All computation was performed in double precision (approximately 15 decimal digits).

The dimensions of some of the LP models are as follows:

Name	*Rows*	*Columns*	*Equalities*
WEAPON	12	100	0
SHARE2B	99	79	13
ISRAEL	175	142	0
BANDM	306	472	305
STAIR	357	467	209

Many test runs were made on these and other problems. Figures 4–6 illustrate a typical set of results. The inescapable conclusion is that even the best variant of the ellipsoid algorithm performs exceedingly poorly. The hope that a point would be reached where a "good basis" could be identified was rarely realized, even when a small sum of infeasibilities was attained. Some variants could be said to perform better than others, but the difficulties of comparison were complicated by the fact that convergence does not occur in any conventional sense. There is no quantity (such as the sum of infeasibilities) that decreases monotonically, and only in exceptional circumstances was a feasible point ever found.

The following are some tentative conclusions, observations, and (where possible) explanations.

1. There was usually a rapid initial reduction in the sum of infeasibilities. This was followed by slow but discernible convergence. Eventually the sum of infeasibilities oscillated around a steady-state value. If this value was small enough, a feasible point would occasionally be obtained by chance.
2. Choosing a single constraint is usually much worse than aggregating constraints. (Clearly, reducing one infeasibility without regard to the others will have, in the short term, an unpredictable effect on the total sum of infeasibilities.)
3. Weighting aggregated constraints by r_i can produce oscillations about a (perturbed) equality constraint; i.e., the iterates x_k are reflected back and forth across the constraint and converge only slowly towards it. This was a symptom of the single-constraint strategy when the range ellipsoid was *not* used. It arises when the aggregated range is too large for the features of the range ellipsoid to take effect.

4. Using $\sqrt{r_i}$ as weights tended to reduce the oscillations, because the smaller weights would soon allow some other equality constraint to dominate in the aggregation. It would sometimes result in slightly poorer performance on problems for which oscillation was not a difficulty, but it seemed to be the best compromise.

5. Choosing equal weights was often a poor strategy, since the most violated constraint would sometimes remain the same for thousands of iterations. This is the opposite extreme to oscillation and justifies the previous comment.

6. The resetting frequency K was not critical. Resetting more frequently often resulted in a more rapid initial convergence, particularly if the initial ellipsoid was very large. This was because of the reduction factor φ that was applied each reset. However, the subsequent convergence would usually be slower. The net result was a degree of invariance with respect to the frequency (assuming that even the largest K was small compared to n). The range of values tried was 5, 10, 20, 30, 40, and 50. The value $K = 20$ seems to be a reasonable value in practice. The work and storage per iteration are then roughly equivalent to refactorizing the basis in the simplex method every 50 iterations.

7. Problem SHARE2B was small enough to allow use of $K = 200$ ($= 2.2n$ — normally an unthinkable ratio). This was the one case where termination at a feasible point could reasonably be expected. However, the total work and iterations far exceeded that required by the simplex method to solve the unperturbed problem exactly.

8. In spite of its promising properties, the cycling algorithm did not perform more favorably than the resetting algorithm. In fact, since there was no artificial shrinking of σ at the end of each K iterations, both the initial rate of convergence and the overall performance tended to be worse.

9. Forcing a reduction in σ on resetting eventually results in violated constraints lying outside the current ellipsoid. *Not* forcing a reduction meant poorer initial convergence. Possibly the size of the reduction should be related to the estimated progress made during the last K iterations. Progress is ultimately so slow that this would suppress any artificial reduction each reset. This in turn would *ensure* that progress was slow or non-existent, once ρ_{max} starts forcing an increase in σ during the iterations.

10. In most cases the target objective value was reached at a point where the sum of infeasibilities was reasonably low. (Of course it could also be reached in an early iteration at the expense of gross infeasibility elsewhere.)

11. The algorithm is highly sensitive to scaling — much more so than the simplex method. Scaling the constraints by rows is essential. Scaling them by columns (i.e., scaling the variables) would doubtless help in general, but was not tried. The problem ISRAEL was one with poor column scaling, and although the algorithm was able to reduce the sum of infeasibilities by several orders of magnitude, it was unable to obtain an objective value anywhere close to the target.

12. If the original problem was really one of finding a feasible point and if the volume of the feasible region was large, the algorithm tended to perform much the same as described, but with a somewhat greater chance of terminating. A dramatic improvement could be made in this situation by shifting the constraints *inwards*, i.e., by changing $Ax \geq b$ to $Ax \geq b + \delta b$, where δb is positive. In effect, a feasible point to the original problem was then found in the rapid convergence stage of the algorithm.

13. One of the more difficult matters is to propose a stopping criterion that recognizes the time when further progress is unlikely. Of all the usual quantities that could be monitored, the most stable would probably be the size of the maximum constraint violation, r_i.

Some of these observations are illustrated in the following figures. For comparison, the simplex method was applied to unperturbed linear programs, cast in the form LP2 with t set to the optimal objective value (see section 2). Hence, all iterates except the last were infeasible. The beginning and end of the simplex iterations are marked by a cross.

The ellipsoid algorithms were applied to perturbed problems of the form LP3, with equality-constraint bounds perturbed each way by 5%. (If the i-th components of b_l and b_u were both β, the quantity $\delta\beta = 0.05(|\beta| + 1)$ was computed and the bounds on the i-th row of Ax were taken to be $\beta - \delta\beta$ and $\beta + \delta\beta$.) The perturbation to the objective value was $\delta t = 0.01t$. During the test for feasibility, a constraint was considered violated only if the residual r_i exceeded 0.05.

The curves drawn for the ellipsoid algorithms have been smoothed to show the general trend. The quantity plotted is the *minimum* sum of infeasibilities achieved during the previous 10, 20 or 50 iterations. The actual sum varies erratically with iteration number and would lie above the curves shown.

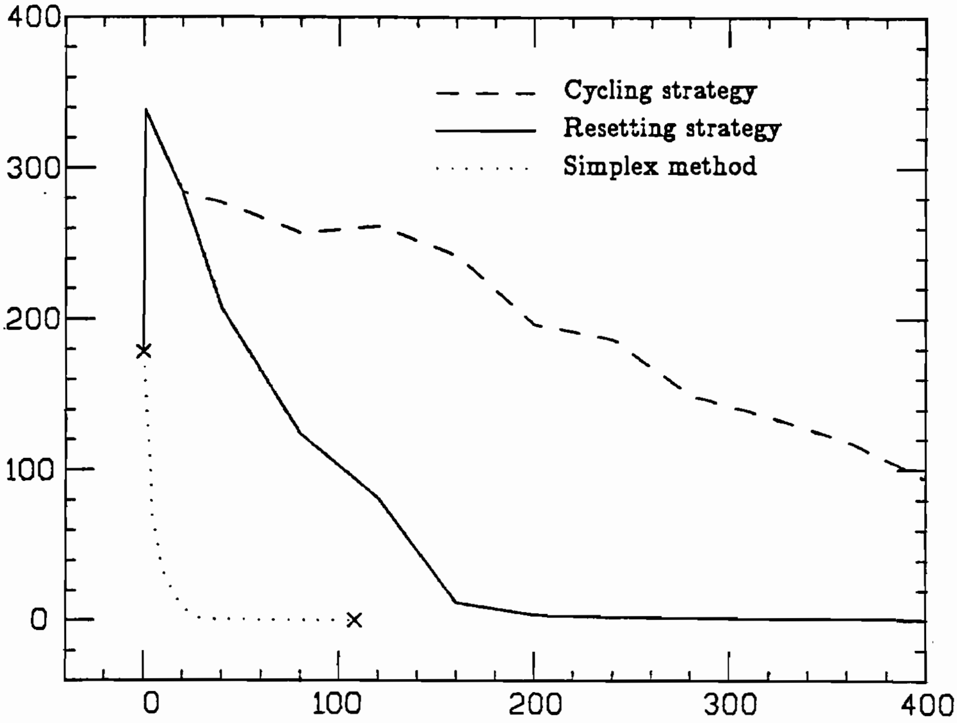


Figure 4. **Problem SHARE2B. Sum of infeasibilities vs. iteration number.**

Comparison of resetting and cycling strategies with large initial radius.
$K = 20, \quad \rho_0 = 0.01, \quad \sigma_0 = 6500, \quad \rho_{min} = 0.01, \quad \omega_i = \sqrt{r_i}.$

1. The sum of infeasibilities increases substantially in the early iterations. This is typical when the initial radius σ_0 is large.
2. The bound $\rho \geq \rho_{min}$ was often active, particularly after each reset. This leads to a more rapid reduction in σ for the resetting strategy, and hence to greater initial progress.
3. Approximate cpu times:
   Ellipsoid algorithm, cycling: 5.6 seconds for 400 iterations.
   Ellipsoid algorithm, resetting: 4.8 seconds for 400 iterations.
   Simplex method: 2.4 seconds for 108 iterations (solution found).

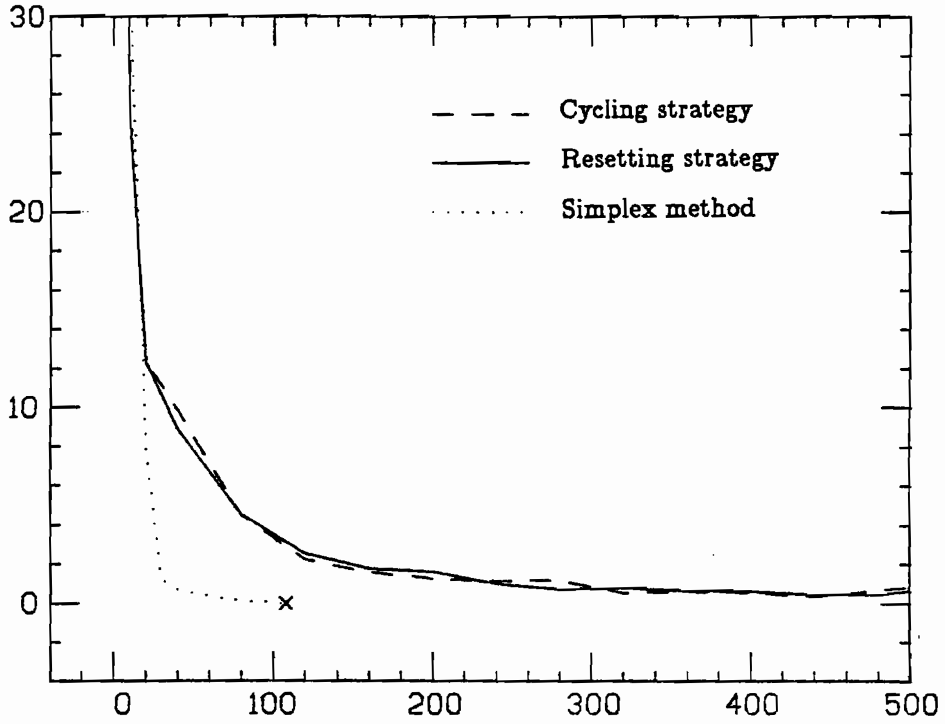


Figure 5. Problem SHARE2B. Sum of infeasibilities vs. iteration number.

Comparison of resetting and cycling strategies with smaller initial radius.
$K = 20, \quad \rho_0 = 0.1, \quad \sigma_0 = 650, \quad \rho_{min} = 0.0127 = 1/\sqrt{n}, \quad \omega_i = \sqrt{r_i}.$

1. The bound ρ_{min} was active for both ellipsoid algorithms during early iterations, reducing σ and allowing rapid initial progress.
2. The finer scale used for the vertical axis illustrates the slow terminal convergence that can be expected in general.

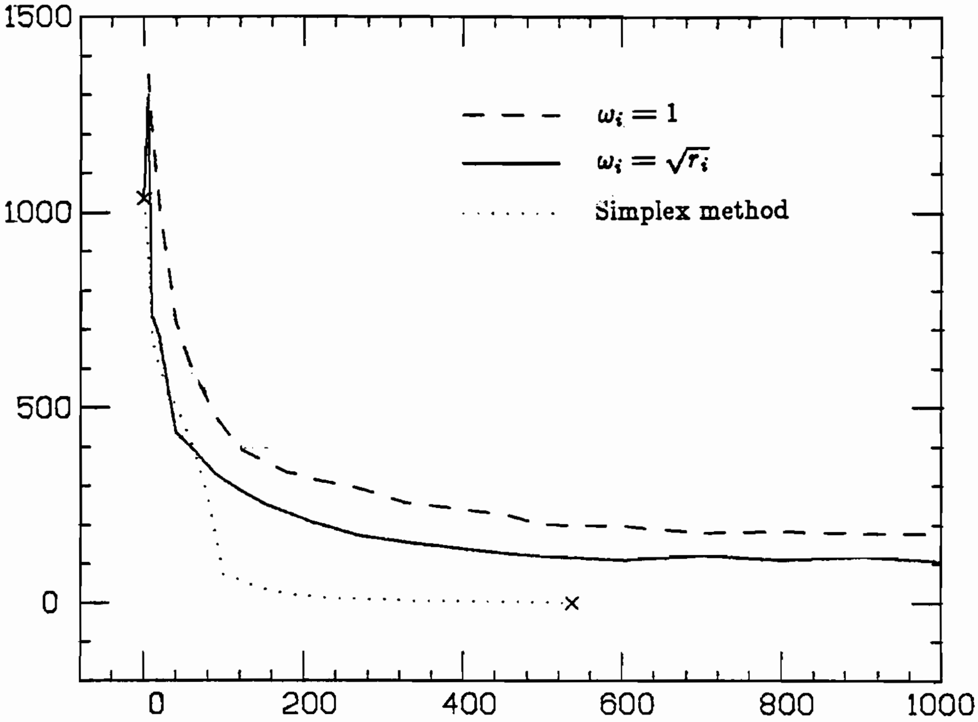


Figure 6. Problem STAIR. Sum of infeasibilities vs. iteration number.

Comparison of weights $\omega_i = 1$ and $\omega_i = \sqrt{r_i}$ in aggregated constraints.
Resetting strategy with $K = 30$.
$\rho_0 = 0.1, \quad \sigma_0 = 2500.$

1. The ellipsoid algorithms could not be expected to converge within a tolerable time.
2. This is a difficult example even for the simplex method (in which the basis factorizations are unusually dense). The workspace required by the simplex method is slightly more than that needed by the ellipsoid algorithms (storing 30 updates).
3. Approximate cpu times:
   Ellipsoid algorithms: 53 seconds for 1000 iterations.
   Simplex method: 50 seconds for 537 iterations (solution found).

Final comments.

The main hope was that a feasible point to a perturbed linear program could be found that would suggest a good set of basic variables in the LP sense. This hope was not realized. There are several reasons for this, particularly with large problems. In practice, there are frequently many Lagrange multipliers (dual variables) that are zero or close to zero. This means that a feasible point to the perturbed problem need not be close to any vertex. Also, the size of the perturbations used was sufficiently large that for a many-variable problem, an accumulation of small changes in some variables could allow other variables to take on values quite unlike their optimal values.

Given a system of linear equations $Bx = b$, the effect on x of a perturbation to b has been studied at length (e.g., Wilkinson, 1965). For example, if the components of b were perturbed by 1% and if the condition number of B were greater than 100, then the perturbed solution may be different from x in all figures. In most linear programs, we would expect the condition number of the basis matrix to be 100 at least. Hence, even if a feasible solution could be found to a perturbed problem, we could not expect to recognize the solution if the perturbations were as much as 1%. On the other hand, for problems containing any equality constraints, the hope for obtaining a solution when the perturbations are as *small* as 1% would seem to be remote.

Regarding convergence, the difficulty remains that if the problem is too large to allow the ellipsoid matrix B to be stored and updated continuously, then the assurance of convergence is lost. In spite of certain optimism during the early stages of this research, we can only conclude that for large-scale linear programs, the prospects for developing an efficient ellipsoid algorithm are indeed quite bleak.

REFERENCES

Aspvall, B. and Stone, R. E. (1979). Khachiyan's linear programming algorithm, Report STAN-CS-79-776, Computer Science Department, Stanford University, California (November 1979).

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, New Jersey.

Gill, P. E., Murray, W. and Saunders, M. A. (1975). Methods for computing and modifying the *LDV* factors of a matrix, *Math. Comp.* 29, 132, pp. 1051-1077.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming, *Doklady Akademiia Nauk SSSR Novaia Seriia* 244:5, pp. 1093–1096. [English translation in *Soviet Mathematics Doklady* 20:1 (1979), pp. 191–194.]

Lawler, E. L. (1980). The great mathematical sputnik of 1979, University of California, Berkeley, California (February 1980).

Wilkinson, J. H. (1965). *The Algebraic Eigenvalue Problem,* The Clarendon Press, Oxford.

Wolfe, P. (1980a). A bibliography for the ellipsoid algorithm, IBM Research Center Report No. 8237 (April 1980).

Wolfe, P. (1980b). The ellipsoid algorithm, in *Optima,* 1 (Newsletter of the Mathematical Programming Society, June 1980).

THE ELLIPSOID METHOD AND ITS CONSEQUENCES IN COMBINATORIAL OPTIMIZATION

M. Grötschel,* L. Lovász,** and A. Schrijver***†

*Institut für Ökonometrie und Operations Research, Universität Bonn
**Bolyai Institute, Josef Attila University, Szeged
***Mathematisch Centrum, Amsterdam

L.G. Khachian recently published a polynomial algorithm to check feasibility of a system of linear inequalities. The method is an adaptation of an algorithm proposed by Shor for non-linear optimization problems. In this paper we show that the method also yields interesting results in combinatorial optimization. Thus it yields polynomial algorithms for vertex-packing in perfect graphs; for the matching and matroid intersection problems; for optimum covering of directed cuts of a digraph; for the minimum value of a submodular set function; and for other important combinatorial problems. On the negative side, it yields a proof that weighted fractional chromatic number is NP-hard.

†Research supported by the Netherlands Organisation for the Advancement of Pure Research (ZWO).

0. INTRODUCTION.

A typical problem in combinatorial optimization is the following. Given a finite set S of vectors in $\mathbb{R}^n$ and a linear objective function c^Tx, find

$$(1) \qquad \max\{c^Tx \mid x \in S\}.$$

Generally S is large (say exponential in n) but highly structured. For example, S may consist of all characteristic vectors of perfect matchings in a graph. We are interested in finding the value of (1) by an algorithm whose running time is polynomial in n. Therefore, enumerating the elements of S is not a satisfactory solution.

The following approach was proposed by Edmonds [1965], Ford and Fulkerson [1962] and Hoffman [1960], and is the classical approach in combinatorial optimization. Let P denote the convex hull of S. Then clearly

$$(2) \qquad \max\{c^Tx \mid x \in S\} = \max\{c^Tx \mid x \in P\}.$$

The right hand side here is a linear programming problem: maximize a linear objective function on a polytope. Of course, to be able to apply the methods of linear programming, we have to represent P as the set of solutions of a system of linear inequalities. Such a representation, of course, always exists, but our ability to find the necessary inequalities depends on the structure of S. However, in many cases these inequalities (the facets of P) can be described. There are some beautiful theorems of this kind, e.g. Edmonds' description of the matching polytope. In these cases, the methods of linear programming can be applied to solve (1). However, until about a year ago there were two main obstacles in carrying out the above program even for nice sets S like the set of perfect matchings. First, no algorithm to solve linear programming with polynomial running time in the worst case was known. Second, the number of inequalities describing S is typically large (exponential) and hence even to formulate the linear program takes exponential space and time. Indeed, the well-known efficient combinatorial algorithms, like Edmonds' matching algorithm [1965] or Lucchesi's algorithm to find optimum coverings for directed cuts [1976] are based on different - ad hoc - ideas.

A recent algorithm to solve linear programs due to L.G. Khachian [1979], based on a method of Shorr [1970], removes both difficulties. Its running time is polynomial; also, it is very insensitive to the number of constraints in the following sense: we do not need to list the faces in advance, but only need a subroutine which recognizes feasibility of a vector and if it is infeasible then computes a hyperplane separating it from P. Searching for such a hyperplane is another combinatorial optimization problem which is often much easier to solve. It is interesting that if we want to apply the same method to this second problem, we get back the first one.

The main purpose of this paper is to exploit this equivalence between problems. After formulating the optimization problem in Chapter 1 exactly, we survey the "ellipsoid method" in Chapter 2. In Chapter 3 we prove the equivalence of the optimization and the separation problem, and their equivalence with other optimization problems. So we show that optimum dual solutions can be obtained by the method (since the dual problem has, generally in combinatorial problems, exponentially many variables, the method cannot be applied to the dual directly). Chapter 4 contains applications to the matching, matroid intersection, and branching problems, while in Chapter 5 we show how to apply the method to minimize a submodular set function and, as an application, to give algorithmic versions of some results of Edmonds and Giles [1977] and Frank [1979]. These include an algorithm to find optimum covering of directed cuts in a graph, solved first by Lucchesi [1976].

It is interesting to point out that these applications rely on the deep theorems characterizing facets of the corresponding polytope. This is in quite a contrast to previously known algorithms, which typically do not use these characterizations but quite often give them as a by-product.

The efficiency of the algorithms we give is polynomial but it seems much worse than those algorithms developed before. Even if we assume that this efficiency can be improved with more work, we do not consider it the purpose of our work to compete with the special-purpose algorithms. The main point is that the ellipsoid method proves the polynomial solvability of a large number of different combinatorial optimization problems at once, and hereby points out directions for the search for practically feasible polynomial algorithms.

Chapter 6 contains an algorithm to find maximum independent sets in perfect graphs. The algorithm makes use of a number $\vartheta(G)$ introduced by one of the authors as an estimation for the Shannon capacity of a graph (Lovász [1979]). Finally, in Chapter 7 we note that the vertex-packing problem of a graph is in a sense equivalent to the fractional chromatic number problem, and comment on the phenomenon that this latter problem is an example of a problem in NP which is NP-hard but (as for now) not known to be NP-complete.

1. OPTIMIZATION ON CONVEX BODIES: FORMULATION OF THE PROBLEMS AND THE RESULTS

Let K be a non-empty convex compact set in $\mathbb{R}^n$. We formulate the following two algorithmic problems in connection with K.

(1) *Strong optimization problem:* given a vector $c \in \mathbb{R}^n$, find a vector x in K which maximizes $c^T x$ on K.

(2) *Strong separation problem:* given a vector $y \in \mathbb{R}^n$, decide if $y \in K$, and if not, find a hyperplane which separates y from K; more exactly, find a vector $c \in \mathbb{R}^n$ such that $c^T y > \max\{c^T x \mid x \in K\}$.

Examples. Let K be the set of solutions of a system of linear inequalities

$$a_i^T x \le b_i \qquad (i = 1,\dots,m) \tag{3}$$

($a_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$). Then the strong separation problem can be solved trivially: we substitute x = y in the constraints. If each of them is satisfied, $y \in K$. If constraint $a_i^T x \le b_i$ is violated, it yields a separating hyperplane. On the other hand, the optimization problem on K is just the linear programming problem.

As a second example, let K be given as the convex hull of a set $\{v_1,\dots,v_m\}$ of points in $\mathbb{R}^n$. Then the optimization problem is easily solved by evaluating the objective function at each of the given points and selecting the maximum. On the other hand, to solve the separation problem we have to find a vector c in $\mathbb{R}^n$ such that

$$c^T y > c^T v_i \qquad (i = 1,\dots,m) \tag{4}$$

So this problem requires finding a feasible solution to a system of linear inequalities; this is again essentially the same as linear programming.

Note that the convex hull of $\{v_1,\dots,v_m\}$ is, of course, a polytope and so it can be described as the set of solutions of a system of linear inequalities as well. But the number of these inequalities may be very large compared to m and n, and so their determination and the checking is too long. This illustrates that the solvability of the optimization and separation problems depends on the way K is given and not only on K.

We do not want to make any a priori arithmetical assumption on K. Thus it may well be that the vector in K maximizing $c^T x$ has irrational coordinates. In this case the formulation of the problem is not correct, since it is not clear how to state the answer. Therefore we have to formulate two weaker and more complicated, but more correct problems.

(5) *(Weak) optimization problem*: given a vector $c \in \mathbb{R}^n$ and a number $\varepsilon > 0$, find a vector $y \in \mathbb{R}^n$ such that $d(y,K) \le \varepsilon$ and y almost maximizes $c^T x$ on K, i.e. for every $x \in K$, $c^T x \le c^T y + \varepsilon$.

(6) *(Weak) separation problem*: given a vector $y \in \mathbb{R}^n$ and a number $\varepsilon > 0$, conclude with one of the following: (i) asserting that $d(y,K) \le \varepsilon$; (ii) finding a vector $c \in \mathbb{R}^n$ such that $\|c\| \ge 1$ and for every $x \in K$, $c^T x \le c^T y + \varepsilon$.

We shall always assume that we are given a point a_0 and $0 < r \le R$ such that

$$S(a_0,r) \subseteq K \subseteq S(a_0,R). \tag{7}$$

The second inclusion here simply means that K is bounded, where a bound is known explicitly; this is quite natural to assume both in theoretical and in (possible) practical applications. The first assumption, namely that K contains an explicit ball, is much less natural and we make it for purely technical reasons. What it really means is that K is full-dimensional, or at least we can find the affine subspace it spans and also that we can find a ball in this subspace contained in K. At the end of Chapter 3 we shall show that some assumption like this must be made.

So we define a *convex body* as a quintuple $(K;n,a_0,r,R)$ such that $n \ge 1$, K is a convex set in $\mathbb{R}^n$, $a_0 \in K$, $0 < r \le R$ and (7) is satisfied.

Let $\mathcal{K}$ be a class of compact convex bodies. We assume that each $K \in \mathcal{K}$ has some encoding. An *input* of the optimization problem for $\mathcal{K}$ is then the code of some member K of $\mathcal{K}$, a vector $c \in \mathbb{R}^n$, and a number $\varepsilon > 0$. Inputs of the other problems are defined similarly. The *length* of the input is defined in the (usual) binary encoding. Thus the length of the input is at least $n + |\log r| + |\log R| + |\log \varepsilon|$. An algorithm to solve the optimization problem for the class $\mathcal{K}$ is called *polynomial* if its running time is bounded by some polynomial of the size of the input.

The fact that the running time must be polynomial in $|\log \varepsilon|$ is crucial: it means that running the algorithm for $\varepsilon = \frac{1}{2}, \frac{1}{4}, \ldots$ we get a sequence of approximations which converge exponentially fast in the running time. Other approximation algorithms for linear programming (Motzkin and Schoenberg [1954]) have only polynomial convergence speed. This exponential conergence rate enables

Khachian to obtain exact optimum in polynomial time (essentially by rounding) and us to give the combinatorial applications in this paper.

2. THE ELLIPSOID METHOD

Let us first describe the simple geometric idea behind the method. We start with a convex body K, included in a ball $S(a_0,R) = E_0$, and a linear objective function c^Tx. In the k-th step there will be an ellipsoid E_k, which includes the set K_k of those points x of K for which c^Tx is at least as large as the best found so far. We look at the centre x_k of E_k. If $x_k \notin K$ then we take a hyperplane through x_k which avoids K. This hyperplane cuts E_k into two halves; we pick that one which includes K_k and include it in a new ellipsoid E_{k+1}, "smaller" than E_k. If $x_k \in K$ then we cut with the hyperplane $c^Tx = c^Tx_k$ similarly. The volumes of the ellipsoids E_k will tend to 0 exponentially and this guarantees that those centres x_k which are in K will tend to an optimum solution exponentially fast.

We now turn to the exact formulation of the procedure. Let $K \subseteq \mathbb{R}^n$ be a compact convex set, $S(a_0,r) \subseteq K \subseteq S(a_0,R)$, c^Tx a linear objective function, $\|c\| \geq 1$ and $\varepsilon > 0$. Assume that there is a subroutine SEP to solve the separation problem for K. This means that given a vector $y \in \mathbb{R}^n$ and $\delta > 0$, SEP either concludes that $y \in S(K,\delta)$ or yields a vector d such that

$$\max\{d^Tx \mid x \in K\} \leq d^Ty + \delta. \tag{1}$$

To solve the optimization problem on K we run the following algorithm. Let

$$N = 4n^2 \left\lceil \log \frac{2R^2\|c\|}{r\varepsilon} \right\rceil, \tag{2}$$

$$\delta = \frac{R^2 4^{-N}}{24(n-1)}, \tag{3}$$

and

$$p = 5N \left\lceil \log \frac{12\sqrt{n}}{R^2} \right\rceil. \tag{4}$$

We now define a sequence $x_0, x_1, \ldots$ of vectors and a sequence $A_0, A_1, \ldots$ of positive definite matrices as follows. Let $x_0 = a_0$ and $A_0 = R^2I$. Assuming that x_k, A_k are defined, we run the subroutine SEP with $y = x_k$ and δ. If it concludes that $x_k \in S(K,\delta)$ we say that k is a *feasible index*, and set $a = c$. If SEP yields a vector $d \in \mathbb{R}^n$ such that $\|d\| \geq 1$ and

$$(5)\qquad \max\{d^Tx \mid x \in K\} \le d^Tx_k + \delta,$$

then we call k an *infeasible index* and let a = -d. Next define

$$(6)\qquad b_k = A_ka/\sqrt{a^TA_ka},$$

$$(7)\qquad x_k^* = x_k + \frac{1}{n+1}b_k,$$

$$(8)\qquad A_k^* = \frac{2n^2+3}{2n^2}(A_k - \frac{2}{n+1}b_kb_k^T),$$

and

$$(9)\qquad x_{k+1} \approx x_k^*, \text{ and } A_{k+1} \approx A_k^*,$$

where the sign ≈ means that the left hand side is obtained by rounding the right hand side to p binary digits, taking care that A_{k+1} is symmetric.

The sequence (x_k), k feasible, will give good approximations for the optimum solution of our problem. To prove this, we shall need some lemmas, which will also illuminate the geometric background of the algorithm.

First we introduce some further notation. Let

$$(10)\qquad E_k = \{x \in \mathbb{R}^n \mid (x-x_k)^TA_k^{-1}(x-x_k) \le 1\},$$

and

$$(11)\qquad E_k^* = \{x \in \mathbb{R}^n \mid (x-x_k^*)^TA_k^{*-1}(x-x_k^*) \le 1\}.$$

(2.1) LEMMA. *The matrices* A_0, A_1, ... *are positive definite. Moreover,*

$$(12)\qquad \|x_k\| \le \|a_0\| + R.2^k,\ \|A_k\| \le R^2.2^k, \textit{ and } \|A_k^{-1}\| \le R^{-2}.4^k.$$

PROOF. By induction on k. For k = 0 all the statements are obvious. Assume that they are true for k. Then note first that

$$(13)\qquad \|A_k^*\| = \frac{2n^2+3}{2n^2}\|A_k - \frac{2}{n+1}b_kb_k^T\| \le \frac{2n^2+3}{2n^2}\|A_k\| \le (1+\frac{3}{2n^2})R^2.2^k$$

and so

$$\|A_{k+1}\| \le \|A_k^*\| + \|A_{k+1} - A_k^*\| \le (1+\frac{3}{2n^2})R^2.2^k + n.2^{-p} \le R^2.2^{k+1}. \tag{14}$$

Further,

$$\|b_k\| = \frac{\|A_k a\|}{\sqrt{a^T A_k a}} = \sqrt{\frac{a^T A_k^2 a}{a^T A_k a}} \le \sqrt{\|A_k\|} \le R.2^k, \tag{15}$$

and so

$$\|x_{k+1}\| \le \|x_k\| + \frac{1}{n+1}\|b_k\| + \|x_{k+1} - x_k^*\| \le \|a_0\| + R.2^k + \frac{1}{n+1}R.2^k + \sqrt{n}.2^{-p} \le \tag{16}$$

$$\le \|a_0\| + R.2^{k+1}.$$

Finally we have

$$(A_k^*)^{-1} = \frac{2n^2}{2n^2+3}(A_k^{-1} + \frac{2}{n-1} \cdot \frac{aa^T}{a^T A_k a}), \tag{17}$$

as it is easy to verify bu computation, and hence A_k is positive definite. Further,

$$\|(A_k^*)^{-1}\| \le \frac{2n}{2n+3}(\|A_k^{-1}\| + \frac{2}{n-1} \cdot \frac{\|a\|^2}{a^T A_k a}) \le \frac{2n^2}{2n^2+3}(\|A_k^{-1}\| + \frac{2}{n-1}\|A_k^{-1}\|) < \frac{n+1}{n-1}\|A_k^{-1}\|. \tag{18}$$

Let λ_0 denote the least eigenvalue of A_{k+1} and let v be a corresponding eigenvector, $\|v\| = 1$. Then

$$\lambda_0 = v^T A_{k+1} v = v^T A_k^* v + v^T(A_{k+1} - A_k^*)v \ge \|(A_k^*)^{-1}\|^{-1} - \|A_{k+1} - A_k^*\| \ge \tag{19}$$

$$\ge \frac{n-1}{n+1}\|A_k^{-1}\|^{-1} - \sqrt{n}.2^{-p} \ge \frac{n-1}{n+1}. R^2.4^{-k} - \sqrt{n}.2^{-p} > R^2.4^{-(k+1)}.$$

This proves that A_{k+1} is positive definite and also that

$$\|A_{k+1}^{-1}\| = 1/\lambda_0 \le R^{-2}.4^{k+1}. \quad \square \tag{20}$$

(2.2) LEMMA. *Let* μ *denote the* n*-dimensional volume. Then*

$$\frac{\mu(E_{k+1})}{\mu(E_k)} < e^{-1/4n}. \tag{21}$$

PROOF. See Gács and Lovász [1979]. □

Set

(22) $$\zeta_k = \max\{c^T x_j \mid 0 \le j < k,\ j \text{ feasible}\},$$

and

(23) $$K_k = K \cap \{x \mid c^T x \ge \zeta_k\}.$$

<u>(2.3) LEMMA</u>. $E_k \supseteq K_k$, *for* $k = 0,1,\ldots,N$.

<u>PROOF</u>. By induction on k. For $k = 0$ the assertion is obvious. Let $x \in K_{k+1}$. Then

(24) $$x \in K_k \subseteq E_k,$$

and also

(25) $$a_k^T x \ge a_k^T x_k - \delta,$$

where a_k equals the auxiliary vector a used in step k (if k is a feasible index we do not even have the δ here). Write

(26) $$x = x_k + y + tb_k,$$

where $a_k^T y = 0$ (since b_k and a_k are not perpendicular because of the positive definiteness of A_k, such a decomposition of x always exists). By (24),

(27) $$1 \ge (y+tb_k)^T A_k^{-1}(y+tb_k) = y^T A_k^{-1} y + t^2 b_k^T A_k^{-1} b_k = y^T A_k^{-1} y + t^2.$$

Hence $t \le 1$. On the other hand, (25) yields

(28) $$-\delta \le t a_k^T b_k = t\sqrt{a_k^T A_k a_k}.$$

Now we have

(29) $$(x-x_{k+1})^T A_{k+1}^{-1}(x-x_{k+1}) \le (x-x_k^*) A_k^{*-1}(x-x_k^*) + R_1,$$

where the remainder term R_1 can be estimated easily by similar methods to those in the proof of Lemma (2.1), and it turns out that $R_1 < 1/12n^2$.

For the main term we have

$$(30)\qquad (x-x_k^*)^T A_k^{*-1}(x-x_k^*) = \frac{2n^2}{2n^2+3}\Big((t-\frac{1}{n-1})b_k+y\Big)^T\Big(A_k^{-1}+\frac{2}{n-1}\cdot\frac{aa^T}{a^TA_ka}\Big)\Big((t-\frac{1}{n+1})b_k+y\Big) =$$

$$= \frac{2n^2}{2n^2+3}\Big((t-\frac{1}{n+1})^2 + y^TA_k^{-1}y + \frac{2}{n-1}(t-\frac{1}{n+1})^2\Big) \le \frac{2n^2}{2n^2+3}\Big(\frac{n^2}{n^2-1} - \frac{2t(1-t)}{n-1}\Big) \le$$

$$\le \frac{2n^4}{(2n^2+3)(n^2-1)} + \frac{4\delta}{(n-1)\sqrt{a^TA_ka}} \le 1 - \frac{1}{6n^2} + \frac{4\delta}{n-1}\|A_k^{-1}\| \le 1 - \frac{1}{6n^2} + \frac{4\delta R^{-2}.4^N}{n-1}.$$

Hence $(x-x_{k+1})^T A_{k+1}(x-x_{k+1}) \le 1$, and so $x \in E_{k+1}$. □

Now we are able to prove the main theorem in this section.

(2.4) THEOREM. *Let* j *be a feasible index for which*

$$(31)\qquad c^Tx_j = \max\{c^Tx_k \mid 0 \le k < N,\ k \text{ feasible}\}.$$

Then $c^Tx_j \ge \max\{c^Tx \mid x \in K\} - \varepsilon$.

PROOF. Let us observe first that Lemmas (2.2) and (2.3) imply that

$$(32)\qquad \mu(K_N) \le \mu(E_N) \le e^{-N/4n}\mu(E_0) = e^{-N/4n}R^nV_n,$$

where V_n is the volume of the n-dimensional unit ball. On the other hand, let

$$(33)\qquad \zeta = \max\{c^Tx \mid x \in K\}$$

and $y \in K$ such that $c^Ty = \zeta$. Consider the cone whose base is the (n-1)-dimensional ball of radius r and centre x_0 in the hyperplane $c^Tx = c^Tx_0$ and whose vertex is y. The piece of this cone in the half-space $c^Tx \ge c^Tx_j$ is contained in K_N. The volume of this piece is

$$(34)\qquad \frac{V_{n-1}.r^{n-1}.(\zeta-c^Tx_0)}{n\|c\|}\Big(\frac{\zeta-c^Tx_j}{\zeta-c^Tx_0}\Big)^n \le \mu(K_N) \le e^{-N/4n}.R^nV_n.$$

Hence

$$(35)\qquad \zeta - c^Tx_j \le e^{-N/4n^2}.R.\Big(\frac{\zeta-c^Tx_0}{r}\Big)^{\frac{n-1}{n}}\Big(\frac{nV_n}{V_{n-1}}\Big)^{1/n}\|c\|^{1/n}.$$

We still need an upper bound on ζ. Since

(36) $$|\zeta - c^T x_0| = |c^T(y-x_0)| \le \|c\|.\|y-x_0\| \le R.\|c\|,$$

we finally have

(37) $$\zeta - c^T x_j < 2e^{-N/4n}.\frac{R^2}{r}\|c\| \le \varepsilon. \quad \Box$$

3. EQUIVALENCE OF OPTIMIZATION AND OTHER PROBLEMS

First we prove the equivalence of the separation problem and the optimization problem, for any given K. More exactly, this means the following.

(3.1) THEOREM. *Let $\mathcal{K}$ be a class of convex bodies. There is a polynomial algorithm to solve the separation problem for the members of $\mathcal{K}$, if and only if there is a polynomial algorithm to solve the optimization problem for the members of $\mathcal{K}$.*

A class $\mathcal{K}$ with this property will be called *solvable*.

PROOF. I. The "only if" part. In view of the results of Chapter 2, the only thing to check is that the algorithm described there is polynomial-bounded. This follows since by assumption, the subroutine SEP is polynomial, hence the number of digits in the entries of a is polynomial and so the computation of x_{k+1} and A_{k+1} requires only a polynomial number of steps. All other numbers occurring have only a polynomial number of digits, by Lemma (2.1). The number of iterations is also polynomial. Hence the algorithm runs in polynomial time.

II. The "if" part. Without loss of generality assume that $a_0 = 0$. Let K^* be the polar of K, i.e.,

(1) $$K^* = \{u \mid u^T x \le 1 \text{ for each } x \in K\}.$$

It is well-known that K^* is a convex body, and

(2) $$S(0,1/R) \subseteq K^* \subseteq S(0,1/r).$$

If $\mathcal{K}$ is a class of convex bodies with $a_0 = 0$, let $\mathcal{K}^* = \{K^* \mid K \in \mathcal{K}\}$.

(3.2) LEMMA. *The separation problem for a class $\mathcal{K}$ of convex bodies with $a_0 = 0$ is polynomially solvable iff the optimization problem is polynomially solvable for the class $\mathcal{K}^*$.*

Since $(\mathcal{K}^*)^* = \mathcal{K}$, this lemma immediately implies the "if" part of the theorem: if the optimization problem is polynomially solvable for $\mathcal{K}$ then the separation problem is polynomially solvable for $\mathcal{K}^*$. But then by part I, the optimization problem is polynomially solvable for $\mathcal{K}^*$ and so using the lemma again, it follows that the separation problem is polynomially solvable for $\mathcal{K}$.

PROOF OF THE LEMMA. I. The "if" part. Let $K^* \in \mathcal{K}^*$, $v \in \mathbb{R}^n$ and $\varepsilon > 0$. Using the optimization subroutine for K, with objective function v and error $\varepsilon.r$, we get a vector $z \in \mathbb{R}^n$ such that $d(z,K) \le \varepsilon r$, and

$$(3) \qquad v^Tz \ge \max\{v^Tx \mid x \in K\} - \varepsilon r.$$

Now if $v^Tz \le 1$ then $v^Tx \le 1 + \varepsilon r$ and hence $v_0 = \frac{1}{1+\varepsilon r} v \in K^*$, whence $d(v,K^*) \le \varepsilon$.

On the other hand, if $v^Tz > 1$ then z is a solution of the separation problem for K^*. In fact, let $z_0 \in K$ such that $\|z-z_0\| \le \varepsilon r$. Then for every $u \in K^*$,

$$(4) \qquad z^Tu = (z-z_0)^Tu + z_0u \le \|u\|.\|z-z_0\| + 1 \le \varepsilon + z^Tv,$$

which proves that z is a solution of the separation problem for K^*.

II. The "only if" part follows by interchanging the roles of $\mathcal{K}$ and $\mathcal{K}^*$. □

Let $\mathcal{K}$ and $\mathcal{L}$ be two classes of convex bodies. Define

$$(5) \qquad \mathcal{K} \wedge \mathcal{L} = \{K \cap L \mid K \in \mathcal{K},\ L \in \mathcal{L},\ \dim K = \dim L,\ a_0(K) = a_0(L)\}.$$

(3.3) COROLLARY. *If $\mathcal{K}$ and $\mathcal{L}$ are solvable then so is $\mathcal{K} \wedge \mathcal{L}$.*

PROOF. The separation problem for $\mathcal{K} \wedge \mathcal{L}$ goes trivially back to the separation problems for $\mathcal{K}$ and $\mathcal{L}$. □

(3.4) COROLLARY. *Let $\mathcal{K}$ be a class of convex bodies with $a_0 = 0$. Then $\mathcal{K}$ is solvable iff $\mathcal{K}^*$ is solvable.*

The proof is trivial by Lemma (3.2).

Let $\mathbb{R}^n_+$ be the non-negative orthant in $\mathbb{R}^n$. Next we study convex bodies K such that there are $\rho > 0$, $R > 0$ with

$$(6) \qquad \mathbb{R}^n_+ \cap S(0,\rho) \subseteq K \subseteq \mathbb{R}^n_+ \cap S(0,R).$$

The *anti-blocker* of K is defined by

$$(7)\qquad A(K) = \{y \in \mathbb{R}^n \mid y^T x \le 1 \text{ for every } x \in K\}.$$

Moreover, $A(\mathcal{K}) = \{A(K) \mid K \in \mathcal{K}\}$.

(3.5) COROLLARY. *Let $\mathcal{K}$ be a class of convex bodies satisfying* (6). *Then $\mathcal{K}$ is solvable iff $A(\mathcal{K})$ is solvable.*

The *proof* is the same as that of Lemma (3.2).

Next we want to show that without the assumption that K contains a ball, there is no algorithm at all to solve the optimization problem. More exactly, consider the class of polytopes $(K_\lambda; 2, \underline{0}, *, 1)$, where the * means that no ball is supposed to be contained in K_λ, and

$$(8)\qquad K_\lambda = \{(x_1, x_2) \mid 0 \le x_1 \le \lambda,\ x_2 = (\tfrac{1}{2} + \tfrac{1}{2}\sqrt{5})x_1\}.$$

First note that if λ is known then both optimization and separation algorithms are easily given, even in the strong sense: $c^T x$ is maximized by either $(0,0)$ or $(\lambda, (\tfrac{1}{2}+\tfrac{1}{2}\sqrt{5})\lambda)$; and if $(y_1, y_2) \in \mathbb{R}^2$ then let

$$(9)\qquad c = \begin{cases} (-1,0) & \text{if } y_1 < 0; \\ (0,-1) & \text{if } y_1 \ge 0,\ y_2 < 0; \\ (-y_1-2y_2, y_1+y_2) & \text{if } y_1 \ge 0,\ y_2 \ge 0,\ y_1^2+y_1y_2-y_2^2 < 0; \\ (y_1+2y_2, -y_1-y_2) & \text{if } y_1 \ge 0,\ y_2 \ge 0,\ y_1^2+y_1y_2-y_2^2 > 0; \\ (1,0) & \text{if } y_1 > \lambda,\ y_2 \ge 0,\ y_1^2+y_1y_2-y_2^2 = 0. \end{cases}$$

If $0 \le y_1 \le \lambda$, $y_2 \ge 0$, and $y_1^2+y_1y_2-y_2^2 = 0$ then conclude $(y_1,y_2) \in K_\lambda$. It is easy to check that this algorithm solves the separation problem for K_λ.

On the other hand, we show that there is no algorithm at all (even arbitrarily slow) which would use a *separation oracle* for the class $\{K_\lambda \mid 0 \le \lambda \le 1\}$, and would be able to maximize the objective function x_1 (i.e., $c^T x$ with $c = (1,0)$) over K_λ. By this we mean an algorithm whose input is a block box which is known to solve the strong separation problem for one of the K_λ's; but the box cannot be broken open to see which value of λ is there; and the algorithm should work regardless which separation algorithm is used by the black box. In particular, it must work if we use the above-described separation algorithm. But then *our algorithm runs the same way for every* $0 \le \lambda \le 1$: the only way it could distinguish between different K_λ's is to run the subroutine with an input vector (y_1,y_2) such that $y_1 > 0$ and $y_1^2+y_1y_2-y_2^2 = 0$. But then one of y_1,y_2 is irrational and so it cannot be the input of an algorithm. So the algorithm *cannot* determine λ, a contradiction. Thus no such algorithm may exist.

Finally we show that for polytopes many of the results are even nicer. By a *rational polytope* we mean a quadruple $(P;n,a_0,T)$ where P is a full-dimensional polytope (in $\mathbb{R}^n$), $a_0 \in \operatorname{Int} P$, and every component of a_0 as well as of every vertex of P is a rational number with numerator and denominator not exceeding T in absolute value. (This definition is much in the spirit of our previous discussion: the vertices of P must be rational in order to be able to explicitly present them and explicit bounds must be known for their complexity.)

<u>(3.6) THEOREM</u>. *Let $(P;n,a_0,T)$ be a rational polytope. Then $S(a_0,r) \subseteq P \subseteq S(a_0,R)$, where $R = 2nT$ and $r = (2T)^{-n^2-n}$. Furthermore, every facet of P can be written as $a^Tx \leq b$, where a $(\neq 0)$ is an integral vector, b is an integer, and the entries of a as well as b are less than $T' = (nT)^n$.*

Thus every rational polytope can be viewed as a convex body, with r and R as above. A certain converse of this assertion holds as well.

<u>(3.7) THEOREM</u>. *Let $P \subseteq \mathbb{R}^n$ be a polytope, $a_0 \in \operatorname{Int} P$, and assume that every component of a_0 is a rational number with numerator and denominator less than T in absolute value. Also assume that every facet of P can be written as $a^Tx \leq b$, where a $(\neq 0)$ is an integral vector, b is an integer and the entries of a as well as b are less than T in absolute value. Then $(P;n,a_0,T')$ is a rational polytope where $T' = (nT)^n$.*

The proof of these two theorems is rather straightforward arithmetic and is omitted (cf. Lemmas 1-2 in Gács and Lovász [1979]).

<u>(3.8) THEOREM</u>. *Let K be a class of rational polytopes. Suppose that K is solvable. Then the strong optimization problem and the strong separation problem are solvable for K in time polynomial in n, $\log T$, and $\log \|c\|$ (respectively $\log S$, where S is the maximum of the absolute values of the numerators and denominators occurring in y).*

PROOF. Let $(P;n,a_0,T) \in \mathcal{K}$, $Q = 2T^2$, and

$$(10) \qquad d = Q^n c + (1,Q,\dots,Q^{n-1})^T.$$

We prove that $\max\{d^T x \mid x \in P\}$ is attained at a unique vertex of P and that this vertex maximizes $c^T x$ as well.

For let x_0 be a vertex of P maximizing $d^T x$ and let x_1 be another vertex. Write

$$(11) \qquad x_0 - x_1 = \frac{1}{\alpha} z,$$

where $0 < \alpha < T^2$ is an integer and $z = (z_1,\dots,z_n)^T$ is an integral vector with $|z_i| < 2T^2 = Q$. Then

$$(12) \qquad 0 \le d^T(x_0-x_1) = \frac{1}{\alpha}\{ Q^n c^T z + \sum_{j=1}^{n} Q^{j-1} z_j \}.$$

Here $c^T z \ge 0$, since it is an integer and if $c^T z \le -1$, then the first term in the bracket is larger in absolute value than the second. Hence

$$(13) \qquad c^T z = c^T(x_0-x_1) \ge 0$$

for every vertex x_1, i.e., x_0 indeed maximizes the objective function $c^T x$ over P. Also note that the second term is non-zero since $z \ne 0$. Hence

$$(14) \qquad d^T(x_0-x_1) \ge \frac{1}{\alpha} \ge \frac{1}{T^2}$$

and so x_0 is the unique vertex of P maximizing the objective function $d^T x$.

Now use the hypothesized polynomial algorithm to find a vector $y \in \mathbb{R}^n$ such that

$$(15) \qquad d(y,P) \le \varepsilon = \tfrac{1}{4}\|d\|^{-1} T^{-6}$$

and $d^T y \ge d^T x_0 - \varepsilon$. We claim that

$$(16) \qquad \|y-x_0\| \le \frac{1}{2T^2}.$$

For let y_0 be the point of P next to y. Represent y_0 as a convex combination of n+1 vertices of P, one of which is x_0:

$$(17) \qquad y_0 = \sum_{i=0}^{n} \lambda_i x_i, \ \lambda_i \geq 0, \ \sum_{i=0}^{n} \lambda_i = 1.$$

Then by (14)

$$(18) \qquad d^T y = d^T(y-y_0) + d^T y_0 \leq \varepsilon \| d \| + \sum_{i=0}^{n} \lambda_i d^T x_i \leq \varepsilon \| d \| + d^T x_0 - \frac{1-\lambda_0}{T^3} .$$

Hence

$$(19) \qquad \frac{1-\lambda_0}{T^3} \leq \varepsilon(\| d \| + 1) \leq 2\varepsilon \| d \|$$

and

$$(20) \qquad \| y-x_0 \| \leq \| y-y_0 \| + \| y_0-x_0 \| \leq \varepsilon + (1-\lambda_0) \| \sum_{i=1}^{n} \frac{\lambda_i}{1-\lambda_0} x_i - x_0 \| \leq$$

$$\leq \varepsilon + (1-\lambda_0) 2T^2 \leq \varepsilon + 2\varepsilon \| d \| . 2T^4 \leq \frac{1}{2T^3} .$$

Now it is rather clear how to conclude: round each entry of y to the next rational number with denominator less than T; the resulting vector is x_0. The rounding can be done by using the technique of continued fractions. We leave the details to the reader.

The separation algorithm can be obtained by applying the previous algorithm to P^* (assuming that $a_0 = 0$, possibly after translation). □

If the strong separation problem concludes that $y \in P$ then it is nice to have a "proof" of that, i.e., a representation of y as a convex combination of vertices of P. This problem can also be solved.

(3.9) THEOREM. *Let K be a solvable class of rational polytopes. Then there exists an algorithm which, given $(P; n, a_0, T) \in K$ and a rational vector $y \in P$, yields vertices $x_0, x_1, \ldots, x_n$ of P and coefficients $\lambda_0, \lambda_1, \ldots, \lambda_n \geq 0$ such that $\lambda_0+\lambda_1+\ldots+\lambda_n = 1$ and $\lambda_0 x_0+\lambda_1 x_1+\ldots+\lambda_n x_n = y$, in time polynomial in n, $\log T$ and $\log S$, where S is the maximum absolute value of numerators and denominators of components of y.*

PROOF. We construct a sequence $x_0, x_1, \ldots, x_n$ of vertices, $y_0, y_1, \ldots, y_n$ of points and $F_1, F_2, \ldots, F_n$ of facets of P as follows. Let x_0 be any vertex of P, and let $y_0 = y$. Assume that x_i, y_i and F_i are defined for $i \leq j$. Let y_{j+1} be the last point of P on the semi-line from x_j through y_j, let

(21) $$y'_{j+1} = y_{j+1} + \varepsilon(y_j - x_j)$$

where $0 < \varepsilon < (nT)^{-3n^2}$. Let F_{j+1} be a facet separating y'_{j+1} from P, and let x_{j+1} be a vertex of $F_1 \cap \ldots \cap F_{j+1}$. It is straightforward to prove by induction that $x_i, y_i \in F_j$ for $j \geq i$, $y \in \mathrm{conv}(x_0,\ldots,x_i,y_i)$, and $\dim(F_1 \cap \ldots \cap F_j) = n-j$. Hence $x_n = y_n$ and so y is contained in the convex hull of $x_0,\ldots,x_n$.

The procedure described above is easy to follow with computation. The vertex of $F_1 \cap \ldots \cap F_j$ can be obtained as follows. Let $a_i^T x \leq b_i$ be the inequality corresponding to facet F_i; then maximize the objective function $(\sum_{i=1}^{j} a_i)^T x$. We leave the details to the reader. □

The "dual" form of this theorem will also play an important role in the sequel. It shows that if we consider optimization on P as a linear program, an optimal dual basic solution can be found in polynomial time, if the class is solvable.

(3.10) **THEOREM**. *Let K be a solvable class of rational polytopes. Then there exists a polynomial-bounded algorithm which, given $(P;n,a_0,T) \in K$, $c \in \mathbb{Z}^n$, provides facets $a_i^T x \leq b_i$ $(i = 1,\ldots,n)$ and rationals $\lambda_i \geq 0$ $(i = 1,\ldots,n)$ such that $\sum_{i=1}^{n} \lambda_i a_i = c$ and $\sum_{i=1}^{n} \lambda_i b_i = \max\{c^T x \mid x \in P\}$.*

The *proof* is easy by considering K^*.

4. MATROID INTERSECTION, BRANCHINGS AND MATCHINGS

We now apply the methods described in the previous chapters to a number of combinatorial problems. As said in the introduction our main aim is to show the *existence* of polynomial algorithms for certain combinatorial problems, and these algorithms are not meant as substitutes for the algorithms developed for these problems before (see Lawler [1976] for a survey). However, in the next chapters we shall show the existence of polynomial algorithms also for certain problems which were not yet solved in this sense. The algorithms found there, though polynomial, in general do not seem to have the highest possible rate of efficiency, and the challenge remains to find better algorithms.

First we apply the ellipsoid method to matroid intersection (cf. Edmonds [1970,1979], Lawler [1970]). Note that given a matroid (V,r), the corresponding *matroid polytope* is the convex hull of the characteristic vectors of independent sets. The idea is very simple: given an integral "weight" function w on V, the trivial "greedy algorithm" finds an independent set V' maximizing $\sum_{v\in V'} w(v)$. That is, it finds a vertex x of the corresponding matroid polytope maximizing the objective function $w^T x$, in time bounded by a polynomial in $|V|$ and $\log\|w\|$. So the class of matroid polytopes is solvable. Therefore, by Corollary (3.3) also intersections of matroid polytopes are solvable. Since the intersection of two matroid polytopes has integer vertices again, this provides us with a polynomial algorithm for matroid intersection. (In fact, we obtain a polynomial algorithm for common "fractional" independent sets for any number of matroids.) Obviously, we may replace "matroid" by "polymatroid". In Chapter 5 we shall extend this algorithm to a more general class of polytopes, and we shall show there how to obtain optimal integral *dual* solutions.

In this application, and in the following examples we leave it to the reader to check that without loss of generality we may restrict the classes of polytopes to full-dimensional polytopes, and to find a vector a_0 and a number T such that (i) each numerator and denominator occurring in the components of the vertices of the polytope, and in those of a_0, do not exceed T in absolute value, (ii) a_0 is an internal point of the polytope, and (iii) $\log T$ is bounded by a polynomial in the size of the original combinatorial problem (in most cases we have $T = 1$).

Also the second application is illustrative for the use of the method. It shows the existence of a polynomial algorithm for finding optimum branchings in a directed graph (cf. Chu and Liu [1965], Edmonds [1967]). Let $D = (V,A)$ be a digraph, and let r be some fixed vertex of D, called the *root*. A *branching* is a set A' of arrows of D making up a rooted directed spanning tree, with root r. A *rooted cut* is a set A' of arrows with $A' = \delta^-(V')$ for some non-empty set V' of vertices not containing r, where $\delta^-(V')$ denotes the set of arrows entering V'. It follows from Edmonds' branching theorem [1973] that the convex hull of the (characteristic vectors of) the sets of arrows containing a branching as a subset (i.e., the sets intersecting each rooted cut), is a polytope P in $\mathbb{R}^A$ defined by the following linear inequalities:

$$(1)\quad \begin{array}{lll} \text{(i)} & 0 \le x(a) \le 1 & (a \in A), \\ \text{(ii)} & \sum_{a\in A'} x(a) \ge 1 & (A' \text{ rooted cut}). \end{array}$$

So there exists an algorithm which, given a digraph $D = (V,A)$, a root r, and a nonnegative integral weight function w defined on A, determines a branching of minimum weight, in time polynomially bounded by $|V|$ and $\log\|w\|$, if and only if the strong optimization problem is solvable for the class of polytopes P arising in this way. By Theorem (3.1) and (3.8) it is enough to show that the strong separation problem is solvable. Indeed, if $x \in \mathbb{R}^A$ one easily checks condition (i) above and one finds a separating hyperplane in case of violation. To check condition (ii), we can find a rooted cut A' minimizing $\sum_{a \in A'} x(a)$ in time polynomially bounded by $|V|$ and $\log T$ (where T is the maximum of the numerators and denominators occurring in x), namely by applying Ford-Fulkerson's max flow-min cut algorithm to the corresponding network with capacity function x, source r and sink s, for each $s \neq r$. If the minimum is not less than 1 we conclude $x \in P$, and otherwise A' determines a separating hyperplane. (Again, see Chapter 5 for a more general approach.)

In fact this branching algorithm is one instance of a more general procedure. Let $\mathcal{E}$ be a *clutter*, i.e., a finite collection of finite sets no two of which are contained in each other. The *blocker* $B(\mathcal{E})$ of $\mathcal{E}$ is the collection of all minimal sets intersecting every set in $\mathcal{E}$ (minimal with respect to inclusion). E.g., if $\mathcal{E}$ is the collection of branchings in a digraph, then $B(\mathcal{E})$ is the collection of minimal rooted cuts. One easily checks that $B(B(\mathcal{E})) = \mathcal{E}$ for every clutter $\mathcal{E}$. Sometimes an even stronger duality relation may hold. Let $V = \bigcup\mathcal{E}$, and let P be the convex hull of the characteristic vectors (in $\mathbb{R}^V$) of all subsets of V containing some set in $\mathcal{E}$. Clearly, each vector x in P satisfies:

$$(2)\qquad \begin{array}{lll} \text{(i)} & 0 \le x(v) \le 1 & (v \in V), \\ \text{(ii)} & \sum_{v \in V'} x(v) \ge 1 & (V' \in B(\mathcal{E})), \end{array}$$

as these inequalities hold for characteristic vectors of sets in $\mathcal{E}$. In case P is completely determined by these linear inequalities, $\mathcal{E}$ (or the hypergraph $(V,\mathcal{E})$) is said to have the *$\mathbb{Q}_+$-max flow-min cut-property* or the *$\mathbb{Q}_+$-MFMC-property* (cf. Seymour [1977]). Thus the clutter of all branchings in a digraph has the $\mathbb{Q}_+$-MFMC-property, as we saw above. Fulkerson [1970] showed the interesting fact that a clutter $\mathcal{E}$ has the $\mathbb{Q}_+$-MFMC-property if and only if its blocker $B(\mathcal{E})$ has the $\mathbb{Q}_+$-MFMC-property.

Now one easily extends the derivation of a polynomial algorithm for branchings from such an algorithm for rooted cuts as described above, to the following theorem.

(4.1) THEOREM. *Let $\mathcal{C}$ be a class of clutters with the $\mathbb{Q}_+$-MFMC-property, such that there exists an algorithm which finds, given $\mathcal{E} \in \mathcal{C}$ and $w \in \mathbb{Z}_+^V$ (where $V = \bigcup\mathcal{E}$), a set V' in $\mathcal{E}$ minimizing $\sum_{v \in V'} w(v)$, in time bounded by a polynomial in $|V|$ and $\log\|w\|$. Then the same is true for the class of blockers of clutters in $\mathcal{C}$.*

One should be caraful with how the clutter $\mathcal{E}$ is given. Perhaps formally the most proper way to formulate the theorem is as follows: there exists an algorithm A and a polynomial f(x) such that A given a "minimization algorithm" for some clutter $\mathcal{E}$ with the $\mathbb{Q}_+$-MFMC-property, with "time bound" $g(\log\|w\|)$, and given some vector $u \in \mathbb{Z}_+^V$ $(V = \cup\mathcal{E})$, A finds a set V' in $B(\mathcal{E})$ minimizing $\sum_{v\in V'} u(v)$ in time bounded by $f(|V|.\log\|u\|.g(|V|.\log\|u\|))$.

Among the other instances of Theorem (4.1) are the following. Let D = (V,A) be a digraph. A *directed cut* is a set A' of arrows of D such that A' = $\delta^-(V')$ for some nonempty proper subset V' of V with $\delta^+(V') = \emptyset$ (as usual, $\delta^-(V')$ and $\delta^+(V')$ denote the sets of arrows entering and leaving V', respectively). A *covering* is a set of arrows intersecting each directed cut, i.e., a set of arrows whose contraction makes the digraph strongly connected. Let $\mathcal{E}$ be the clutter of all minimal directed cuts. It follows from the Lucchesi-Younger theorem [1978] that $\mathcal{E}$ has the $\mathbb{Q}_+$-MFMC-property, and an easy adaptation of Ford-Fulkerson's max flow-min cut algorithm yields a polynomial algorithm for finding minimum weighted directed cuts, given some nonnegative weight function on the arrows (to this end we could add for each arrow also the reversed arrow with infinite capacity). Hence, by Theorem (4.1) there exists a polynomial algorithm for finding minimum weighted coverings in a digraph (such algorithms were found earlier by Lucchesi [1976], Karzanov [1979] and Frank [1979]).

In fact we do not need to call upon Ford-Fulkerson for finding minimum weighted cuts; such an algorithm can be derived also from Theorem (4.1). Indeed, let D = (V,A) be a digraph and let r and s be two specified vertices. Let $\mathcal{E}$ be the clutter of all directed r-s-paths (considered as sets of arrows). So the blocker $B(\mathcal{E})$ of $\mathcal{E}$ consists of all minimal r-s-cuts. It follows from the max flow-min cut theorem that $\mathcal{E}$ has the $\mathbb{Q}_+$-MFMC-property. There exists an (easy) polynomial algorithm for finding shortest paths (Dijkstra [1959]), hence there exists a polynomial algorithm for finding minimum weighted cuts.

Theorem (4.1) also applies to T-cuts and T-joins. Let G = (V,E) be an undirected graph, and let T be a set of vertices of G of even size. A set E' of edges of G is called a *T-cut* if there exists a set V' of vertices with $|V'\cap T|$ odd such that E' is the set of edges of G intersecting V' in exactly one vertex. A *T-join* is a set E' of edges with the property that T coincides with the set of vertices of odd valency in the graph (V,E'). One easily checks that the clutter $\mathcal{E}$ of all minimal T-cuts has as blocker the clutter of all minimal T-joins, and Edmonds and Johnson [1970] showed that $\mathcal{E}$ has the $\mathbb{Q}_+$-MFMC-property. Padberg and Rao [1979] adapted the Ford-Fulkerson minimum cut algorithm to obtain a polynomial algorithm to find minimum weighted T-cuts, given a nonnegative weight function on the edges (cf. also Chapter 5). Hence there exists a polynomial algorithm for finding minimum weighted T-joins, which was demonstrated earlier by Edmonds and Johnson [1970]. As special cases we may derive a polynomial algorithm for the Chinese postman problem (take T to be the set of vertices of odd valency in G), and a polynomial algorithm for finding minimum weighted perfect matchings (take $T = V$, and add a large constant to all weights; it is easy to derive conversely from a polynomial algorithm for minimum weighted perfect matchings, a polynomial algorithm for minimum weighted T-joins). In the latter case we may even take weights to be negative (which could be repaired by adding a large constant to all weights), and hence we have also polynomial algorithms for maximum weighted matchings (cf. Edmonds [1965]).

From Theorem (3.10) we know that if $\mathcal{C}$ is a class of clutters with the properties as described in Theorem (4.1), then there exists an algorithm to find optimal dual solutions, that is, given $\mathcal{E} \in \mathcal{C}$ and $w \in \mathbb{Z}_+^V$ (where $V = \cup\mathcal{E}$), sets $E_1,\dots,E_t$ in $B(\mathcal{E})$, with $t \le |V|$, and nonnegative numbers $\lambda_1,\dots,\lambda_t$ such that

$$(3) \qquad \begin{array}{ll} \text{(i)} & \lambda_1\chi_{E_1} + \dots + \lambda_t\chi_{E_t} \le w, \\ \text{(ii)} & \lambda_1 + \dots + \lambda_t = \min_{V' \in \mathcal{E}} \sum_{v \in V'} w(v) \end{array}$$

(where χ_E denotes the characteristic vector in $\mathbb{R}^V$ of E), in time polynomially bounded by $|V|$ and $\log\|w\|$. So in the special cases discussed above this provides us with polynomial algorithm to find optimal fractional packings of branchings, rooted cuts, coverings, directed cuts, r-s-cuts, r-s-paths (i.e., optimum fractional r-s-flow), T-joins, T-cuts. Similarly, polynomial algorithms for finding optimum fractional two-commodity flow, and for fractional packings of two-commodity cuts may be derived (cf. Hu [1963,1973]). Moreover, a recent theorem of Okamura and Seymour [1979] implies the existence of a polynomial algorithm for finding optimum fractional multicommodity flows in planar undirected graphs, provided that all sources and all sinks are on the boundary of the infinite face.

It is not necessarily true that if $\mathcal{E}$ has the $\mathbb{Q}_+$-MFMC-property, we can take the $\lambda_1,\dots,\lambda_t$ in the dual solution to be integers. If this is the case for each $w \in \mathbb{Z}_+^V$ then $\mathcal{E}$ is said to have the $\mathbb{Z}_+$*-MFMC-property*; and if for each such w we can take the $\lambda_1,\dots,\lambda_t$ to be half-integers, $\mathcal{E}$ has the $\frac{1}{2}\mathbb{Z}_+$*-MFMC-property*. E.g., the clutters of branchings, rooted cuts, coverings, r-s-cuts, r-s-paths all have the $\mathbb{Z}_+$-MFMC-property (proved by Fulkerson [1974], Edmonds [1973], Lucchesi and Younger [1978], Ford and Fulkerson [1956], and Fulkerson [1968], respectively), and the clutters of T-joins, two-commodity cuts, two-commodity paths, and multicommodity cuts in planar graphs (with commodities on the boundary), have the $\frac{1}{2}\mathbb{Z}_+$-MFMC-property (proved by Edmonds and Johnson [1970], Hu [1963], Seymour [1978], and Okamura and Seymour [1979], respectively). Edmonds and Giles [1977] posed the problem whether the clutter of directed cuts has the $\mathbb{Z}_+$-MFMC-property.

We were not able to derive from the ellipsoid method in general a polynomial algorithm for optimum (half-)integer dual solutions, if such solutions exist. However, in the case of optimum packings of (rooted, directed, r-s-, T-) cuts we can find by Theorem (3.10) an optimum fractional solution in which the number of cuts with nonzero coefficient is at most $|V|$. Hence, by well-known techniques (cf. Edmonds and Giles [1977], Frank [1979], Lovász [1975]) we can make these cuts *laminar* (i.e., non-crossing) in polynomial time, and we can find (half-)integer coefficients for the new collection of cuts, again in polynomial time, thus yielding an optimum (half-)integer packing of cuts.

We do not know whether the class $\mathcal{C}$ of *all* clutters with the $\mathbb{Q}_+$-MFMC-property is polynomially solvable in the sense of Theorem (4.1) (in which case Theorem (4.1) would become trivial). In Chapter 6 we shall see this indeed is the case for its anti-blocking analogue.

In this chapter, as well as in the next chapters we see that the existence of polynomial algorithms can be derived from the ellipsoid method for many problems for which such algorithms have been designed before. However, we were not able to derive such an algorithm for the following two problems, for which (complicated) polynomial algorithms are known: the problem of finding a maximum independent set of vertices in a $K_{1,3}$-free graph (Minty [1977]), and that of finding a maximum collection of independent lines in a projective space (Lovász [1978]). A main obstacle to derive such algorithm from the ellipsoid method is that so far no characterizations in terms of facets of the corresponding convex-hull polytopes have been found. Such characterizations, previously considered as more theoretical of nature, might now be useful to derive polynomial algorithms even for the *weighted* versions of these problems.

5. SUBMODULAR FUNCTIONS AND DIRECTED GRAPHS

In this chapter we show the existence of a polynomial algorithm finding the minimum value of a submodular set function, and we derive polynomial algorithms for the optimization problems introduced by Edmonds and Giles [1977] and by Frank [1979].

Let X be a finite set, let F be a collection of subsets of X closed under union and intersection, and let f be an integer-valued submodular function defined on F, that is, let $f:F\longrightarrow\mathbb{Z}$ be such that

$$(1) \qquad f(X')+f(X'')\geq f(X'\cap X'')+f(X'\cup X'')$$

for $X', X''\in F$. Examples of submodular functions are the rank functions of matroids, and the function f defined on all sets V' of vertices of a capacitated digraph by: f(V') is the sum of the capacities of the arrows leaving V'. We shall give an algorithm to find X' in F minimizing f(X'), in time polynomially bounded by $|X|$ and $\log B$, where B is some (previously known) upper bound for $|f(X')|$ $(X'\in F)$. So as special cases we can decide in polynomial time, given a matroid (X,r) and a weight function w on X, whether w is in the corresponding matroid polytope (i.e., whether $\sum_{x\in X'}w(x)\leq r(X')$ for each subset X' of X), and we can derive a polynomial algorithm for finding minimum capacitated cuts in networks.

We need to make some requirements on the way F and f are given. First we should know an upper bound B for $|f(X')|$ $(X'\in F)$. Secondly, we must know in advance the sets $\cap F$ and $\cup F$, as well as for which pairs x_1,x_2 in X there exists $X'\in F$ with $x_1\in X'$ and $x_2\notin X'$. This makes it possible to decide whether a given subset X' of X is in F, and it allows us to assume without loss of generality that $\cap F=\emptyset$ and $\cup F=X$. Finally, given X' in F we must be able to find f(X'). It is enough to know that f(X') can be calculated in time polynomial in $|X|$ and $\log B$, or that some oracle gives the answer. Most of the special-case submodular functions fulfil these requirements.

We shall reduce the minimization problem for submodular functions to the strong separation problem for polymatroid polytopes. Since the class of polymatroid polytopes is solvable (as optimization can be don by the greedy algorithm - see Edmonds [1970]), this will solve the problem.

Since we know an upper bound B for $|f(X')|$ we can find the minimum value of f by applying binary search. So it suffices to have a polynomial algorithm finding an X' in F with $f(X')<K$, or deciding that no such X' exists, for any

given K. Since adding a constant to the values of f does not violate submodularity we can take $K = 0$. Now if $f(\emptyset) < 0$ we can take $X' = \emptyset$. Hence we may assume that $f(\emptyset) = 0$.

Let g be the function defined on $\mathcal{F}$ by

$$(2) \qquad g(X') = f(X') + 2B.|X'|,$$

for $X' \in \mathcal{F}$. So g is nonnegative, integral, monotone and submodular. Moreover, $f(X') < 0$ if and only if $g(X') < 2B.|X'|$.. Next define for each subset X' of X the set

$$(3) \qquad \overline{X'} = \cap\{X'' \in \mathcal{F} \mid X' \subseteq X''\}.$$

(Note that $\overline{X'}$ can be determined in polynomial time.) Let $h(X') = g(\overline{X'})$ for each subset X' of X. One easily checks that h again is nonnegative, integral, monotone and submodular. Moreover, the problem of the existence of an X' in $\mathcal{F}$ with $g(X') < 2B.|X'|$ is equivalent to that of the existence of a subset X' of X with $h(X') < 2B.|X'|$. But the latter problem is just a special case of the strong separation problem for the polymatroid polytope corresponding to h:

$$(4) \qquad \{v \in \mathbb{R}^X_+ \mid \textstyle\sum_{x \in X'} v(x) \le h(X') \text{ for all } X' \subseteq X\}$$

(by Theorem (3.8) the separation algorithm yields facets as separating hyperplanes, i.e., subsets X' of X violating the inequality). As the optimization problem is solvable for the class of these polytopes we are finished.

We apply the algorithm for finding the minimum value of a submodular function to theorems of Edmonds and Giles and of Frank.

Let $D = (V,A)$ be a digraph, and let $\mathcal{F}$ be a collection of subsets of V such that if $V',V'' \in \mathcal{F}$ and $V' \cap V'' \ne \emptyset$ and $V' \cup V'' \ne V$ then $V' \cap V'' \in \mathcal{F}$ and $V' \cup V'' \in \mathcal{F}$. Let f be an integer-valued function defined on $\mathcal{F}$ such that for all $V',V'' \in \mathcal{F}$ with $V' \cap V'' \ne \emptyset$ and $V' \cup V'' \ne V$ we have

$$(5) \qquad f(V') + f(V'') \ge f(V' \cap V'') + f(V' \cup V'').$$

Denote by $\delta^+(V')$ and $\delta^-(V')$ the sets of arrows leaving (entering, respecively) the set V' of vertices. Let vectors $b,c,d \in \mathbb{Z}^A$ be given, and consider the linear programming maximization problem

(6) maximize $\sum_{a\in A} c(a).x(a)$,

where $x\in\mathbb{R}^A$ such that

(7) (i) $d(a)\le x(a)\le b(a)$ $(a\in A)$,

(ii) $\sum_{a\in\delta^+(V')} x(a) - \sum_{a\in\delta^-(V')} x(a) \le f(V')$ $(V'\in \mathcal{F})$.

Edmonds and Giles showed that this problem has an integer optimum solution; this is equivalent to the fact that the polytope defined by the linear inequalities (7) has integral vertices. Edmonds and Giles also showed that the dual minimization problem can be solved with integral coefficients.

As special cases of Edmonds and Giles' result one has Ford and Fulkerson's max flow-min cut-theorem, the Lucchesi-Younger theorem on packing directed cuts and minimum coverings, Edmonds' (poly-)matroid intersection theorem, and theorems of Frank [1979] on orientations of undirected graphs. Moreover, one may derive the theorem due to Frank that if f is an integral submodular function defined on a collection $\mathcal{F}$ and g is an integral supermodular function on $\mathcal{F}$ (i.e., -g is submodular) such that $g(X')\le f(X')$ for all X' in $\mathcal{F}$, then there exists an integral *modular* function h on $\mathcal{F}$ (i.e., both sub- and supermodular) such that $g(X')\le h(X')\le f(X')$ for all X' in $\mathcal{F}$.

We shall give an algorithm which solves the maximization problem (6) in time polynomially bounded by $|V|$ and $\log B$, where B is some (previously known) upper bound on $|f(X')|$ $(X'\in\mathcal{F})$, $\|b\|$, $\|c\|$ and $\|d\|$. We must know in advance for each pair of vertices v_1, v_2 of D whether $v_1\in V'$ and $v_2\notin V'$ for some $V'\in\mathcal{F}$ (this makes it possible to decide whether $V'\in\mathcal{F}$). Moreover we must have a subroutine calculating $f(V')$ if $V'\in\mathcal{F}$, in time polynomially bounded by $|V|$ and $\log B$.

First of all, we may suppose that $d(a)<b(a)$ for each arrow a, since if $d(a)>b(a)$ the polytope (7) is empty, and if $d(a)=b(a)$ we can remove the arrow a from the digraph and replace $f(V')$ by $f(V')\pm d(a)$ if $a\in\delta^{\mp}(V')$. We may even assume that the polytope (7) is full-dimensional and that we know an interior point x whose components have numerators and denominators not larger than a polynomial in $|V|$ and B. Otherwise we can extend the digraph D with one new vertex v_0 and with new arrows (v_0,v) for each "old" vertex v of D. Define $d(a) = -4B$, $b(a)=0$ and $c(a)=2nB^2$ for the new arrows a. One easily checks that the corresponding new polytope is full-dimensional, and one easily finds an x as required. Moreover, the solutions of the original optimization problem correspond exactly to those solutions x of the new problem with $x(a)=0$ for each new arrow a.

Assuming the polytope (7) to be full-dimensional, by Theorem (3.8) it is enough to show that the strong separation problem is solvable. Let $x \in \mathbb{R}^A$. One easily checks in polynomial time whether condition (i) is fulfilled. In case of violation we find a separating hyperplane. To check condition (ii) it suffices to find a set V' in $\mathcal{F}$ minimizing

$$(8) \qquad g(V') := f(V') - \textstyle\sum_{a \in \delta^+(V')} x(a) + \sum_{a \in \delta^-(V')} x(a)$$

in time polynomial in log B and log T, where T is the maximum of the numerators and denominators occurring in x. Note that g is submodular, hence we can appeal to the algorithm finding the minimum value of a submodular function. To this end we have to multiply the values by a factor to make the function integral (this factor is bounded above by $T^{|V|^3}$), and we have to apply the algorithm for each v_1, v_2 in V with $v_r \neq v_2$, to the function restricted to $\{V' \in \mathcal{F} \mid v_1 \in V', v_2 \notin V'\}$, since this collection is closed under union and intersection. Note that these requisites do not affect the polynomial boundedness of the required time.

So we proved that the class of "Edmonds-Giles" polytopes is solvable. Hence, by Theorem (3.10) we can find an optimum solution for the dual linear programming problem. In general, this solution will be fractional, but one can make this solution integral by making the collection of sets in $\mathcal{F}$ with non-zero dual coefficient laminar, by the well-known techniques (see Edmonds and Giles [1977]), in polynomial time. Now the (possibly fractional) coefficients can be replaced by integer coefficients, and these coefficients can be found by solving a linear program of polynomial size.

We leave it to the reader to derive by similar methods a polynomial algorithm for finding the solution to the following optimization problem, designed by Frank [1979]. Let D = (V,A) be a digraph, and let $\mathcal{F}$ be a collection of subsets of V such that if $V', V'' \in \mathcal{F}$ and $V' \cap V'' \neq \emptyset$ then $V' \cap V'' \in \mathcal{F}$ and $V' \cup V'' \in \mathcal{F}$. Let f be a nonnegative integral function defined on $\mathcal{F}$ such that

$$(9) \qquad f(V') + f(V'') \leq f(V' \cap V'') + f(V' \cup V'')$$

if $V', V'' \in \mathcal{F}$ and $V' \cap V'' \neq \emptyset$. Let $b, c, d \in \mathbb{Z}_+^A$. Consider the linear programming problem

$$(10) \qquad \text{minimize } \textstyle\sum_{a \in A} c(a).x(a),$$

where $x \in \mathbb{R}^A$ such that

$$(11) \qquad \text{(i)}\ d(a) \le x(a) \le b(a) \qquad (a \in A),$$
$$\text{(ii)}\ \sum_{a\in\delta^{+}(V')} x(a) \ge f(V') \qquad (V' \in \mathcal{F}).$$

Frank showed that this problem, and its dual, has an integer solution. As special cases one may derive again Ford and Fulkerson's max flow-min cut theorem and Edmonds polymatroid intersection theorem, and also Fulkerson's theorem on minimum weighted branchings [1974].

We finally remark that the algorithm for finding a set V' in $\mathcal{F}$ minimizing $f(V')$, where f is a submodular function defined on $\mathcal{F}$, can be modified to a polynomial algorithm for finding a set V' in $\mathcal{F}$ of odd size minimizing $f(V')$. This extends Padberg and Rao's algorithm [1979] to find minimum odd cuts. More generally, let $\mathcal{G} \subseteq \mathcal{F}$ be such that if $V' \in \mathcal{G}$ and $V'' \in \mathcal{F}\backslash\mathcal{G}$ then $V'\cap V'' \in \mathcal{G}$ or $V'\cup V'' \in \mathcal{G}$. (E.g., $\mathcal{G}$ is the collection of sets in $\mathcal{F}$ intersecting V_0 in a number of elements not divisible by k, for some fixed subset V_0 of V and some natural number k.) Then there exists a polynomial algorithm to find V' in $\mathcal{G}$ minimizing $f(V')$ (by this we mean: $f(V') = \min\{f(V'') \mid V'' \in \mathcal{G}\}$). This algorithm needs, besides the prerequisites for $\mathcal{F}$ and f as above, a polynomial subroutine deciding whether a given set V' is in $\mathcal{G}$. Without loss of generality we may assume that $\emptyset \notin \mathcal{G}$ and $V \notin \mathcal{G}$.

The algorithm is defined by induction on $|V|$. Suppose the algorithm has been defined for all such structures with smaller $|V|$. Find a set V' in $\mathcal{F}$ such that $\emptyset \neq V' \neq V$ which minimizes $f(V')$. This can be done by applying the polynomial algorithm described above to the function f restricted to the collection $\{V'\in\mathcal{F} \mid v_1 \in V',\ v_2 \notin V'\}$, for all v_1, v_2 in V. If $V' \in \mathcal{G}$ we are finished. If $V' \notin \mathcal{G}$ there will be a set V'' in $\mathcal{G}$ minimizing $f(V'')$ such that $V'' \subseteq V'$ or $V' \subseteq V''$. Indeed, if $V'' \in \mathcal{G}$ minimizes $f(V'')$ then either $V'\cap V'' \in \mathcal{G}$ or $V'\cup V'' \in \mathcal{G}$; in the former case we have

$$(12) \qquad \begin{aligned} &f(V'\cap V'') + f(V'\cup V'') \le f(V') + f(V''),\\ &f(V'\cap V'') \ge f(V''),\\ &f(V'\cup V'') \ge f(V'),. \end{aligned}$$

as V' and V'' minimize $f(V')$ and $f(V'')$ for $V' \in \mathcal{F}$ and $V'' \in \mathcal{G}$, respectively. Hence $f(V'\cap V'') = f(V'')$. If $V'\cup V'' \in \mathcal{G}$ we can exchange $\cup$ and $\cap$ in this reasoning. Now there exists an algorithm finding $V'' \in \mathcal{G}$ with $V'' \subseteq V'$ minimizing $f(V'')$, and an algorithm finding $V'' \in \mathcal{G}$ with $V'' \supseteq V'$ minimizing $f(V'')$, for these algorithms follow straightforwardly from the previously defined algorithms for sets of size $|V'|$ and $|V\backslash V'|$. We leave it to the reader to check that this gives us a polynomial algorithm.

6. INDEPENDENT SETS IN PERFECT GRAPHS

In the previous chapters we have applied the ellipsoid method to classes of polytopes. We now apply the method to a class of non-polytopal convex sets, in order to obtain a polynomial algorithm finding maximum (weighted) independent sets and minimum colourings for perfect graphs (cf. Lovász [1972]).

Let $G = (V,E)$ be an undirected graph, and let $\alpha(G)$ denote the *independence number* of G, i.e., the maximum number of paiwise non-adjacent vertices. Let $\alpha^*(G)$ denote the *fractional independence number* of G, i.e., the maximum value of $\sum_{v\in V} c(v)$ where the $c(v)$ are nonnegative real numbers such that $\sum_{v\in C} c(v) \leq 1$ for each clique C of G. So $\alpha(G) \leq \alpha^*(G)$, and furthermore G is perfect if and only if $\alpha(G') = \alpha^*(G')$ for each induced subgraph G' of G. Since $\alpha^*(G)$ is the optimum of a linear programming problem, we could try to calculate $\alpha^*(G)$ by means of the ellipsoid method; but the size of this problem is not polynomially bounded as there can exist too many cliques C.

However, the following number $\vartheta(G)$ was introduced in Lovász [1979]. Suppose $V = \{1,\dots,n\}$. Then $\vartheta(G)$ is the maximum value of $\sum_{i,j=1}^{n} b_{ij}$, where $B = (b_{ij})$ belongs to the following convex body of matrices

$$(1)\qquad \{B = (b_{ij}) \mid B \text{ is positive semi-definite with trace 1, and } b_{ij} = 0 \text{ if } i \text{ and } j \text{ are adjacent vertices of } G\ (i \neq j)\}.$$

If B belongs to this class we shall say that B *represents* G. It was shown that $\alpha(G) \leq \vartheta(G) \leq \alpha^*(G)$. (In fact, $\vartheta(G)$ is an upper bound for the *Shannon capacity* of G.) We show that $\vartheta(G)$ can be calculated (approximated) by the ellipsoid method in time bounded by a polynomial in $|V|$. This allows us to find $\vartheta(G)$ for graphs G with $\alpha(G) = \vartheta(G)$, in particular for perfect graphs.

In fact we exhibit an algorithm finding the maximum *weight* $\sum_{v\in A} w(v)$, where A is an independent set in a perfect graph, given some nonnegative integral weight function on V, in time polynomially bounded by $|V|$ and $\log\|w\|$. Obviously, this maximum weight is equal to $\alpha(G_w)$, where the graph G_w arises from G by replacing each vertex v of G by w(v) pairwise non-adjacent new vertices and where two vertices of G_w are adjacent iff their originals in G are adjacent. Note that if G is perfect then also any G_w is perfect (cf. Lovász [1972]). Moreover, $\vartheta(G_w)$ is equal to the maximum value of

$$(2)\qquad \sum_{i,j=1}^{n} \sqrt{w_i w_j}\,.\,b_{ij},$$

where $B = (b_{ij})$ represents G. This can be seen as follows. If $B = (b_{ij})$ represents G then, by replacing each entry b_{ij} by a matrix of size $w_i \times w_j$ with constant entries $(w_iw_j)^{-\frac{1}{2}}b_{ij}$, we obtain a matrix B' representing G_w, with

(3) $$\sum_{i,j=1}^{n} \sqrt{w_iw_j}\cdot b_{ij} = \sum_{i,j=1}^{n} b'_{ij}.$$

Conversely, if B' represents G_w, then, by replacing the $w_i \times w_j$ submatrix induced by the copies of i and j, by the sum of its entries divided by $\sqrt{w_iw_j}$, we obtain a matrix B representing G, satisfying (3) again.

To approximate $\vartheta(G_w)$ up to an error of at most $\varepsilon > 0$, we can replace $\sqrt{w_iw_j}$ of (2) by some rational number ω_{ij} with

(4) $$|\omega_{ij} - \sqrt{w_iw_j}| < \varepsilon/2n^2$$

(taking $\omega_{ij} = \omega_{ji}$), where the denominators of the ω_{ij} are at most $2n^2/\varepsilon$. Then $\vartheta(G_w)$ differs by at most $\frac{1}{2}\varepsilon$ from the maximum value of $\sum_{i,j}\omega_{ij}.b_{ij}$ with $B = (b_{ij})$ representing G. So we need to approximate this last number with accuracy $\frac{1}{2}\varepsilon$, which can be done by the ellipsoid method.

To apply the ellipsoid method we replace the set (1) by a full-dimensional convex body, by forgetting the coordinates below the main diagonal, as well as the coordinates (i,j) for adjacent i and j, and one diagonal coordinate. We end up with a full-dimensional convex body in the $(n + \binom{n}{2} - |E| - 1)$-dimensional space. One easily finds an interior point a_0 in it, and radii r and R such that the convex body contains $S(a_0,r)$ and is contained in $S(a_0,R)$, and such that the logarithms of r and R and of the numerators and denominators occurring in a_0 are bounded (in absolute value) by a polynomial in n (fixed over all graphs G). So we may apply Theorem (3.1). We show that the separation problem is solvable for the class of convex bodies ontained in this way. Let b be some vector in the $(n + \binom{n}{2} - |E| - 1)$-dimensional space. Extend this vector, in the obvious way, to a symmetric $n \times n$-matrix $B = (b_{ij})$ with trace 1 and with $b_{ij} = 0$ if i and j are adjacent vertices of G. Find a principal minor B' of B such that rank(B') = rank(B) and B' is nonsingular (this is easy by Gaussian elimination). Without loss of generality assume that $B' = (b_{ij})_{i=1,j=1}^{k\ \ k}$. The matrix B is positive semi-definite iff

(5) $$\det B_t = \det(b_{ij})_{i=1,j=1}^{t\ \ t} \geq 0,$$

for $t = 0,1,\ldots,k$. Since these determinants can be calculated in polynomial time, thereby we have checked in polynomial time whether B belongs to the convex set (1). If, moreover, we find that B is not positive semidefinite then let t be the smallest index for which $\det B_t < 0$. Let ϕ_i denote $(-1)^i$ times the (i,t)-th minor of B_t $(i = 1,\dots,t)$, and $\phi_i = 0$ if $i > t$. Then

$$(6) \qquad \sum_{i,j=1}^{n} \phi_i \phi_j \beta_{ij} \geq 0$$

for every positive semidefinite matrix (β_{ij}). By definition, and by simple computation,

$$(7) \qquad \sum_{i=1}^{n} \sum_{j=1}^{n} \phi_i \phi_j b_{ij} = \det B_k . \det B_{k-1} \leq 0.$$

So the matrix $(\phi_i \phi_j)_{i,j=1}^{n}$ is a solution of the separation problem.

Therefore, by Theorem (3.1), we can approximate the maximum value of $\sum \omega_{ij} b_{ij}$ for $B = (b_{ij})$ in (1) with accuracy $\frac{1}{2}\varepsilon$ (and hence $\vartheta(G_w)$ with accuracy ε), in time polynomially bounded by $|V|$, $|\log \varepsilon|$ and $\log T$ where T is the maximum among the denominators and numerators occurring in (ω_{ij}). If we know that $\alpha(G_w) = \vartheta(G_w)$ it follows that $\vartheta(G_w)$ is an integer, and we can take $\varepsilon = \frac{1}{2}$. In particular there exists an algorithm which calculates $\alpha(G_w)$ for perfect graphs G in time polynomially bounded by $|V|$ and $\log \| w \|$.

We can find an explicit maximum weighted independent set in a perfect graph as follows. Compare $\alpha(G_w)$ with $\alpha(G'_{w'})$, where G' and w' arise from G and w by removing vertex 1 from G and the corresponding compnent from w. If $\alpha(G'_{w'}) = \alpha(G_w)$ we replace G by G' and w by w'; otherwise we leave G and w unchanged. Next we try to remove vertex 2 similarly, and so on. At the end we are left with a collection of vertices forming a maximum weighted independent set in G.

So given a perfect graph $G = (V,E)$ and a weight function w on V we can find an independent set V' maximizing $\sum_{v \in V'} w(v)$. This implies that the strong optimization problem is solvable for the class of convex hulls of the independent sets in perfect graphs. For perfect graphs $G = (V,E)$ this convex hull is given by the linear inequalities

$$(8) \qquad \begin{array}{lll} \text{(i)} & x(v) \geq 0 & (v \in V), \\ \text{(ii)} & \sum_{v \in C} x(v) \leq 1 & (C \text{ clique}). \end{array}$$

This yields that also the strong separation problem is solvable for this class, but this is not interesting anymore, as it amounts to finding a maximum weighted clique in a perfect graph, i.e., a maximum weighted independent set in the complementary graph, which is perfect again.

However, by Theorem (3.9) we can find an optimal (fractional) dual solution for the corresponding linear programming problem. So, given $w \in \mathbb{Z}_+^V$, we can find cliques $C_1,\dots,C_t$ and positive real numbers $\lambda_1,\dots,\lambda_t$ $(t \le |V|)$ such that $\lambda_1+\dots+\lambda_t = \alpha(G_w)$ and

$$(9) \qquad \sum_{\substack{j \\ i\in C_j}} \lambda_j = w_i$$

for each vertex i, in polynomial time. But for perfect graphs $\alpha(G)$ is equal to the minimum number of cliques needed to cover V (i.e., to the chromatic number of the complementary graph), which means that there exist *integers* $\lambda_1,\dots,\lambda_t$ with the required properties. Indeed we can find such integers as follows.

First, if $w \equiv 1$, each clique C_j with $\lambda_j > 0$ intersects all maximum-sized independent sets. So we can remove clique C_1 from G, thus obtaining a graph G' with $\alpha(G') = \alpha(G) - 1$, and we can repeat the procedure for G'. After $\alpha(G)$ repetitions we have found $\alpha(G)$ cliques covering V.

If w is arbitrary, let λ_j' be the lower integer part of λ_j, and let

$$(10) \qquad w_i' = w_i - \sum_{\substack{j \\ i\in C_j}} \lambda_j'.$$

Since $(\lambda_j - \lambda_j') < 1$ we know by (9) that $w_i' < t \le |V|$. Therefore, $G_{w'}$ has at most $|V|^2$ vertices, and as in the previous paragraph a covering with $\alpha(G_{w'})$ cliques can be found in time polynomially bounded by $|V|^2$. This covering, together with the covering by $C_1,\dots,C_t$ with coefficients $\lambda_1',\dots,\lambda_t'$, yields an optimum integral dual solution as required.

We remark that the algorithm to find $\alpha(G)$ clearly works for all graphs G with $\alpha(G) = \vartheta(G)$ but that our method to find an explicit maximum independent set requires that $\alpha(G') = \vartheta(G')$ for each induced subgraph G' of G; this is the case if and only if G is perfect, as was shown by Lovász [1979].

7. INTRACTABILITY OF VERTEX-PACKING AND FRACTIONAL COLOURING

The ellipsoid method yields a certain "polynomial equivalence" of combinatorial problems, in the sense that there exists a polynomial algorithm for one problem iff such an algorithm exists for some other problem. We can use this principle also in the negative: if some problem is "hard" (e.g., NP-complete) then it follows that also certain other problems are hard.

We apply this to the problem of determining the independence number $\alpha(G)$

of a graph G, which is known to be NP-complete (cf. Garey and Johnson [1979]). More precisely, and more generally: given a graph $G = (V,E)$, a weight function $w:V \longrightarrow \mathbb{Z}_+$ and a number K, the problem of deciding whether there exists an independent set V' of vertices such that $\sum_{v\in V'} w(v) \geq K$ (i.e., whether $\alpha(G_w) \geq K$) is NP-complete. To formulate this in terms of polytopes, let P(G) be the convex hull of the characteristic vectors of independent sets in G. Then the strong optimization problem for the class of polytopes P(G), is NP-complete.

Now consider the anti-blocker A(P(G)) of P(G) (cf. Chapter 3). By Corollary (3.5) the strong optimization problem for the class of polytopes A(P(G)) is solvable iff it is solvable for the class of polytopes P(G). This remains to be true if we restrict G to a subclass of the class of all graphs.

Now the strong optimization problem for A(P(G)) asks for a *maximum weighted fractional clique*, i.e., for a vector x in $\mathbb{R}_+^V$ such that $\sum_{v\in V'} x(v) \leq 1$ for each independent set V', and such that $\sum_{v\in V} w(v).x(v)$ is as large as possible, given some weight function w in $\mathbb{Z}_+^V$. By linear programming duality this maximum is equal to the *weighted fractional chromatic number*, i.e., to the minimum value $\gamma_w^*(G)$ of $\lambda_1+\ldots+\lambda_t$, where $\lambda_1,\ldots,\lambda_t$ are positive numbers for which there exist independent sets $V_1,\ldots,V_t$ such that for every vertex v we have

$$(1) \qquad \sum_{\substack{j \\ v\in V_j}} \lambda_j = w(v)$$

(we can take $t \leq |V|$). Hence, given a class of graphs, there exists a polynomial algorithm determining $\alpha(G_w)$ for each graph G in this class and for each weight function w, iff such an algorithm exists determining the fractional chromatic number for each such G and w. In fact, the ellipsoid method shows that both problems are "Turing reducible" to each other (cf. Garey and Johnson [1979]). This implies that, since the former problem for the class of all graphs is NP-complete, the latter problem is both NP-hard and NP-easy, i.e., NP-equivalent.

In fact the problem of determining the fractional chromatic number belongs to the class NP, as in order to show in polynomial time that $\gamma_w^*(G) \leq K$ we can bound the numerators and denominators of the λ_j by $\|w\|.|V|^{|V|}$. So the fractional colouring problem is not only Turing reducible, but even polynomial reducible to the independence number problem, but we do not know the other way around.

Since the problem of determining $\alpha(G)$ is already NP-complete if we restrict G to planar cubic graphs the problem of determining $\gamma_w^*(G)$ remains to be NP-equivalent if G is restricted similarly. The problem of determining the fractional chromatic number $\gamma_w^*(G)$ and that of determining the *chromatic number*

$\gamma(G)$ seem to be incomparable with respect to hardness. For cubic graphs G, $\gamma(G)$ can be determined easily in polynomial time, but the problem of determining $\gamma_w^*(G)$ is NP-equivalent. In contrast to this, for *line graphs* G of cubic graphs the problem of determining $\gamma(G)$ is NP-complete (Holyer [1979]), whereas $\gamma_w^*(G)$ can be determined in polynomial time (since $\alpha(G_w)$ can be determined in polynomial time by the matching algorithm).

REFERENCES

[1965] CHU Yoeng-jin and LIU Tseng-hung, *On the shortest arborescence of a directed graph*, Scientia Sinica 4 (1965) 1396-1400.

[1959] E.W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math. 1 (1959) 269-271.

[1965] J. EDMONDS, *Maximum matching and a polyhedron with 0,1-vertices*, J. Res. Nat. Bur. Standards Sect. B 69 (1965) 125-130.

[1967] J. EDMONDS, *Optimum branchings*, J. Res. Nat. Bur. Standards Sect. B 71 (1967) 233-240.

[1970] J. EDMONDS, *Submodular functions, matroids, and certain polyhedra*, in: "Combinatorial structures and their applications" (Proc. Intern. Conf. Calgary, Alb., 1969; R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds.), Gordon and Breach, New York, 1970, pp. 69-87.

[1973] J. EDMONDS, *Edge-disjoint branchings*, in: "Combinatorial algorithms" (Courant Comp. Sci. Symp. Monterey, Ca., 1972; R. Rustin, ed.), Acad. Press, New York, 1973, pp. 91-96.

[1979] J. EDMONDS, *Matroid intersection*, Annals of Discrete Math. 4 (1979) 39-49.

[1977] J. EDMONDS and R. GILES, *A min-max relation for submodular functions on graphs*, Annals of Discrete Math. 1 (1977) 185-204.

[1973] J. EDMONDS and E.L. JOHNSON, *Matching, Euler tours and the Chinese postman*, Math. Programming 5 (1973) 88-124.

[1956] L.R. FORD and D.R. FULKERSON, *Maximum flow through a network*, Canad. J. Math. 8 (1956) 399-404.

[1962] L.R. FORD and D.R. FULKERSON, *Flows in networks*, Princeton Univ. Press, Princeton, N.J., 1962.

[1977] A. FRANK, *On the orientations of graphs*, J. Combinatorial Theory (B), to appear.

[1979] A. FRANK, *Kernel systems of directed graphs*, Acta Sci. Math. (Szeged) 41 (1979) 63-76.

[1979] A. FRANK, *How to make a digraph strongly connected?*, Combinatorica, to appear.

[1968] D.R. FULKERSON, *Networks, frames, and blocking systems*, in: "Mathematics of the decision sciences" Part I (G.B. Dantzig and A.F. Veinott, eds.), Amer. Math. Soc., Providence, R.I., 1968, pp. 303-334.

[1970] D.R. FULKERSON, *Blocking polyhedra*, in: "Graph theory and its applications" (Proc. adv. Seminar Madison, Wis., 1969; B. Harris, ed.), Acad. Press, New York, 1970, pp. 93-112.

[1974] D.R. FULKERSON, *Packing rooted directed cuts in a weighted directed graph*, Math. Programming 6 (1974) 1-13.

[1979] P. GÁCS and L. LOVÁSZ, *Khachian's algorithm for linear programming*, Math. Programming Studies, to appear.

[1979] M.R. GAREY and D.S. JOHNSON, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[1960] A.J. HOFFMAN, *Some recent applications of the theory of linear inequalities to extremal combinatorial analysis*, in: "Combinatorial analysis" (Proc. 10th Symp. on Appl. Math. Columbia Univ., 1958; R.E. Bellman and M. Hall, jr, eds.), Amer. Math. Soc., Providence, R.I., 1960, pp. 113-127.

[1979] I. HOLYER, to appear.

[1963] T.C. HU, *Multicommodity network flows*, Operations Res. 11 (1963) 344-360.

[1973] T.C. HU, *Two-commodity cut-packing problem*, Discrete Math. 4 (1973) 108-109.

[1979] A.V. KARZANOV, *On the minimal number of arcs of a digraph meeting all its directed cutsets*, to appear.

[1979] L.G. KHACHIAN, *A polynomial algorithm in linear programming*, Doklady Akademii Nauk SSSR 244 (1979) 1093-1096 (English translation: Soviet Math. Dokl. 20 (1979) 191-194).

[1970] E.L. LAWLER, *Optimal matroid intersections*, in: "Combinatorial structures and their applications" (Proc. Intern. Conf. Calgary, Alb., 1969; R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds.), Gordon and Breach, New York, 1970, pp. 233-235.

[1976] E.L. LAWLER, *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, New York, 1976.

[1972] L. LOVÁSZ, *Normal hypergraphs and the perfect graph conjecture*, Discrete Math. 2 (1972) 253-267.

[1975] L. LOVÁSZ, *2-Matchings and 2-coverings of hypergraphs*, Acta Math. Acad. Sci. Hungar. 26 (1975) 433-444.

[1978] L. LOVÁSZ, *The matroid parity problem*, Proc. Conf. Algebraic Methods in Graph Theory (Szeged, 1978), to appear.

[1979] L. LOVÁSZ, *On the Shannon capacity of a graph*, IEEE Trans. on Information Theory 25 (1979) 1-7.

[1979] L. LOVÁSZ, to appear.

[1976] C.L. LUCCHESI, *A minimax equality for directed graphs*, Doctoral Thesis, Univ. Waterloo, Waterloo, Ont., 1976.

[1978] C.L. LUCCHESI and D.H. YOUNGER, *A minimax relation for directed graphs*, J. London Math. Soc. (2) 17 (1978) 369-374.

[1977] G.J. MINTY, *On maximal independent sets of vertices in claw-free graphs*, J. Combinatorial Theory (B), to appear.

[1954] T.S. MOTZKIN and I.J. SCHOENBERG, *The relaxation method for linear inequalities*, Canad. J. Math. 6 (1954) 393-404.

[1979] H. OKAMURA and P.D. SEYMOUR, *Multicommodity flows in planar graphs*, J. Combinatorial Theory (B), to appear.

[1979] M.W. PADBERG and M.R. RAO, *Minimum cut-sets and b-matchings*, to appear.

[1977] P.D. SEYMOUR, *The matroids with the max-flow min-cut property*, J. Combinatorial Theory (B) 23 (1977) 189-222.

[1978] P.D. SEYMOUR, *A two-commodity cut theorem*, Discrete Math. 23 (1978) 177-181.

[1970] N.Z. SHOR, *Convergence rate of the gradient descent method with dilatation of the space*, Kibernetika 2 (1970) 80-85 (English translation: Cybernetics 6 (1970) 102-108).

ELLIPSOIDAL ALGORITHMS FOR LINEAR PROGRAMMING

Stanisław Walukiewicz

Systems Research Institute[*]
Polish Academy of Sciences
Warsaw

A description of ellipsoidal algorithms for linear programming is given. We establish a new estimation for the initial ellipsoid and study the rate of convergence of such algorithms. As a result we obtain the ellipsoidal algorithm with the best rate of convergence and show that even in the worst case its performance is bounded by a polynomial of the number of variables and the length of the input. Detailed proofs, geometrical interpretation, and a numerical example are given.

[*] This work has been done during my stay at the Department of Mathematics, Linköping, Sweden. I am most grateful to colleagues from the Department for their comments and suggestions on the subject of this paper.

1. INTRODUCTION AND AN IDEA OF THE ALGORITHM

These lecture notes provide a complete, self-contained description of the ellipsoidal algorithm for linear programming. Detailed proofs, geometrical interpretation and a numerical example are given. We describe the context of these notes more precisely after the presentation of an idea of the algorithm.
For a given (primal) linear programming problem

$$\text{(PL)} \qquad \begin{aligned} v(PL) = \max\ & c^T y \\ \text{subject to } & My \leqslant d, \\ & y \geqslant 0, \end{aligned}$$

where M is $\bar{m} \times \bar{n}$ matrix, d is $\bar{m} \times 1$ matrix or $d \in R^{\bar{m}}$, $c \in R^{\bar{n}}$ and $y \in R^{\bar{n}}$, the dual problem is

$$\text{(DL)} \qquad \begin{aligned} v(DL) = \min\ & d^T u \\ & M^T u \geqslant c \\ & u \geqslant 0 \end{aligned}$$

where $u \in R^{\bar{m}}$. By v(PL) we denote the optimal value of the objective function and by F(PL) we denote the set of all feasible solutions to (PL). We will always assume that all vectors are column vectors and x^T denotes the transposition of a vector x. From duality theory we know that there are only four possibilities:

1) $v(PL) < +\infty$ or $v(DL) > -\infty$, then $c^Ty^* = d^Tu^*$ where y^* is any optimal solution to (PL). Since $c^Ty \leqslant d^Tu$ for any $y \in F(PL)$ and for any $u \in F(DL)$, then we can write this equality as the inequality $c^Ty^* \geqslant d^Tu^*$.

2) $v(PL) = +\infty$, then $F(DL) = \emptyset$.

3) $v(DL) = -\infty$, then $F(PL) = \emptyset$.

4) $F(PL) = \emptyset$ and $F(DL) = \emptyset$.

Therefore a given problem (PL) (or (DL)) is equivalent to the system of $2(\bar{m} + \bar{n}) + 1$ linear inequalities

$$\begin{aligned} My &\leqslant d \\ -y &\leqslant 0 \\ -M^Tu &\leqslant -c \\ -u &\leqslant 0 \\ d^Tu - c^Ty &\leqslant 0 \end{aligned}$$

in $(\bar{m} + \bar{n})$ real variables. For further studies we write the above system in a more comprehensive form as a problem (P)

(P) $\quad a_i^Tx \leqslant b_i$, $i = 1,\ldots,m$, $x \in R^n$.

We note that the above construction is not a unique transformation of a linear programming problem into a system or systems of linear inequalities. We choose it because of its simplicity. Throughout these notes we assume that:

(A1) $\quad a_i \in Z^n$, $b \in Z^m$, $i = 1,\ldots,m$,

i.e., all data in (P) are integer (Z^n denotes set of all n-dimensional integer vectors) and

(A2) $\quad a_i \neq 0$, $i = 1,\ldots,m$.

To avoid some pathological cases we will require

(A3) $n \geqslant 2$.

From a practical point of view, the assumption (A1) is not a restriction since in practical problems data are rational numbers and can be converted into integers. The assumption (A2) may be easily verified, namely if $a_i = 0$ and $b_i \geqslant 0$, then the i-th inequality may be dropped, since it is satisfied by any $x \in R^n$, and if $a_i = 0$, but $b_i < 0$, then (P) is inconsistent. Obviously for $n = 1$, the system (P) can be solved in polynomial time for any m.

At the beginning of 1979 Khachian published a paper [5], in which he showed that one can solve (P) using operations of the form +, -, x, :, $\sqrt{\ }$ max and the number of such operations is bounded by some polynomial of n and of L, where L is defined as

$$1) \quad L = \sum_{i=1}^{m} \sum_{j=1}^{n} \log_2(|a_{ij}| + 1) + \sum_{i=1}^{m} \log_2(|b_i| + 1) + \log_2 mn + 1.$$

It is easy to see that L is "almost" equal the number of symbols 0 and 1 that are necessary to write (P) in the binary arithmetic. The "almost" comes from that the $\lceil \log_2(|a_{ij}| + 1) \rceil$ binary symbols are required to write a_{ij} in the binary arithmetic, where $\lceil x \rceil$ denote the smallest integer grater or equal x. But since the computational complexity of the method is estimated with the accuracy to the order of a polynomial, we can say that L is somehow proportional to the number of binary symbols necessary to write (P) in the binary arithmetic and it is somehow proportion-

al to the number of bits necessary to write (P) on a tape of a deterministic Turing machine. We will consider L as one of possible parameters describing system (P).
Carleson in [1] used another parameter, namely

$$|a_{ij}|, |b_i| < p, \quad i = 1,\ldots,m, \quad j = 1,\ldots,n.$$

Now we present an idea of the ellipsoidal algorithm solving graphically system of three inequalities in R^2. These inequalities are labeled in Fig. 1 as (1), (2) and (3) and they represent the inequality (1), (2) and (3) from our numerical example solved in Section 6. For simplicity of presentation we assume that F(P) ≠ ∅, i.e., the set of solutions to (P) is nonempty.

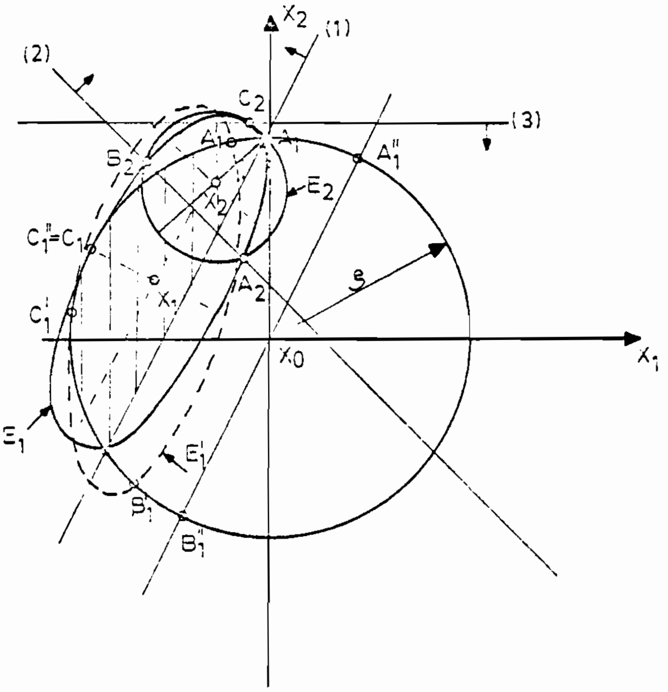


Fig. 1

At the initial iteration we construct a ball (an ellipsoid) E_0, with the centre x_0 that containes at least one feasible solution to (P). Since x_0 is not a solution to our system, we construct the next ellipsoid E_1, that covers the set of all candidates for solution to (P) contained in E_0. With reference to (1) the set of candidates represents a shaded part of E_0. We will show later that the ellipsoidal algorithm convergences independently of the choice of the reference constraint (in our example we may choose (2) as a reference constraint as well) but the speed of convergence depends heavily on that choice. We construct E_1 in the following way. Points A_1 and B_1 are intersection points of (1) with the boundary of E_o while the line parallel to (1) is tangent to E_0 at point C_1. These 3 points combined with the requirement that E_1 is an ellipsoid of the minimal volume define a unique ellipsoid E_1. From Fig. 1 we can see that the center x_1 of E_1 does not satisfy (2) although it satisfies (1), so in the next iteration we construct E_2 and check that its center, x_2, is a solution to (P).

In the above constructions on Fig. 1 and throughout this paper we assume that

(A4) all computations are exact

or can be carried out with infinite precision.

Computing on real computers with finite precision it is possible that instead of A_1 we find point A_1' and construct an ellipsoid E_1' instead of E_1. Then it is possible that we loose some part of the set of candidates for the solutions of (P) and, therefore it is possible that a computer gives the answer $F(P) = \emptyset$ while in reality $F(P) \neq 0$. We introduce the assumption (A4) because it simplifies the presentation of the algorithm. In the forthcoming paper we will lift this unrealistic assumption and describe the ellipsoid algorithm for real computers.

It follows from the above description that we have to proof that:

i) there exist an upper bound on the radius ρ of the initial hypersphere,

ii) the volume of the next ellipsoid is always strictly less than the volume of the previous one or that

$$R_k = \frac{\mathrm{Vol}\ E_{k+1}}{\mathrm{Vol}\ E_k} \leqslant q < 1 \quad \text{for any } k \geqslant 0. \tag{2}$$

We will call the number R_k the convergence rate of the ellipsoidal algorithm. In the next section we give a new extimation of ρ, while in Section 3 and 4 the convergence rate is studied. In these section we show how to construct the ellipsoidal algorithm with the best rate of convergence i.e. the algorithm for that R_k is the smallest possible for any $k \geqslant 0$. We note, that it is not by no means obvious that the construction of E_1 described in Fig. 1 gives the ellipsoidal algorithm with the best rate of convergence. A description of such an algorithm is given in Section 5 and in Section 6 a numerical example is solved. In the concluding remarks we give preliminary results of comparison between the Simplex Method and the ellipsoidal algorithm.

2. AN ESTIMATION OF THE RADIUS OF THE INITIAL BALL

We recall that the determinant of r by r matrix $B = (b_{ij})$ is the number

$$\det B = \Sigma(-1)^k b_{1,j_1} b_{2,j_2} \cdots b_{r,j_r}$$

where in each term the column (second) subscripts $j_1, j_2, \ldots, j_r$ are some ordering of $1, 2, \ldots, r$, and the sum is taken over all possible r! orderings. For each term, the exponent k of $(-1)^k$ is the number of inversions of the column subscripts $j_1, j_2, \ldots, j_r$. The permanent of r by r matrix B, per B, is defined in the same way as det B, but we drop $(-1)^k$. Therefore

$$\det \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = b_{11}b_{22} - b_{12}b_{21},$$

while

$$\operatorname{per} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = b_{11}b_{22} + b_{12}b_{21}.$$

The theorem below gives the answer to the question i) stated in the previous sections.

<u>Theorem 1</u>

If $F(P) \neq \emptyset$, then there exists a solution x^o to (P) in the Euclidean ball

$$E_0 = \{x \mid \|x\| \leqslant \rho \leqslant \frac{1}{2mn} 2^L\}$$

and a better estimation of ρ is

$$\rho \leqslant \frac{1}{2mn}\, 2^L - \Big(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}| + \sum_{i=1}^{m} |b_i| + \sum_{k=2}^{r} \Sigma \operatorname{per}|\bar{B}_k|\Big), \tag{3}$$

where B is a non-singular submatrix of (a_{ij}) with the maximal rank $r \leqslant \min(m,n)$ and $\Sigma\operatorname{per}|\bar{B}_k|$ denotes the sum of permanents of all possible principal square k by k matrices $|\bar{B}_k|$ obtained from $|B| = (|b_{ij}|)$, $i = 1,\ldots,r$, $j = 1,\ldots,r$.

Proof

It is possible that F(P) has no vertices at all, e.g. when $m = 1$. Therefore in the first part of the proof we show that x^o is an intersection point of some number of hyperplanes obtained from (P) and in the second part we estimate the norm of x^o.

Part I

For any $x \in R^n$ we define

$$I(x) = \{i \mid a_i^T x = b_i\}.$$

Let $\bar{x} \in F(P)$ be such that

$$|I(\bar{x})| = \max_{x \in F(P)} |I(x)|.$$

As $F(P) \neq \emptyset$ such an $\bar{x}$ exists and we may assume that

$$I(\bar{x}) = \{1,\ldots,k\},\ k \leqslant m.$$

So we have a system of equations

(P′) $a_i^T x = b_i$, $i = 1,\ldots,k$

and we wonder if a solution to (P′) is at the same time a solution to (P). Therefore we proof the following

Claim: If $x \in F(P')$, then $x \in F(P)$.

Assume that $x' \in F(P')$, but $x' \notin F(P)$ and define

$$I' = \{i \mid i > k,\ a_i^T x' > b_i\}.$$

As $\bar{x} \in F(P)$, then $a_i^T \bar{x} \leqslant b$ for $i \in I'$, and then there exist $0 < t_i < 1$ such that

$$a_i^T(\bar{x} + t_i(x' - \bar{x})) = b_i \text{ for } i \in I'.$$

Let

$$t_{i^*} = \min_{i \in I'} t_i$$

then $x'' = (\bar{x} + t_{i^*}(x' - \bar{x})) \in F(P)$ and

$$I(x'') \supset I(\bar{x}) \cup \{i^*\}$$

but that contradicts the fact that

$$|I(\bar{x})| = \max_{x \in F(P)} |I(x)|$$

and proves the Claim.

Now from (P') we choose the maximal independent system of equations and write it in the form

$$Bx_B + Nx_N = \bar{b}$$

where B is non-singular r by r matrix and $r \leqslant n$.

Setting $x_N = 0$, by Cramer's rule, we find

$$x_{Bj} = \frac{\det B_j}{\det B}$$

where B_j is obtained from B by substituting the j-th column of B by $\bar{b}$. Therefore we have shown that $x^o = (x_B, 0) \in F(P)$.

Part II

(Estimation of $||x^o||$).

$$(4) \qquad ||x^o|| = \left(\sum_{j=1}^{r} x_{Bj}^2\right)^{1/2} \leqslant \sum_{j=1}^{r} |x_{Bj}| = \sum_{j=1}^{r} \left|\frac{\det B_j}{\det B}\right| \leqslant$$

$$\leqslant \sum_{j=1}^{r} |\det B_j| \qquad \text{as } |\det B| \geqslant 1 \text{ by (A1)}$$

$$\leqslant \sum_{j=1}^{r} \text{per } |B_j| \qquad \text{as per } |B_j| \geqslant |\det B_j| \text{ for all } j.$$

From (1) we have

$$2^{L-1} = mn \prod_{i=1}^{m} \prod_{j=1}^{n} (|a_{ij}| + 1) \prod_{i=1}^{m} (|b_i| + 1)$$

and this is equivalent to

$$(5) \qquad \frac{1}{2mn} 2^L = 1 + \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}| + \sum_{i=1}^{m} |b_i| + \Sigma P^2 +$$

$$+ \Sigma P^3 + \ldots + \Sigma P^r + \ldots + P^{mn+n},$$

where ΣP^k denotes the sum of all possible products of k numbers ($2 \leqslant k \leqslant mn+n$) taken from the ordered set

$$\{|a_{11}|, |a_{12}|, \ldots, |a_{mn}|, |b_1|, |b_2|, \ldots, |b_m|\}.$$

ΣP^r includes all products from

$$\sum_{j=1}^{r} \text{per } |B_j| \quad \text{for all } j.$$

and moreover all products from per $|B| \geqslant |\det B|$, while

$$\Sigma P^2 + \Sigma P^3 + \ldots + \Sigma P^{r-1}$$

includes all products from the permanent of any principal square k by k matrix $|\bar{B}_k|(2 \leqslant k \leqslant r-1)$ taken from matrix $|B|$. As $|\det \bar{B}_k| \geqslant 1$ then per $|\bar{B}_k| \geqslant 1$ for any $2 \leqslant k \leqslant r$.

Combining (4) and (5) we have (3) . □

It is worth to note, that in general, $k > r$ cannot guarantee that $|\det B_k| \neq 0$ and per $|B_k| \neq 0$ where $|B_k|$ is a k by k matrix obtained from (a_{ij}).

Corollary 1

$$|\det B| \leqslant \frac{1}{2mn} 2^L,$$

$$|\det B_j| \leqslant \frac{1}{2mn} 2^L, \; j = 1,2,\ldots,r.$$

From Corollary 1 we conclude that any solution to (P) is either zero, i.e. $x^o = 0$, or it cannot be infinitly close to zero as

$$|x_{B_j}| = \left|\frac{\det B_j}{\det B}\right| \geqslant \frac{1}{|\det B|} \geqslant 2mn\, 2^{-L}.$$

In other words, the assumption (A1) odds a certain "discreteness" to the problem (P), namely, all points that we look at the proof of Theorem 1 are not arbitrary points in R^n, but have entries that are rational numbers with bounded denominators. This allows us to perturb the right-hand-side of (P) by a certain small number $\varepsilon \geqslant 0$ in such a way that a solution to the perturbed system which we call (P''), is at the same time a solution to (P).

Consider a system (P'') of strict inequalities

$$(P'') \qquad a_i^T x < b_i + \varepsilon, \quad i = 1,\ldots,m$$

where

(6) $$0 \leqslant \varepsilon \leqslant \frac{2m}{n} 2^{-L}.$$

Theorem 2

$$F(P) \neq \emptyset \text{ iff } F(P'') \neq \emptyset .$$

Proof

If $F(P) \neq \emptyset$, then $F(P'') \neq \emptyset$ as $\varepsilon \geqslant 0$. We show that the converse is also true by constructing a solution to (P) given a solution to (P'').

The set F(P'') is open so we take its closure in R^n and using the same reasoning as in Part I of the proof of Theorem 1 we derive a system of equations from (P''). A solution to this system can be written in the form $x = (x_B, x_N)$ where

$$x_{B_j} = \sum_{k=1}^{r} \frac{\det B_{kj}}{\det B}(b_k + \varepsilon), \; j = 1,\dots,r, \; x_N = 0,$$

and B is the same as in Theorem 1 and B_{kj} is a square matrix obtained from B by delating the k-th row and the j-th column. Setting this solution into (P''), we obtain for i = 1,...,m

$$\frac{1}{\det B} \sum_{j=1}^{r} \sum_{k=1}^{r} a_{ij} \det B_{kj} b_k - b_i < \varepsilon +$$

$$- \frac{\varepsilon}{\det B} \sum_{j=1}^{r} \sum_{k=1}^{r} a_{ij} B_{kj}.$$

We denote the left-hand-side of this inequality by Δ and note that it is the difference between $a_i^T x$ in the case $\varepsilon = 0$ and b_i, i.e., the right-hand-side in (P). We estimate Δ in the following way:

$$
\begin{aligned}
\Delta|\det B| &< \varepsilon|\det B| + \varepsilon \sum_{j=1}^{n} |\det B_j| && \text{as } n \geqslant r \\
&\leqslant \frac{1}{n^2} + \frac{1}{n} && \text{as } \varepsilon \leqslant \frac{2m}{n} 2^{-L} \quad \text{and } |\det B_j| \leqslant \frac{1}{2mn} 2^{L} \\
&< 1 && \text{as } n \geqslant 2
\end{aligned}
$$

Now from $|\det B| \geqslant 1$ and $\Delta|\det B| \geqslant 0$ and integer we have $\Delta = 0$. Therefore $(x_B, x_N) \in F(P)$ for $0 \leqslant \varepsilon \leqslant \frac{2m}{n} 2^{-L}$. □

By Vol $(F(P'') \cap E_0)$ we denote the volume of all solutions to (P'') inside E_0.

Theorem 3

If $F(P'') \neq \emptyset$, then Vol $(F(P'') \cap E_0) \geqslant (2m)^n 2^{-nL}$.

Proof

If $F(P'') \neq \emptyset$, then by Theorem 1 $F(P'') \cap E_0 \neq \emptyset$ and moreover the above intersection of sets has nonempty interior, i.e. Int $(F(P'') \cap E_0) \neq \emptyset$. Therefore there exist $x_0 \in$ Int $(F(P'') \cap E_0)$. If $x_0 \neq 0$, then we shift the origin to 0. The set $F(P'') \cap E_0$ is open so we take its closure in R^n and using the same reasoning as in Part I of Theorem 1 we find points $x_1, \ldots, x_n$ such that

$$x_i = \frac{u_i}{\det D_i} e_i = \frac{1}{\det D_i} \bar{e}_i$$

where e_i is a unit vector and u_i is some integer.

Points $x_1, \ldots, x_n$ are linearly independent and together with x_0 form a simplex in R^n and the volume of this simplex is given by a well-known formula

$$V = \frac{1}{n!} \left|\det\begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_0 & x_1 & \ldots & x_n \end{pmatrix}\right|$$

and Vol $(F(P'') \cap E_0) \geqslant V$ as V is the volume of the symplex inside $F(P'') \cap E_0$. We estimate V in the following way:

$$V = \frac{1}{n!} |\det(x_1\ x_2\ \dots\ x_n)| \qquad \text{as } x_0 = 0$$

$$= \frac{1}{n!} \frac{1}{|\det D_1| \dots |\det D_n|} |\det(\bar{e}_1\ \bar{e}_2\ \dots\ \bar{e}_n)|$$

$$\geqslant \frac{1}{n!} \frac{1}{|\det D_1| \dots |\det D_n|} \qquad \text{as } |\det(\bar{e}_1 \dots \bar{e}_2)| \geqslant 1$$

$$\geqslant \frac{1}{n!} (2mn)^n\ 2^{-nL} \qquad \text{by Corollary 1}$$

$$\geqslant (2m)^n 2^{-nL} \qquad \text{as } n! \geqslant n^n$$

and this concludes the proof. □

Theorems 2 and 3 show that assuming $R_k < 1$ the ellipsoidal algorithm requires finite number of iterations to reduce the initial $E_0 = \{x|\ ||x|| \leqslant \frac{1}{2mn} 2^L\}$ to the ellipsoid which volume is at most $(2m)^n\ 2^{-nL}$.

3. DESCRIPTION OF THE ALGORITHM

We will use two ways of describing an ellipsoid E in R^n:

1^o as a shifted image of a unit ball, i.e., the set of points in R^n such that

(8) $\qquad E = \{x|x = \bar{x} + Qz,\quad ||z|| \leqslant 1\}$

where $\bar{x}$ is the center of E and Q is any n by n matrix with det Q ≠ 0. (See Fig. 2)

2^o as a set of points described by a quadratic positive definite form, i.e., the set of points that satisfy a quadratic form

(9) $\qquad (x - \bar{x})^T A^{-1}(x - \bar{x}) \leqslant 1$

where A is a symmetric positive definite matrix. We recall that A is positive definite if $x^T Ax > 0$ for every $x \in R^n$ except $x = 0$.

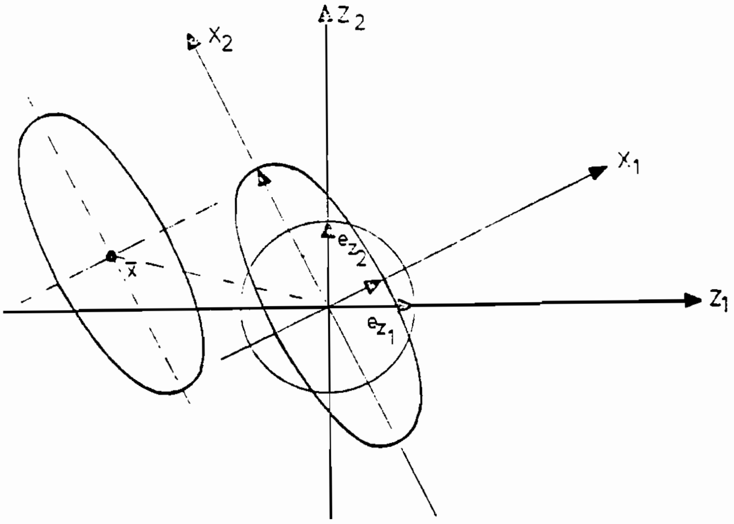


Fig. 2

Now we establish relations between these two descriptions. From (8), det Q $\neq$ 0 we obtain

$$z = Q^{-1}(x - \bar{x}),$$

and since $||z|| \leqslant 1$ is equivalent to $z^T z \leqslant 1$ we have

$$(Q^{-1}(x - \bar{x}))^T Q^{-1}(x - \bar{x}) \leqslant 1$$

$$(x - \bar{x})^T (Q^{-1})^T Q^{-1}(x - \bar{x}) \leqslant 1.$$

Therefore from (9)

(10) $$A^{-1} = (Q^{-1})^T Q^{-1} = (Q^T)^{-1} Q^{-1} = (QQ^T)^{-1}$$

So

(11) $$A = QQ^T.$$

As we described it in Section 1 the ellipsoidal algorithm constructs a sequence of ellipsoids

$$E_0 = (x_0, Q_0),\ E_1 = (x_1, Q_1), \dots, E_k = (x_k, Q_k), \dots$$

where $x_0 = 0$ and $Q_0 = \rho I$. (I is the unit matrix) since then we have from (9) and (11)

$$x^T(QQ^T)^{-1}x \leqslant 1$$

but $(QQ^T)^{-1} = \frac{1}{\rho^2} I$, so

$$\frac{1}{\rho^2} x^T I x = \frac{1}{\rho^2}||x||^2 \leqslant 1.$$

Therefore by Theorem 1

$$||x|| \leqslant \rho \leqslant \frac{1}{2mn} 2^L.$$

Assume now that we have finished the k-th iteration, $k = 0,1,2,\ldots,$ so we have constructed $E_k = (x_k,Q_k)$. To simplify notations we denote $E_k = (x_k,Q_k)$ as $E = (\bar{x},Q)$. Then one and only one of two possibilities may occur:

1° $\quad a_i^T\bar{x} - b_i \leqslant 0 \quad$ for $i = 1,\ldots,m,$

then $\bar{x}$ satisfies each inequality of (P) therefore $\bar{x} \in F(P)$ and the ellipsoidal algorithm stops.

2° For some i $\quad a_i^T\bar{x} - b_i > 0$, then we construct the next, i.e., the (k+1)-st ellipsoid $E_{k+1} = (x_{k+1},Q_{k+1})$, that for simplicity of notation we denote as $E' = (x',Q')$.

We define $d_i(\bar{x})$ as the algebraic distance from $\bar{x}$ to the hyperplane bounding the half-space $H_i = \{x \in R^n | a_i^T x \leqslant b_i\}$. Let D denote the maximal algebraic distance from $\bar{x}$, namely,

$$D = \frac{a_l^T\bar{x} - b_l}{||Q^T a_l||} = \max_i \frac{a_i^T\bar{x} - b_i}{||Q^T a_i||} = \max_i d_i(\bar{x}) \tag{12}$$

We call $a_l^T x \leqslant b_l$ the most violated inequality. By $u_i(\bar{x})$ we denote the difference $a_i^T\bar{x} - b_i$.

Theorem 4

i) If $d_i(\bar{x}) > 1$, then $H_i \cap E = \emptyset$ and (P) is inconsistent.

ii) If $-1 \leqslant d_i(\bar{x}) \leqslant 1$, then $H_i \cap E \neq \emptyset$.

iii) If $d_i(\bar{x}) < -1$, then $H_i \cap E = E$ and H_i is inessential.

Moreover, if $d_i(\bar{x}) = 1$, then $H_i \cap E$ degenerates to a point and for $d_i(\bar{x}) \leqslant 0$, $\bar{x} \in H_i \cap E$, i.e., $\bar{x}$ satisfies $a_i^T x \leqslant b_i$.

Proof

We prove only i). The proof of ii) and iii) is similar.

For any $x \in E$, i.e., $x = \bar{x} + Qz$, where $||z|| \leqslant 1$, we have

$$\begin{aligned}
a_i^T(\bar{x} + Qz) &= a_i^T\bar{x} + a_i^T Qz \\
&= b_i + u_i(\bar{x}) + Q^T a_i z && \text{as } u_i(\bar{x}) = a_i^T\bar{x} - b_i \\
&\geqslant b_i + u_i(\bar{x}) - ||Q^T a_i||\ ||z|| && \text{as } |Q^T a_i z| \leqslant \\
& && \leqslant ||Q^T a_i||\ ||z|| \\
&\geqslant b_i + u_i(\bar{x}) - ||Q^T a_i|| && \text{as } ||z|| \leqslant 1 \\
&\geqslant b_i + u_i(\bar{x})(1 - \frac{1}{d_i(\bar{x})}) && \text{as } d_i(\bar{x}) = \frac{u_i(\bar{x})}{||Q^T a_i||}
\end{aligned}$$

If $d_i(\bar{x}) > 1$ then for any $x \in E$

$$a_i^T(\bar{x} + Qz) \geqslant b_i + u_i(\bar{x})(1 - \frac{1}{d_i(\bar{x})}) > b_i \tag{13}$$

so no $x \in E$ can satisfy inequality $a_i^T x \leqslant b_i$ and therefore the system (P) is consistent. □

We note that if $0<D<1$ then

$$a_1^T(\bar{x} + Qz) \geqslant b_1 + u_1(\bar{x})(1 - \frac{1}{D}) < b_1 + u_1(\bar{x})$$

so there exist in E points form that the algebraic distance to the bounding hyperplane of H_1 is smaller than from $\bar{x}$ to the same hyperplane and we construct a new ellipsoid $E' = (x', Q')$ that covers $H_1 \cap E$.

We assume that x' is given by the formula

$$(14) \qquad x' = x - h \frac{QQ^T a_1}{||Q^T a_1||}$$

where $h \geqslant 0$ is a parameter, and Q' is given by the formula

$$(15) \qquad Q' = QGH$$

where G is an n by n orthogonal matrix that transforms vectors from z-coordinate system into x-coordinate system (see Fig. 2), with the first column given as

$$e_1 = \frac{Q^T a_1}{||Q^T a_1||}$$

and H is a diagonal matrix

$$H = \begin{pmatrix} v^{-1} & & & 0 \\ & w^{-1} & & \\ & & \ddots & \\ 0 & & & w^{-1} \end{pmatrix}$$

where $v,w > 0$ are parameters of this transformation.

Now we choose the parameters h,v,w in such a way that the volume of E' is as small as possible. Therefore we start constructing the ellipsoidal algorithm with the best rate of convergence.

A solution to (P) has to satisfy two conditions, first the most violated inequality, i.e.,

$$a_1^T x \leqslant a_1 \bar{x} - D \, ||Q^T a_1||$$

and secondly $x \in E$, i.e., $x = \bar{x} + Qz$, $||z|| \leqslant 1$. Therefore

$$(16) \qquad a_1^T(\bar{x} + Qz) \leqslant a_1 \bar{x} - D \, ||Q^T a_1||.$$

But the left-band-side of this inequality equals

$$(17) \qquad a_1^T\bar{x} + a_1^TQz = ||a_1^TQ||z_1$$

as from the definition of an orthogonal matrix G we get $a_1^TQz = ||a_1^TQ||z_1$. Combining (16) and (17) gives

$$(18) \qquad z_1 \leqslant D.$$

A new ellipsoid $E' = \{x | x = x' + Q'z' \; , \; ||z'|| \leqslant 1\}$ has to cover all points of the set

$$E \cap \{x | a^Tx \leqslant a_1^T\bar{x} - D\, ||Q^Ta_1||\}$$

so we have a system of vector equations and one scalar equation

$$(19) \qquad \begin{array}{ll} x' + Q'z' = \bar{x} + Qz & \text{and } ||z'|| \leqslant 1 \\ \text{for all } z_1 \leqslant -D & \text{and all } ||z|| \leqslant 1. \end{array}$$

In (19) a vector z is a parametr. Setting (14) and (15) in (19) we obtain:

$$\bar{x} - h\frac{QQ^Ta_1}{||Q^Ta_1||} + (QGH)z' = \bar{x} + Qz \quad \text{and } ||z'|| \leqslant 1$$

or

$$z' = (QGH)^{-1}[Qz + h\frac{QQ^Ta_1}{||Q^Ta_1||}] = (QGH)^{-1}[Qz + hQe_1].$$

The above equality can be written in the form

$$z' = v(h + z_1)e_{z_1} + w\sum_{j=2}^{n} z_je_{z_j} \; ,$$

where $e_{z_1}, e_{z_2}, \ldots, e_{z_n}$ is the original basis (see Fig. 2). Then the requirement $||z'|| \leqslant 1$ takes the form

$$(20) \qquad v^2(h + z_1)^2 + w^2(\sum_{j=2}^{n} z_j)^2 \leqslant 1$$

for any $z^T = (z_1, \ldots, z_n)^T$ such that $||z|| \leqslant 1$ and $z_1 \leqslant -D$. This can be expressed as a parametric inequality, namely for any t such that $0 \leqslant D \leqslant t < 1$ from (20) we have

(21) $$g(t) = v^2(h - t)^2 + w(1 - t^2) \leqslant 1$$

and g(t) is a convex function of a parameter t.

It is well-known that for a linear transformation given by (15) the volume of E' is equal

$$\mathrm{Vol}(E') = v^{-1}w^{-(n-1)}\mathrm{Vol}(E).$$

So minimization of Vol (E') is equivalent to minimization of the rate of convergence $R^k = v^{-1}w^{-(n-1)}$ or to maximization of a convex function

(22) $$f(v,w) = v^2w^{2n-2}$$

under the constraints given by (21) for $0 \leqslant D \leqslant t \leqslant 1$. This is a convex programming problem and the maximum of f(v,w) is obtained on the boundry of the set of feasible parameters. For t = D we have

(23) $$v^2(h - D)^2 + w^2(1 - D^2) \leqslant 1$$

while for t = 1, we obtain

(24) $$v^2(1 - h)^2 \leqslant 1$$

The last inequality gives us

(25) $$v^2 \leqslant \frac{1}{(1 - h)^2}$$

and taking this into account from (23), we get

(26) $$w^2 \leqslant \frac{1}{1 - D^2}\left(1 - \frac{(h - D)^2}{(1 - h)^2}\right).$$

Setting (25) and (26) into (22), we obtain

$$f(h,D) = \frac{1}{(1 - h)^2}\,\frac{1}{(1 - D^2)^{n-1}}\left(1 - \frac{(h - D)^2}{(1 - h)^2}\right)^{n-1}$$

as a function of h and D.

The optimal value of h, h^*, that maximizes $f(h,D)$ is found by setting the partial derivative $\frac{\partial f(h,D)}{\partial h}$ to zero

$$h^* = \frac{Dn+1}{h+1}. \tag{27}$$

Then from (25) and (26) we obtain

$$v^* = \frac{n+1}{n} \frac{1}{1-D}, \tag{28}$$

$$w^* = \frac{\sqrt{n^2-1}}{n} \frac{1}{\sqrt{1-D^2}} \tag{29}$$

where $0<D<1$.

Now we can compute the rate of convergence

$$R_k^* = \frac{1}{v^*(w^*)^{n-1}} = \frac{n^n}{n+1} (n^2-1)^{-\frac{n-1}{2}} (1-D)(1-D^2)^{\frac{n-1}{2}}. \tag{30}$$

So the rate of convergence is a function of the maximal algebraic distance of (P) defined by (12). We will study this function in the next section.

From the algorithmic point of view it is better to describe an ellipsoid by means of a quadratic from. Recalling that $E = E_k = (x_k, A_k)$ and $E' = E_{k+1} = (x_{k+1}, A_{k+1})$ and using (11), (14), (15) and (27) - (29) we write a recursive formulae

$$x_{k+1} = x_k - \frac{Dn+1}{n+1} \frac{A_k a_1}{\sqrt{a_1^T A_k a_1}} \tag{31}$$

$$A_{k+1} = \frac{n^2(1-D^2)}{n^2-1} \left(A_k - \frac{2(Dn+1)}{(n+1)(D+1)} \frac{A_k a_1 (A_k a_1)^T}{a_1^T A_k a_1}\right) \tag{32}$$

for $k = 0,1,2,\ldots$.

The initial values are

$$(33) \qquad x_0 = 0 \quad \text{and} \quad A_0 = \rho^2 I = (\frac{1}{2mn})^2 \; 2^{2L} I.$$

In (31) and (32) D is defined as

$$(34) \qquad D = \max_i \frac{a_i^T \bar{x}_k - b_i}{\sqrt{a_i^T A_k a_i}} = \frac{a_l^T \bar{x}_k - b_l}{\sqrt{a_l^T A_k a_l}}$$

This completes a description of the ellipsoidal algorithm for linear programming since we have shown how to construct the initial ellipsoid E_0 and how to construct the ellipsoid E_{k+1} from the ellipsoid E_k for $k = 0,1,2,\ldots$.

4. THE CONVERGENCE RATE

Now we prove that (30) is the best possible convergent rate of the ellipsoidal algorithm for linear programming.

Theorem 5

$$R_k^* = \frac{n^n}{n+1}(n^2-1)^{-\frac{n-1}{2}}(1-D)(1-D^2)^{\frac{n-1}{2}}$$

is the best possible rate of convergence of the ellipsoidal algorithm

Proof

We construct E_{k+1} for $k = 0,1,2,\ldots$ in such a way that it covers all candidates for solution inside E_k. In (19) we do not specify any restriction on intersection points of the boundry of E_k and of the boundry of E_{k+1}. Next we establish all parameters of E_{k+1}, i.e., h^*, v^*, w^*, in such a way that they minimize the rate of convergence and for these values of h, v and w we obtain (30).

Therefore (30) gives the best possible rate of convergence. □

It is interesting to note that Murray in [6] has received the same rate of convergence under the same assumption about the intersection points of E_k and E_{k+1} as we made in Section 1. We prove this in

Theorem 6

The ellipsoidal algorithm has the following features for k = 0,1, 2,... :

i) x_{k+1} satisfies the most violated inequality,

ii) A_{k+1} is positive definite,

iii) the boundary of E_{k+1} intersects the boundary of E_k at the same points as the hyperplane $a_1^T x = b_1$ corresponding to the most violated inequality,

iv) E_{k+1} touches E_k at the same point as the hyperplane parallel to $a_1^T x = b_1$ touches E_k.

Proof

i) We have to prove that x_{k+1} satisfies $a_1^T x \leqslant b_1$

$$a_1^T x_{k+1} - b_1 = a_1^T\left(x_k - \frac{Dn+1}{n+1}\,\frac{A_k a_1}{\sqrt{a_1^T A_k a_1}}\right) - b_1$$

$$= a_1^T x_k - \frac{Dn+1}{n+1}\sqrt{a_1^T A_k a_1} - b_1$$

$$= a_1^T x_k - \frac{Dn+1}{n+1}\,\frac{a_1^T x_k - b_1}{D} - b_1 \qquad \text{by (12)}$$

$$= \frac{D-1}{Dn+D}(a_1^T x_k - b_1) < 0 \qquad \text{as } 0<D<1$$

ii) The proof is exactly the same as the proof of Sylvester's theorem (see e.g. [4] p. 340).

iii) If we multiply the equality

$$(x - x_k)^T A_k^{-1} (x - x_k) = 1$$

by a positive number $a_1^T A_k a_1$, then

$$(x - x_k)^T a_1^T A_k a_1 A_k^{-1} (x - x_k) = a_1^T A_k a_1$$

$$a_1^T (x - x_k) A_k (A_k^{-1})^T a_1^T (x - x_k) = a_1^T A_k a_1$$

(35) $$(a_1^T (x - x_k))^2 = a_1^T A_k a_1 \qquad \text{by (9) } A_k = A_k^T .$$

Similarly for E_{k+1} we have

(36) $$(a_1^T (x - x_{k+1}))^2 = a_1^T A_{k+1} a_1 .$$

Setting in (36) the expressions (31) and (32), we obtain

(37) $$\left(a_1^T x - a_1^T x_k + \frac{Dn+1}{n+1} \sqrt{a_1^T A_k a_1}\right)^2 = \frac{n^2 (1-D)^2}{(n+1)^2} a_1^T A_k a_1 .$$

But x has to satisfy $a_1^T x = b_1$ and (35), therefore by (34)

$$a_1^T x - b_1 - D \sqrt{a_1^T A_k a_1} = - \sqrt{a_1^T A_k a_1} .$$

We take minus square root, since the common points are in the half-space $a_1^T x \leqslant b_1$

(38) $$a_1^T x = b_1 - (1-D) \sqrt{a_1^T A_k a_1} .$$

Setting (38) in (37) the left-hand-side of it equals by (34) the right-hand-side of (37). This proves (iii).

iv) The general expression for a supporting hyperplane, which is tangent to some convex set at x_0 is

(39) $$z - z_0 = (\nabla z(x_0))^T (x - x_0),$$

where in our case $z = (x-x_k)^T A_k^{-1} (x-x_k) - 1$.

Next in the same way as in iii) we prove that the common point of E_k, E_{k+1} and (39) is the same. This concludes proof. □

From Theorem 5 one can see that the best rate of convergence is a function of the maximal algebraic distance D.

It is interesting to note that for D = 0 we have from Theorem 5

$$(40) \qquad R_k^*(0) = \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{\frac{n-1}{2}} < e^{-\frac{1}{2(n+1)}} .$$

Obviously $R_k^*(D) < 1$ for any $0 \leqslant D \leqslant 1$ and any $n \geqslant 2$. Therefore the ellipsoidal algorithm is convergent.

Gacs and Lovasz proved in [3] (see also [6] and [5]) that the performance of Khachian's algorithm is given by (40). This version of the ellipsoidal algorithm constructs the next ellipsoid taking into account points $A_1''B_1''C_1''$ on Fig.1. Therefore one may conclude that the performance of Khachian's version is a lower bound on the best rate of convergence given by (30).

By M we denote the maximal number of iterations required to solve (P) by the ellipsoidal algorithm

Theorem 7

$$M \leqslant \lceil 6n^2L \rceil$$

Proof

Suppose that the algorithm does not stop after M iterations, and yet (P) has a solution. Then by Theorem 3 the set S of its solutions inside E_0 has the volume at least $(2m)^n 2^{-nL}$ and $S \subset E_M$. So Vol $(E_M) \geqslant (2m)^n\ 2^{-nL}$.

Obviously the maximal number of iterations is obtained if the convergence rate is given by (40) for any k = 0,1,2,... . But then

$$\begin{aligned} \mathrm{Vol}(E_M) &\leqslant \mathrm{Vol}(E_0)\cdot\exp(-\frac{M}{2(n-1)}) \\ &\leqslant (2\cdot\frac{1}{2mn}\,2^L)^n\cdot\exp(-\frac{6n^2L}{2(n+1)}) \qquad \text{by Theorem 1} \\ &\leqslant (\frac{1}{mn})^n\,2^{-nL} \end{aligned}$$

So Vol $(E_M) < (2m)^n\,2^{-nL}$, a contradiction. Therefore $F(P) = \emptyset$ □

The same estimation of M has been obtained by Gacs and Lovasz in [3].

5. THE FIRST COMPUTER VERSION OF THE ALGORITHM

In many problems taken from practice we know that $F(P) \neq \emptyset$ although we do not know the solution to (P), e.g. any practical linear programming problem is feasible, and therefore $F(P) \neq \emptyset$ in such a case. This reduces the number of iterations substantially.

If $F(P) \neq \emptyset$ we can start from a smaller ball E_0' instead of E_0 choosing ρ' in such a way that

$$(41) \qquad (\rho')^2 = 4D^2,$$

then by (12) the new value of D will be 0.5. If for some $k = 1,2, \ldots$ we obtain $D > 1$ we cannot say by (13) that $F(P) = \emptyset$. This only means that our choice of ρ is not good for all iterations and we have to increase E_k multiplying A_k by $4D^2$ which again gives the new value of D equal 0.5.

Below we give the algorithm written in an ALGÒL-like language. We assume that all data m,n, A and b given. We introduce a boolean variable

$$\text{SURE} = \begin{cases} 1, \text{ if we are sure that } F(P) \neq \emptyset \\ 0, \text{ otherwise} \end{cases}$$

which also has to be given in advance.

<u>Algorithm A</u>

```
STEP 1 (Initialization, $k = 0$).
Compute L by (1),
$\rho : = \frac{1}{2mn} 2^L$;     $M : = 6 n^2 L$
$x_k := 0$ and $A_k : = \rho^2 I$.
If $b_i \geqslant 0$ for $i = 1,\ldots,m$, then stop ($x_k \in F(P)$)
else compute
    $D = \frac{-b_q}{\sqrt{a_q^T A_k a_q}} = \max \{\frac{-b_1}{\sqrt{a_1^T A_k a_1}}, \ldots, \frac{-b_m}{\sqrt{a_m^T A_k a_m}}\}$
1 := q;
If SURE = 1 then $\rho^2 : = 4D^2$; $A_k : = \rho^2 A_k$; $D : = 0.5$
else go to STEP 2.
STEP 2 (Main iteration, $k \geqslant 1$)
Compute $x_{k+1}$ by (31) and $A_{k+1}$ by (32);
If $u_i(x_{k+1}) = a_i^T x_{k+1} - b_i \leqslant 0$ for $i = 1,\ldots,m$ then stop else
compute
    $D = \frac{u_c(x_{k+1})}{\sqrt{a_q^T A_{k+1} a_q}} = \max_i \{\frac{u_1(x_{k+1})}{\sqrt{a_1^T A_{k+1} a_1}}, \ldots, \frac{u_m(x_{k+1})}{\sqrt{a_m^T A_{k+1} a_m}}\}$
1 := q;
If $D > 1$ and SURE = 1 then $\rho^2 : = 4D^2$; $A_{k+1} = \rho^2 A_{k+1}$;
$D := 0.5$;
$k := k + 1$
If $k > M$ then stop ($F(P) = \emptyset$)
else go to STEP2.
```

6. NUMERICAL EXAMPLES

Using the first computer version of the algorithm we solve the following example

$$(P) \quad \begin{aligned} 2x_1 - x_2 &\leqslant -2 && (1) \\ -2x_1 - 2x_2 &\leqslant -1 && (2) \\ x_2 &\leqslant q && (3) \end{aligned}$$

where q is a parameter. We will solve three cases:

Case A	$q = 2$, then $F(P) \neq 0$
B	$q = 1$, then $F(P) = \{(-0.5, 1)\}$
C	$q = 0$, then $F(P) = \emptyset$

In all cases $\rho = 1944$.

If we set SURE = 1 in the case A then starting from $x_0 = 0$ we have $(\rho') = 4D^2 = 3.2$. Next we find $x_1^T = (-1{,}066667,\ 0.533333)^T$ and that the second constraints is violated. In next iteration we find $x_2^T = (-0.538367,\ 1.498846)^T \in F(P)$. The geometrical representation of these results is given in Fig. 1.

Göran Tellström from the Department of Mathematics has coded the first computer version of the algorithm and solved this example obtaining the following results:

Case	Number of iterations		Solution $x = \binom{x_1}{x_2}$	
	SURE = 1	SURE = 0	SURE = 1	SURE = 0
A	2	11	-0.538367 1.498846	-0.601530 1.888583
B	17	-	-0.500056 1.00005	-
C	-	6	-	$F(P) = \emptyset$

7. CONCLUDING REMARKS

The most important question right now can be formulated in the following way:

> Q: How good is the ellipsoidal algorithm in comparison with the Simplex Method?

The answer requires results of a serious computer experiment we have already started. Dantzig in [2] compare these two methods basing mostly on [3]. Here we would like to underline some points:

1. The Simplex Method is in fact a discrete method while the ellipsoidal algorithm is a continuous method.
2. The numerical stability of the Simplex Method is unknown, because we do not know how to define a conditioning number for a linear programming problem. The ellipsoidal algorithm is numerically stable, although the precision of computations cannot be realized on any computer for practical problems. One has to note that this precision is far beyond the needs of practice. It is possible to construct a near-optimal ellipsoidal elgorithm that will not require such a precision of computations.

REFERENCES

[1] Carleson, L. (1980). Hacian's Method in Linear Programming, Report No. 3. Institute Mittag-Leffler.

[2] Dantzig, G.B. (1979). Comments on Khachian's Algorithm for Linear Programming, Technical Report SOL 79-22. Systems Optimization Laboratory, Department of Operations Research, Stanford University.

[3] Gacs, P. and Lovasz, L. (1979). Khachian's Algorithm for Linear Programming, Report STAN-CS-79-750. Computer Science Department, Stanford University.

[4] Hohn, F.E. (1964). Elementary Matrix Algebra, Mac Millan, New York.

[5] Khachian, L.G. (1979). Polynomial Algorithm for Linear Programming. (In Russian), Doklady Akademii Nauk SSSR, Vol. 244 No. 5.

[6] Murray, W. (1979). Ellipsoidal Algorithms for Linear Programming, Working Paper 79-1, Systems Optimization Laboratory, Department of Operations Research, Stanford University.

LARGE-SCALE LINEAR PROGRAMMING

Proceedings of a IIASA Workshop,
2–6 June 1980

Volume 2

George B. Dantzig, M.A.H. Dempster,
and Markku Kallio
Editors

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
Laxenburg, Austria
1981

OTHER ITERATIVE ALGORITHMS

THE LAGRANGIAN RELAXATION METHOD FOR SOLVING INTEGER PROGRAMMING PROBLEMS*

Marshall L. Fisher**

Department of Decision Sciences
The Wharton School
University of Pennsylvania

One of the most computationally useful ideas of the 1970's is the observation that many hard integer programming problems can be viewed as easy problems complicated by a relatively small set of side constraints. Dualizing the side constraints produces a Lagrangian problem that is easy to solve and whose optimal value is a lower bound (for minimization problems) on the optimal value of the original problem. The Lagrangian problem can thus be used in place of a linear programming relaxation to provide bounds in a branch and bound algorithm. This approach has led to dramatically improved algorithms for a number of important problems in the areas of routing, location, scheduling, assignment, and set covering. This paper is a review of Lagrangian relaxation based on what has been learned in the last decade.

*Published in *Management Science* 27:1, 1981. Reproduced with permission. Copyright © 1981 The Institute of Management Sciences.

**This paper was supported in part by NSF Grant ENG 78-26500 to the University of Pennsylvania.

1. Introduction

It is well-known that combinatorial optimization problems come in two varieties. There is a small number of "easy" problems which can be solved in time bounded by a polynomial in the input length and an all-too-large class of "hard" problems for which all known algorithms require exponential time in the worst-case. Among the hard problems, there are "easier hard" problems, like the knapsack problem, that have pseudo-polynomial algorithms that run in polynomial time if certain numbers in the problem data are bounded.

One of the most computationally useful ideas of the 1970's the observation that many hard problems can be viewed as easy problems complicated by a relatively small set of side constraints. Dualizing the side constraints produces a Lagrangian problem that is easy to solve and whose optimal value is a lower bound (for minimization problems) on the optimal value of the original problem. The Lagrangian problem can thus be used in place of a linear programming relaxation to provide bounds in a branch and bound algorithm. As we shall see, the Lagrangian approach offers a number of important advantages over linear programming.

There were a number of forays prior to 1970 into the use of Lagrangian methods in discrete optimization, including the Lorie-Savage [31] approach to capital budgeting, Everett's proposal for "generalizing" Lagrange multipliers [14] and the philosophically-related device of generating columns by solving an easy combinatorial optimization problem when pricing out in the simplex method [24]. However, the "birth" of the Lagrangian approach as it exists today occurred in 1970 when Held and Karp [27, 28] used a Lagrangian problem based on minimum spanning trees to devise a dramatically successful algorithm for the traveling salesman problem. Motivated by Held and Karp's success, Lagrangian methods were applied in the early 70s to scheduling problems (Fisher [15]) and the general integer programming problem (Shapiro [42], Fisher and Shapiro [16]). Lagrangian methods had gained considerable currency by 1974 when Geoffrion [22] coined the perfect name for this approach--"Lagrangian relaxation." Since then the list of applications of Lagrangian relaxation has grown to include over a dozen of the most infamous combinatorial optimization problems. For most of these problems, Lagrangian relaxation has provided the best existing algorithm for the problem and has enabled the solution of problems of practical size.

This paper is a review of Lagrangian relaxation based on what has been learned in the last decade. The reader is referred to Shapiro [43] for another recent survev of Lagranqian relaxation from a somewhat different perspective. The recent book by Shapiro [44] marks the first appearance of the term Lagrangian relaxation in a textbook.

2. Basic Constructions

We begin with a combinatorial optimization problem formulated as the integer program

$$\begin{aligned} Z = \min \quad & cx \\ \text{s.t.} \quad & Ax = b \\ & Dx \le e \qquad\qquad (P) \\ & x \ge 0 \text{ and integral} \end{aligned}$$

where x is n × 1, b is m × 1, e is k × 1 and all other matrices have conformable dimensions. Let (LP) denote problem (P) with the integrality constraint on x relaxed, and let Z_{LP} denote the optimal value of (LP).

We assume that the constraints of (P) have been partitioned into the two sets $Ax = b$ and $Dx \leq e$ so as to make it easy to solve the Lagrangian problem

$$\begin{aligned} Z_D(u) = \min \quad & cx + u\,(Ax - b) \\ & Dx \le e \qquad\qquad (LR_u) \\ & x \ge 0 \text{ and integral} \end{aligned}$$

where $u = (u_1, \ldots, u_m)$ is a vector of Lagrange multipliers. By "easy to solve" we of course mean easy relative to (P). For all applications of which I am aware, the Lagrangian problem has been solvable in polynomial or pseudo-polynomial time.

For convenience we assume that (P) is feasible and that the set $X = \{x | Dx \le e,\ x \ge 0 \text{ and integral}\}$ of feasible solutions to (LR_u) is finite. Then $Z_D(u)$ is finite for all u. It is straightforward to extend the development when these assumptions are violated or when inequality constraints are included in the set to be dualized.

It is well-known that $Z_D(u) \le Z$. This is also easy to show by assuming an optimal solution x* to (P) and observing that

$$Z_D(u) \le cx^* + u(Ax^* - b) = Z \quad .$$

The inequality in this relation follows from the definition of $Z_D(u)$ and the equality from $Z = cx^*$ and $Ax^* - b = 0$. If $Ax = b$ is replaced by $Ax \le b$ in (P), then we require $u \ge 0$ and the argument becomes

$$Z_D(u) \le cx^* + u(Ax^* - b) \le Z$$

where the second inequality follows from $Z = cx^*$, $u \ge 0$ and $Ax^* - b \le 0$. Similarly, for $Ax \ge b$ we require $u \le 0$ for $Z_D(u) \le Z$ to hold.

We will discuss in a later section methods for determining u. In general, it is not possible to guarantee finding u for which $Z_D(u) = Z$, but this frequently happens for particular problem instances.

The fact that $Z_D(u) \le Z$ allows (LR_u) to be used in place of (LP) to provide lower bounds in a branch and bound algorithm for (P). While this is the most obvious use of (LR_u), it has a number of other uses. It can be a medium for selecting branching variables and choosing the next branch to explore. Good feasible solutions to (P) can frequently be obtained by perturbing nearly feasible solutions to (LR_u). Finally, Lagrangian relaxation has been used recently [10, 20] as an analytic tool for establishing worst-case bounds on the performance of certain heuristics.

3. Example

The generalized assignment problem is an excellent example for illustrating Lagrangian relaxation because it is rich with readily apparent structure. The generalized assignment problem (GAP) is the integer program

$$Z = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1,\ j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} a_{ij}\quad x_{ij} \le b_i,\ i = 1, \ldots, m \tag{3}$$

$$x_{ij} = 0 \text{ or } 1, \text{ all } i \text{ and } j. \tag{4}$$

There are two natural Lagrangian relaxations for the generalized assignment problem. The first is obtained by dualizing constraints (2).

$$Z_{D1}(u) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{j=1}^{n} u_j \left(\sum_{i=1}^{m} x_{ij} - 1\right)$$

subject to (3) and (4)

$$(LR1_u)$$

$$= \min \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} + u_j)\, x_{ij} - \sum_{j=1}^{n} u_j$$

subject to (3) and (4)

This problem reduces to m 0 - 1 knapsack problems and can thus be solved in time proportional to n $\sum_{i=1}^{m} b_i$.

The second relaxation is obtained by dualizing constraints (3) with v $\geq$ 0.

$$Z_{D2}(v) = \min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} v_i \left(\sum_{j=1}^{n} a_{ij}\, x_{ij} - b_i\right)$$

subject to (2) and (4)

$$(LR2_v)$$

$$= \min \sum_{j=1}^{n} \left(\sum_{i=1}^{m} (c_{ij} + v_i\, a_{ij})\, x_{ij}\right) - \sum_{i=1}^{m} v_i\, b_i$$

subject to (2) and (4) .

This relaxation is defined for v $\geq$ 0, which is a necessary condition for $Z_{D2}(v) \le Z$ to hold. Since constraints (2) are generalized upper bound (GUB) constaints, we will call a problem like $(LR2_v)$ a 0 - 1 GUB problem. Such a problem is easily solved in time proportional to nm by determining $\min_i (c_{ij} + v_i a_{ij})$ for each j and setting the associated $x_{ij} = 1$. Remaining x_{ij} are set to zero.

4. Issues

A little thought about using $(LR1_u)$ or $(LR2_v)$ within a branch and bound algorithm for the generalized assignment problem quickly brings to mind a number of issues that need to be resolved. Foremost among these is:

(1) How will we select an appropriate value for u?

A closely related question is:

(2) Can we find a value for u for which $Z_D(u)$ is equal to or nearly equal to Z?

The generalized assignment problem also shows that different Lagrangian relaxations can be devised for the same problem. Comparing $(LR1_u)$ and $(LR2_v)$, we see that the first is harder to solve but might provide better bounds. There is also the question of how either of these relaxations compares with the LP relaxation. This leads us to ask

(3) How can we choose between competing relaxations, i.e., different Lagrangian relaxations and the linear programming relaxation?

Lagrangian relaxations also can be used to provide good feasible solutions. For example, a solution to $(LR2_v)$ will be feasible in the generalized assignment problem unless the "weight" of items assigned to one or more of the "knapsacks" corresponding to constraints (3) exceeds the capacity b_i. If this happens, we could reassign items from overloaded knapsacks to other knapsacks, perhaps using a variant of a bin-packing heuristic, to attempt to achieve primal feasibility. In general we would like to know

(4) How can (LR_u) be used to obtain feasible solutions for (P)? How good are these solutions likely to be?

Finally, we note that the ultimate use of Lagrangian relaxation is for fathoming in a branch and bound algorithm, which leads us to ask:

(5) How can the lower and upper bounding capabilities of the Lagrangian problem be integrated within branch and bound?

The remainder of this paper is organized around these five issues, which arise in any application of Lagrangian relaxation. A separate section is devoted to each one. In some cases (issues (1) and (3)) general theoretical results are available. But more often, the "answers" to the questions we have posed must be extrapolated from computational experience or theoretical results that have been obtained for specific applications.

5. Existing Applications

Table 1 is a compilation of the applications of Lagrangian relaxation of which I am aware. I have not attempted to include algorithms, like those given in [4] and [7], that are described without reference to Lagrangian relaxation, but can be described in terms of Lagrangian relaxation with sufficient insight. Nor have I included references describing applications of the algorithms in Table 1. For example, reference [37] describes a

TABLE 1

APPLICATIONS OF LAGRANGIAN RELAXATION

Problem	Researchers	Lagrangian Problem
TRAVELING SALESMAN		
Symmetric	Held & Karp [27,28] Helbig Hansen and Krarup [26]	Spanning Tree Spanning Tree
Asymmetric	Bazarra & Goode [3]	Spanning Tree
Symmetric	Balas & Christofides [1]	Perfect 2-Matching
Asymmetric	Balas & Christofides [1]	Assignment
SCHEDULING		
n|m Weighted Tardiness	Fisher [15]	Pseudo-Polynomial Dynamic Programming
1 Machine Weight Tardiness	Fisher [18]	Pseudo-Polynomial DP
Power Generation Systems	Muckstadt & Koenig [36]	Pseudo-Polynomial DP
GENERAL IP		
Unbounded Variables	Fisher & Shapiro [16]	Group Problem
Unbounded Variables	Burdet & Johnson [5]	Group Problem
0 - 1 Variables	Etcheberry, et. al. [13]	0 - 1 GUB
LOCATION		
Uncapacitated	Cornuejols, Fisher, & Nemhauser [10] Erlenkotter [11]	0 - 1 VUB 0 - 1 VUB
Capacitated	Geoffrion & McBride [23]	0 - 1 VUB
Databases in Computer Networks	Fisher & Hochbaum [19]	0 - 1 VUB
GENERALIZED ASSIGNMENT		
	Ross & Soland [40]	Knapsack
	Chalmet & Gelders [8]	Knapsack, 0-1 GUB
	Fisher, Jaikumar & Van Wassenhove [21]	Knapsack
SET COVERING--PARTITIONING		
Covering	Etcheberry [12]	0 - 1 GUB
Partitioning	Nemhauser & Weber [38]	Matching

successful application of the Lagrangian relaxation in [10] to a specialized uncapacitated location problem involving data clustering. Finally, the breadth and developing nature of this field makes it certain that other omissions exist. I would be happy to learn of any applications that I have overlooked.

This list speaks for itself in terms of the range of hard problems that have been addressed and the types of embedded structures that have been exploited in Lagrangian problems. Most of these structures are well-known but two require comment. The pseudo-polynomial dynamic programming problems arising in scheduling are similar to the 0 - 1 knapsack problem if we regard the scheduling horizon as to the knapsack size and the set of jobs to be scheduled as the set of items available for packing. The notation VUB stands for "variable upper bound" [41] and denotes a problem structure in which some variables are upper bounded by other 0 - 1 variables. An example of this structure is given in Section 7.

6. Determining u

It is clear that the best choice for u would be an optimal solution to the dual problem

$$Z_D = \max_u Z_D(u) \quad . \qquad\qquad (D)$$

Most schemes for determining u have as their objective finding optimal or near optimal solutions to (D).

Problem (D) has a number of important structural properties that make it feasible to solve. We have assumed that the set $X = \{x | Dx \le e,\ x \ge 0 \text{ and integral}\}$ of feasible solutions for (LR_u) is finite, so we can represent X as $X = \{x^t,\ t = 1, \ldots , T\}$. This allows us to express (D) as the following linear program with many constraints.

$$Z_D = \max w$$

$$w \le cx^t + u\,(Ax^t - b),\ t = 1, \ldots, T \qquad (\bar{D})$$

The LP dual of $(\bar{D})$ is a linear program with many columns.

$$Z_D = \min \sum_{t=1}^{T} \lambda_t\, cx^t$$

$$\sum_{t=1}^{T} \lambda_t\, Ax^t = b \qquad (\bar{P})$$

$$\sum_{t=1}^{T} \lambda_t = 1$$

$$\lambda_t \ge 0,\ t = 1, \ldots, T \quad .$$

Problem $(\bar{P})$ with λ_t required to be integral is equivalent to (P), although $(\bar{P})$ and (LP) generally are not equivalent problems.

Both $(\bar{D})$ and $(\bar{P})$ have been important constructs in the formulation of algorithms for (D). Problem $(\bar{D})$ makes it apparent that $Z_D(u)$ is the lower envelope of a finite family of linear functions. The form of $Z_D(u)$ is shown in Figure 1 for $m = 1$ and $T = 4$. The function $Z_D(u)$ has all the nice properties, like continuity and concavity, that make life easy for a hill-climbing algorithm, except one--differentiability. The function is nondifferentiable at any $\bar{u}$ where $(LR_{\bar{u}})$ has multiple optima. Although it is differentiable almost everywhere, it generally is nondifferentiable at an optimal point.

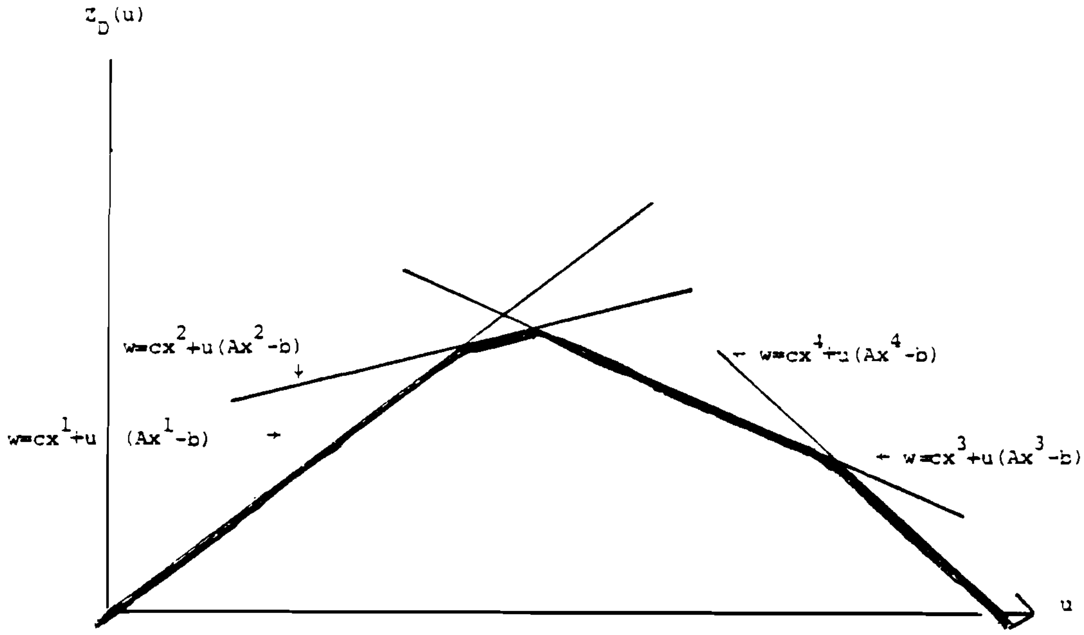


Figure 1

The Form of $Z_D(u)$

An m-vector y is called a subgradient of $Z_D(u)$ at $\bar{u}$ if it satisfies

$$Z_D(u) \le Z_D(\bar{u}) + y\ (u - \bar{u}),\ \text{for all } u \quad .$$

It's apparent that $Z_D(u)$ is subdifferentiable everywhere. The vector $(Ax^t - b)$ is a subgradient at any u for which x^t solves (LR_u). Any other subgradient is a convex combination of these primitive subgradients. With this perspective, the well-known result that u^* and λ^* are optimal for $(\bar{D})$ and $(\bar{P})$ if and only if they are feasible and satisfy a complementary slackness condition can be seen to be equivalent to the obvious fact that u* is optimal in (D) if and only if 0 is a subgradient of $Z_D(u)$ at u*.

Stimulated in large part by applications in Lagrangian relaxation, the field of nondifferentiable optimization using subgradients has recently become an important topic of study in its own right with a large and growing literature. Our review of algorithms for (D) will be brief and limited to the following three approaches that have been popular in Lagrangian relaxation applications: (1) the subgradient method, (2) various versions of the simplex method implemented using column generation techniques, and (3) multiplier adjustment methods. References [17] and [29] contain general discussions on the solution of (D) within the context of Lagrangian relaxation. Reference [2] is a good general source on nondifferentiable optimization.

The subgradient method is a brazen adaptation of the gradient method in which gradients are replaced by subgradients. Given an initial value u^o a sequence $\{u^k\}$ is generated by the rule

$$u^{k+1} = u^k + t_k (Ax^k - b)$$

where x^k is an optimal solution to $(LR_u k)$ and t_k is a positive scalar step size. Because the subgradient method is easy to program and has worked well on many practical problems, it has become the most popular method for (D). There have also been many papers, such as Camerini, et al.,[6], that suggest improvements to the basic relaxation method.

Computational performance and theoretical convergence properties of the subgradient method are discussed in Held, Wolfe and Crowder [29] and their references, and in several references on nondifferentiable optimization, particularly Goffin [25]. The fundamental theoretical result is that $Z_D(u^k) \rightarrow Z_D$ if $t_k \rightarrow 0$

and $\sum_{i=0}^{k} t_i \rightarrow \infty$. The step size used most commonly in practice is

$$t_k = \frac{\lambda_k (Z^* - Z_D (u^k))}{||Ax^k - b||^2}$$

where λ_k is a scalar satisfying $0 < \lambda_k \leq 2$ and Z^* is an upper bound on Z_D, frequently obtained by applying a heuristic to (P). Justification of this formula is given in [29]. Often the sequence λ_k is determined by setting $\lambda_o = 2$ and halving λ_k whenever $Z_D(u)$ has failed to increase in some fixed number of iterations. This rule has performed well empirically, even though it is not guaranteed to satisfy the sufficient condition given above for optimal convergence.

Unless we obtain a u^k for which $Z_D(u^k)$ equals the cost of a known feasible solution, there is no way of proving optimality in the subgradient method. To resolve this difficulty, the method is usually terminated upon reaching an arbitrary iteration limit.

Usually $u^o = 0$ is the most natural choice but in some cases one can do better. The generalized assignment problem is a good example. Assuming $c_{ij} > 0$ for all ij, the solution $x = 0$ is optimal in $(LR1_u)$ for any u satisfying $u_j \leq c_{ij}$ for all i and j. Setting $u_j^o = \min_i c_{ij}$ is thus a natural choice. It is clearly better than $u^o = 0$ and, in fact, maximizes the lower bound over all u for which x = 0 is optimal in $(LR1_u)$.

Another class of algorithms for (D) is based on applying a variant of the simplex method to (P), generating an appropriate entering variable on each iteration by solving $(LR_{\bar{u}})$, where $\bar{u}$ is the current value of the simplex multipliers. Of course, using the primal simplex method with column generation is an approach with a long history [24]. However, this approach is known to converge very slowly and does not produce monotonically increasing lower bounds. These deficiencies have prompted researchers to devise column generation implementations of dual forms of the simplex method, specifically the dual simplex method (Fisher [15]) and the primal-dual simplex method (Fisher, Northup, Shapiro [17]). The primal-dual simplex method can also be modified slightly to make it the method of steepest ascent for (D). Hogan, Marsten and Blankenship [30] and Marsten [33] have had success with an interesting modification of these simplex approaches that they call BOXSTEP. Beginning at given u^o, a sequence $\{u^k\}$ is generated. To obtain u^{k+1} from u^k, we first solve $(\bar{D})$ with the additional requirement that $|u_i - u_i^k| \leq \delta$ for some fixed positive δ. Let $\bar{u}^k$ denote the optimal solution to this problem. If $|\bar{u}_i^k - u_i^k| < \delta$ for all i then $\bar{u}^k$ is optimal in (D). Otherwise set $u^{k+1} = u^k + t_k(\bar{u}^k - u^k)$ where t_k is the scalar that solves

$$\max_t Z_D \; (u^k + t \; (\bar{u}^k - u^k)) \quad .$$

This line search problem is easily solved by Fibonacci methods.

Generally, the simplex-based methods are harder to program and have not performed quite so well computationally as the subgradient method. They should not be counted out, however. Further research could produce attractive variants. We note also that the dual, primal-dual and BOXSTEP methods can all be used in tandem with the subgradient method by initiating them with a point determined by the subgradient method. Using them in this fashion to finish off a dual optimization probably best exploits their comparative advantages.

The third approach, multiplier adjustment methods, are specialized algorithms for (D) that exploit the structure of a particular application. In these methods, a sequence u^k is generated by the rule
$u^{k+1} = u^k + t_k d_k$ where t_k is a positive scalar and d_k is a direction. To determine d_k we define a finite and usually small set of primitive directions S for which it is easy to evaluate the directional derivative of $Z_D(u)$. Usually directions in S involve changes in only one or two multipliers. For directions in S, it should be easy to determine the directional derivative of $Z_D(u)$. Directions in S are scanned in fixed order and d_k is taken to be either the first direction found along which $Z_D(u)$ increases or the direction of steepest ascent within S. The step size t_k can be chosen either to maximize $Z_D(u^k + td_k)$ or to take us to the first point at which the directional derivative changes. If S contains no improving direction we terminate, which, of course, can happen prior to finding an optimal solution to (D).

Successful implementation of primitive-direction ascent for a particular problem requires an artful specification of the set S. S should be manageably small, but still include directions that allow ascent to at least a near optimal solution. Held and Karp [27] experimented with primitive-direction ascent in their early work on the traveling salesman problem. They had limited success using a set S consisting of all positive and negative coordinate vectors. This seemed to discourage other researchers for some time, but recently Erlenkotter [11] devised a multiplier adjustment method for the Lagrangian relaxation of the uncapacitated location problem given in [10] in the case where the number of facilities located is unconstrained. Although discovered independently, Erlenkotter's algorithm is a variation on a method of Bilde and Krarup that was first described in 1967 in a Danish working paper and later published in English as [4]. While there has been no direct comparison, Erlenkotter's method appears to perform considerably better than the subgradient method. Fisher and Hochbaum [19] have experimented with multiplier adjustment for another location problem and found the method to work well, but not quite so well as the subgradient method.

Fisher, Jaikumar, and Van Wassenhove [21] have successfully developed a multiplier adjustment method for the generalized assignment problem in which one multiplier at a time is increased. This method has led to a substantially improved algorithm for the generalized assignment problem.

7. How Good are the Bounds?

The "answer" to this question that is available in the literature is completely problem-specific and largely empirical. Most of the empirical results are summarized in Table 2. Each line of this table corresponds to a paper on a particular application of Lagrangian relaxation and gives the problem type, the source in which the computational experience is given, the number of problems attempted, the percentage of problems for which a u was discovered with $Z_D(u) = Z_D = Z$, and the average value of $Z_D(u^*) \times 100$ divided by the average value of Z, where $Z_D(u^*)$ denotes the largest bound discovered for each problem instance. Except as noted for the generalized assignment problem, all samples included a reasonable number of large problems. In some cases the sample included significantly larger problems than had been previously attempted. Frequently, standard test problems known for their difficulty were included. Table 2 is based on the

TABLE 2

COMPUTATIONAL EXPERIENCE WITH LAGRANGIAN RELAXATION

Problem Type	Source	Number of Problems Solved	Percentage of Problems With $Z_D = Z$	$\frac{\text{Ave. } Z_D(u^*)}{\text{Ave. } Z} \times 100$
TRAVELING SALESMAN				
Symmetric	[28]	18	55.5	99.5
Asymmetric	[3]	8	0.0	97.5
SCHEDULING				
n/m weighted Tardiness	[15]	8	37.5	96.2
1 Machine Weighted Tardiness	[18]	63	49.2	99.6
Power Generation Systems	[36]	15	0.0	98.9
GENERAL IP	[17]	11	0.0	83.2
LOCATION				
Uncapacitated	[10]	33	66.6	99.9
Capacitated	[23]	6	50.0	99.4
Databases in Computer Networks	[19]	29	51.7	95.9
GENERALIZED ASSIGNMENT				
Lagrangian* Relaxation 1	[8]	249**	96.0	99.8
Lagrangian* Relaxation 1	[21]	15	80.0	98.6
Lagrangian* Relaxation 2	[8]	249**	0.0	69.1

*See Section 3.for a definition of Lagrangian relaxations 1 and 2.

**Mostly small problems. The largest had $m = 9$ and $n = 17$.

results reported in each reference for all problems for which complete information was given. Of course. Table 2 gives highly aggregated information, and interested readers are urged to consult the appropriate references.

These results provide overwhelming evidence that the bounds provided by Lagrangian relaxation are extremely sharp. It is natural to ask why Lagrangian bounds are so sharp.
I am aware of only one analytic result that even begins to answer this question. This result was developed by Cornuejols, Fisher and Nemhauser [10] for the K-median problem.

Given n possible facility locations, m markets, and a nonnegative value c_{ij} for serving market i from a facility at location j, the K-median problem asks where K facilities should be located to maximize total value. Let

$$y_j = \begin{cases} 1, & \text{if a facility is placed in location } j \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if market } i \text{ is served from location } j \\ 0, & \text{otherwise} \end{cases} .$$

If $y_j = 0$ we must have $x_{ij} = 0$ for all i. Thus the K-median problem can be formulated as the integer program

$$Z = \max \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{5}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \; i = 1, \ldots, m \tag{6}$$

$$\sum_{j=1}^{n} y_j = K \tag{7}$$

$$0 \le x_{ij} \le y_j \le 1, \quad \text{for all } i \text{ and } j \tag{8}$$

$$x_{ij} \text{ and } y_j \text{ integral, for all } i \text{ and } j \quad . \tag{9}$$

A **Lagrangian** relaxation is obtained by dualizing constraints (6).

$$Z_D(u) = \max \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} u_i \Big(\sum_{j=1}^{m} x_{ij} - 1 \Big)$$

subject to (7), (8) and (9)

$$= \max \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} + u_i)\, x_{ij} - \sum_{i=1}^{m} u_i$$

subject to (7), (8) and (9) .

This problem has the 0 - 1 VUB structure described in Section 4. To solve, we first observe that the VUB constraints (8) and the objective of the Lagrangian problem imply that

$$x_{ij} = \begin{cases} y_j, & \text{if } c_{ij} + u_i \ge 0 \\ 0\ , & \text{otherwise} \quad . \end{cases}$$

Hence, defining $\bar{c}_j = \sum_{i=1}^{m} \max\,(0,\, c_{ij} + u_i)$, optimal y_j's must solve

$$\max \sum_{j=1}^{n} \bar{c}_j\, y_i$$

$$\sum_{j=1}^{n} y_j = K$$

$$y_j = 0 \text{ or } 1,\ j = 1, \ldots , n$$

which is a trivial problem.

Let $Z_D = \min_u Z_D(u)$ and assume $Z_D > 0$. Cornuejols, Fisher and Nemhauser [10] proved that $(Z_D - Z)/Z_D \le \left(\frac{K-1}{K}\right)^K < \frac{1}{e}$ and exhibited examples that show this bound to be the best possible.

This is an interesting first step towards understanding why Lagrangian relaxation has worked well on so many problems. Further study of this type is needed to understand and better exploit the power of Lagrangian relaxation.

8. Selecting Between Competing Relaxations

Two properties are important in evaluating a relaxation: the sharpness of the bounds produced and the amount of computation required to obtain these bounds. Usually selecting a relaxation involves a tradeoff between these two properties; sharper bounds require more time to compute. It is generally difficult to know whether a relaxation with sharper bounds but greater computation time will result in a branch and bound algorithm with better overall performance. However, it is usually possible to at least compare the bounds and computation requirements for different relaxations. This will be demonstrated for the generalized assignment example.

Two Lagrangian relaxations, $(LR1_u)$ and $(LR2_v)$, were defined for this problem. The linear programming relaxation of formulation (1) - (4) provides a third relaxation.

Consider first the computational requirements for each relaxation. We know that solving $(LR1_u)$ requires time bounded by $n \sum_{i=1}^{m} b_i$ and solving $(LR2_v)$ requires time proportional to $n\ m$. From this it would seem that the first relaxation requires greater computation, although it is difficult to know how many times each Lagrangian problem must be solved in optimizing the duals. It is also impossible to know analytically the time required to solve the LP relaxation of (1) - (4).

Reference [8] reports computational times for the three relaxations for examples ranging in size from $m = 4$ and $n = 6$ to $m = 9$ and $n = 17$. The subgradient method was used to optimize the dual problems. On average, the first relaxation required about 50% more computational time than the second. This is much less than would be expected from comparison of worst-case bounds on times to solve Lagrangian problems because the subgradient method converged more quickly for the first relaxation. Solving the LP relaxation required one-fourth of the time for $(LR1_u)$ for small problems but 2.5 times for large problems.

Now consider the relative sharpness of the bounds produced by these relaxations. Let $Z_{D1} = \max_{u} Z_{D1}(u)$, let $Z_{D2} = \max_{v \geq 0} Z_{D2}(V)$, and let Z_{LP}^{GA} denote the optimal value of the LP relaxation of (1) - (4).

A glance at the computational experience reported in the last two lines of Table 2 for the two Lagrangian relaxations strongly suggests that relaxation 1 produces much sharper bounds than relaxation 2. This observation can be verified using an analytic result given by Geoffrion [22]. This result will also allow us to compare Z_{D1} and Z_{D2} with Z_{LP}^{GA}.

The result states that in general $Z_D \geq Z_{LP}$. Conditions are also given for $Z_D = Z_{LP}$. The fact that $Z_D \geq Z_{LP}$ can be established by the following series of relations between optimization problems.

$$\begin{aligned}
Z_D = \max_u & \left\{ \min_x \ cx + u(Ax - b) \right\} \\
& \text{s.t.} \quad Dx \geq e \\
& \qquad\quad x \geq 0 \text{ and integral} \\
\geq \max_u & \left\{ \min_x \ cx + u(Ax - b) \right\} \\
& \text{s.t.} \quad Dx \geq e \\
& \qquad\quad x \geq 0
\end{aligned}$$

$$\begin{aligned}
\text{(By LP duality)} \quad = \max_u \ & \max_{v \geq 0} \ ve - ub \\
& \text{s.t.} \quad vD \leq c + uA
\end{aligned}$$

$$\begin{aligned}
\text{(By LP duality)} \quad = \ & \min_x \quad cx \\
& \text{s.t.} \ A\,x = b \\
& \qquad D\,x \geq e \\
& \qquad\quad x \geq 0 \\
= \ & Z_{LP}
\end{aligned}$$

This logic also reveals a sufficient condition for $Z_D = Z_{LP}$. Namely, $Z_D = Z_{LP}$ whenever $Z_D(u)$ is not increased by removing the integrality restriction on x from the constraints of the Lagrangian problem. Geoffrion [22] calls this the integrality property.

Applying these results to the generalized assignment problem establishes that $Z_{D1} \geq Z_{D2} = Z_{LP}$ since the second Lagrangian relaxation has the integrality property while the first does not.

It should be emphasized that the integrality property is not defined relative to a given problem class but relative to a given integer programming formulation of a problem class. This is an important distinction because a problem often has more than one formulation. The Lagrangian relaxation of the K-median problem given in Section 7 has the integrality property if one takes (P) to be formulation (5) - (9). This fact alone is misleading since there is another formulation of the K-median problem in which constraints (8) are replaced by

$$\sum_{i=1}^{m} x_{ij} \leq my_j, \; j = 1, \ldots, n \qquad (8'a)$$

$$0 \leq x_{ij} \leq 1, \text{ for all } i \text{ and } j \qquad (8'b)$$

$$0 \leq y_j \leq 1, \; j = 1, \ldots, n \quad . \qquad (8'c)$$

This formulation is much more compact than (5) - (9) and is the one used in most LP-based branch and bound algorithms for the K-median problem. The Lagrangian relaxation given previously can be defined equivalently in terms of this formulation but relative to this formulation, it does not have the integrality property. In fact, it is shown in [10] that the Lagrangian bound Z_D and the LP value of (5), (6). (7), (8), (9) are substantially sharper than the LP value of (5), (6), (7), (8') and (9). Others (Williams [45,46] and Mairs, et al [32]) have also noted that there are frequently alternative IP formulations for the same problem that have quite different LP properties.

It is also worth noting that many other successful Lagrangian relaxations (including Held and Karp [27, 28], Etcheberry [12], Etcheberry, et al. [13], and Fisher and Hochbaum [19]) have had the integrality property. For these applications Lagrangian relaxation was successful because the LP relaxation closely approximated (P) and because the method used to optimize (D) (usually the subgradient method) was more powerful then methods available for solving the (generally large) LP relaxation of (P). The important message of these applications is that combinatorial optimization problems frequently can be formulated as a large IP whose LP relaxation closely approximates the IP and can be solved quickly by dual methods. To exploit this fact, future research should be broadly construed to develop methods for solving the large structured LP's arising from combinatorial problems and to understand the properties of combinatorial problems that give rise to good LP approximations. There has already been significant research on methods other than Lagrangian relaxation for exploiting the special structure of LP's derived from combinatorial problems. Schrage [41], Miliotis [34, 35], and Christofides and Whitlock [9] have given clever LP solution methods that exploit certain types of structure that are common in formulationsof combinatorial problems.

9. Feasible Solutions

This section is concerned with using (LR_u) to obtain feasible solutions for (P). It is possible in the course of solving (D) that a solution to (LR_u) will be discovered that is feasible in (P). Because the dualized constraints $Ax = b$ are equalities, this solution is also optimal for (P). If the dualized constraints contain some inequalities, a Lagrangian problem solution can be feasible but nonoptimal for (P). However, it is rare that a feasible solution of either type is discovered. On the other hand, it often happens that a solution to (LR_u) obtained while optimizing (D) will be nearly feasible for (P) and can be made feasible with some judicious tinkering. Such a method might be called a Lagrangian heuristic. After illustrating this approach for the generalized assignment problem and $(LR1_u)$, we will discuss computational experience with Lagrangian heuristics for other problems.

It is convenient to think of the generalized assignment problem as requiring a packing of n items into m knapsacks using each item exactly once. In $(LR1_u)$ the constraints $\sum_{i=1}^{m} x_{ij} = 1$, $j = 1, \ldots , n$ requiring that each item be used exactly once are dualized and may be violated. Let $\bar{x}$ denote an optimal solution to $(LR1_u)$. Partition $N = \{1, \ldots , n\}$ into three sets defined by

$$S_1 = \left\{ j \in J \mid \sum_{i=1}^{m} \bar{x}_{ij} = 0 \right\}$$

$$S_2 = \left\{ j \in J \mid \sum_{i=1}^{m} \bar{x}_{ij} = 1 \right\}$$

$$S_3 = \left\{ j \in J \mid \sum_{i=1}^{m} \bar{x}_{ij} > 1 \right\}$$

The constraints of (P) which are violated by $\bar{x}$ correspond to $j \in S_1 \cup S_3$. We wish to modify $\bar{x}$ so that these constraints are satisfied. This is easy for a $j \in S_3$. Simply remove item j from all but one knapsack. A variety of rules could be used to determine in which knapsack to leave item j. For example, it would be reasonable to choose the knapsack that maximizes $\dfrac{u_i - c_{ij}}{a_{ij}}$.

To complete the construction of a feasible solution it is only necessary to assign items in S_1 to knapsacks. While there is no guarantee that this can be done, the chances of success should be good unless the knapsack constraints are very tight. Many assignment rules are plausible,such as the following one that is motivated by bin packing heuristics. Order items in S_1 by decreasing value of $\sum_{i=1}^{m} a_{ij}$ and place each item in turn into a knapsack with sufficient capacity that maximizes $\dfrac{u_i - c_{ij}}{a_{ij}}$.

Several researchers have reported success using Lagrangian problem solutions obtained during the application of the subgradient method to construct primal feasible solutions. For example, this is easy to do for the K-median problem. Let $\bar{x}$, $\bar{y}$ denote a feasible solution to the

Lagrangian problem defined in Section 7 for the K-median problem. Let $S = \{j | \bar{y}_j = 1\}$ and for each i set $\bar{\bar{x}}_{ij} = 1$ for a j that solves $\max_{j' \varepsilon S} c_{ij'}$. Set $\bar{\bar{x}}_{ij} = 0$ for remaining ij. The solution $\bar{\bar{x}}, \bar{y}$ is feasible and represents the best assignment of x given $\bar{y}$. Cornuejols, Fisher and Nemhauser [10] found that this approach performed as well as the best of several other heuristics they tested.

Fisher [18] reports experience for the problem of sequencing n jobs on one machine to minimize a tardiness function. A Lagrangian solution is a set of start times $\bar{x}_1, \ldots , \bar{x}_n$ for the n jobs that may violate the machine constraints. A primal feasible solution is obtained by sequencing jobs on the machine in order of increasing $\bar{x}_j$ values. This rule was tested on 63 problems. It was applied in conjunction with the subgradient method after an initial feasible solution had been generated by a greedy heuristic. The greedy heuristic found an optimal solution for 18 of the problems. The Lagrangian heuristic found optimal solutions to 21 of the remaining 45 problems. On average the greedy value was 100.4% of the optimal value while the value of the solution produced by the Lagrangian heuristic was 100.16% of the optimal value.

10. Using Lagrangian Relaxation in Branch and Bound

The issues involved in designing a branch and bound algorithm that uses a Lagrangian relaxation are essentially the same as those that arise when a linear programming relaxation is used. Some of these issues are illustrated here for the generalized assignment problem and ($LR1_u$) derived in Section 3.

A natural branching tree for this problem is illustrated in Figure 2. This tree exploits the structure of constraints (2) by selecting a particular index j when branching and requiring exactly one variable in the set x_{ij}, $i = 1, \ldots, m$ to equal 1 along each branch.

A Lagrangian relaxation (presumably $LR1_u$ given the discussion in Section 8) can be used at each node of this tree to obtain lower bounds and feasible solutions.

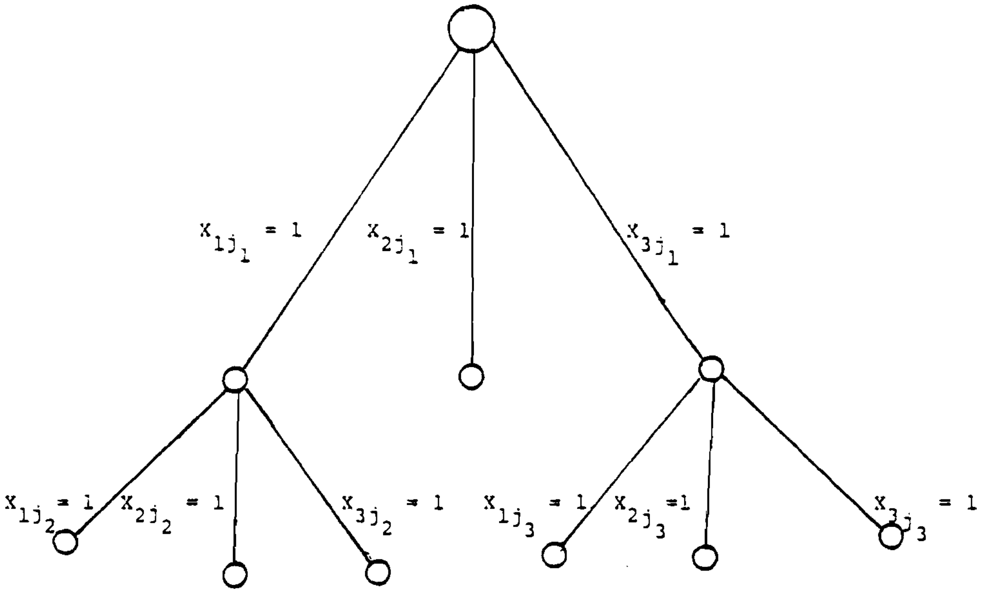


Figure 2

Partial Branching Tree for The Generalized Assignment Problem with $m = 3$.

We note that the Lagrangian problem defined at a particular node of this tree has the same structure as $(LR1_u)$ and is no harder to solve. This is an obvious property that must hold for any application. Sometimes it is desirable to design the branching rules to achieve this property (e.g., Held and Karp [28]).

There are several tactical decisions that must be made in any branch and bound scheme such as which node to explore next and what indices (j_1, j_2 and j_3 in Figure 2) to use in branching. Lagrangian relaxation can be used in making these decisions in much the same way that linear programming would be used. For example, we might choose to branch on an index j for which $u_j(\sum_{i=1}^{m} x_{ij} - 1)$ is large in the current Lagrangian problem solution in order to strengthen the bounds as much as possible.

Finally, we note that the method for optimizing (D) must be carefully integrated into the branch and bound algorithm to avoid doing unnecessary work when (D) is reoptimized at a new node. A common strategy when using the subgradient method is to take u^o equal to the terminal value of u at the previous node. The subgradient method is then run for a fixed number of iterations that depends on the type of node being explored. At the first node a large number of iterations is used. When branching down a small number is used, and when backtracking, an intermediate number.

11. Conclusions and Future Research Directions

Lagrangian relaxation is an important new computational technique in the management scientist's arsenal. This paper has documented a number of successful applications of this technique, and hopefully will inspire other applications. Beside further applications, what opportunities for further research exist in this area? The most obvious is development of more powerful technology for optimizing the nondifferentiable dual function. Nondifferentiable optimization has become an important general research area that surely will continue to grow. One corner of this area that seems to hold great promise for Lagrangian relaxation is the development of multiplier adjustment methods of the type described at the end of section 6. The enormous success that has been obtained with this approach on the uncapacitated location [11] and the generalized assignment problems [21], suggests that it should be tried on other problems. Two other research areas that deserve further attention are the development and analysis of Lagrangian heuristics as described in Section 9 and the analysis (worst-case or probabilistic) of the quality of the bounds produced by Lagrangian relaxation as discussed in Section 7 and [10].

12. Acknowledgement

This paper was supported in part by NSF Grant ENG-7826500 to the University of Pennsylvania.

REFERENCES

[1] Balas, E., & Christofides, N. Talk presented at the Ninth International Symposium on Mathematical Programming, Budapest, August, 1976.

[2] Balinski, M. L., & Wolfe, P., Eds. Math. Prog. Study 3: Nondifferentiable Optimization (November 1975).

[3] Bazaraa, M. S., & Goode, J. J. "The Traveling Salesman Problem: A Duality Approach." Math. Prog. 13 (1977), pp. 221-237.

[4] Bilde, O. & Krarup, J. "Sharp Lower Bounds and Efficient Algorithms for the Simple Plant Location Problem," Annals of Discrete Mathematics 1 (1977), 79-97.

[5] Burdet, C. A., & Johnson, E. L. "A Subadditive Approach to Solve Linear Integer Programs." Annals of Discrete Mathematics 1 (1977) 117-143.

[6] Camerini, P. M., L.Fratta and F. Maffioli, "On Improving Relaxation Methods by Modified Gradient Techniques," Math. Progr. Study, Vol. 3 (1975) 26-34.

[7] Camerini, P. M., & Maffioli, F. "Heuristically Guided Algorithms for K-Parity Matroid Problems," Discrete Mathematics (1978), 103-116.

[8] Chalmet, L. G., & Gelders, L. F. "Lagrangian Relaxations for a Generalized Assignment-Type Problem." Proceedings of the Second European Congress on Operations Research, North-Holland (1976) 103-109.

[9] N. Christofides & C. Whitlock. "An LP-based TSP Algorithm," Imperial College Report 78-79 (1978).

[10] Cornuejols, G., Fisher, M. L., & Nemhauser, G. L. "Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms." Management Science, 23 (1977) 789-810.

[11] Erlenkotter, D. "A Dual-Based Procedure for Uncapacitated Facility Location." Operations Research, Vol. 26, No. 1, (1978) 992-1009.

[12] Etcheberry, J. "The Set-Covering Problem: A New Implicit Enumeration Algorithm." Operations Research 25 (September-October 1977), 760-772.

[13] Etcheberry, J., Conca, C., & Stacchetti, E. "An Implicit Enumeration Approach for Integer Programming Using Subgradient Optimization." Pub. No. 78/04/c, Universidad de Chile, March 1978.

[14] Everett, H. "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources." Operations Research, 11 (1963), 399-417.

[15] Fisher, M. L. "Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I." Operations Research, 21 (1973), 1114-1127.

[16] Fisher, M. L., & Shapiro, J. F. "Constructive Duality in Integer Programming." SIAM Journal on Applied Mathematics, 27 (1974), 31-52.

[17] Fisher, M. L., Northup, W. D., & Shapiro, J. F. "Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience." Mathematical Programming Study, 3 (1975), 56-94.

[18] Fisher, M. L. "A Dual Algorithm for the One-Machine Scheduling Problem." Math. Prog., 11 (1976), 229-251.

[19] Fisher, M. L., & Hochbaum, D. S. "Database Location in Computer Networks." forthcoming in JACM.

[20] Fisher, M. L., Nemhauser, G. L., & Wolsey, L. A. "An Analysis of Approximations for Finding a Maximum Weight Hamiltonian Circuit." Operations Research, Vol. 27, No. 4 (1979) 799-809.

[21] Fisher, M. L., R. Jaikumar, and L. Van Wassenhove, "A Multiplier Adjustment Method for the Generalized Assignment Problem," Decision Sciences Working Paper, University of Pennsylvania, May, 1980.

[22] Geoffrion, A. M. "Lagrangian Relaxation and its Uses in Integer Programming." Mathematical Programming Study, 2 (1974), 82-114.

[23] Geoffrion, A. M., & McBride, R. "Lagrangian Relaxation Applied to Capacitated Facility Location Problems." AIIE Transactions, 10 (1978), 40-47.

[24] Gilmore, P. C.,& Gomory, R. E. "A Linear Programming Approach to the Cutting-Stock Problem, Part II." Operations Research, 11 (1963), 863-888.

[25] Goffin, J. L. "On the Convergence Rates of Subgradient Optimization Methods." Math. Prog., 13 (1977), 329-347.

[26] Helbig Hansen, K., and J. Krarup, "Improvements of the Held-Karp Algorithm for the Symmetric Traveling-Salesman Problem," Math. Prog., 7 (1974) 87-96.

[27] Held, M., & Karp, R. M. "The Traveling Salesman Problem and Minimum Spanning Trees." Operations Research, 18 (1970), 1138-1162.

[28] Held, M., & Karp, R. M. "The Traveling Salesman Problem and Mimimum Spanning Trees: Part II." Mathematical Programming, 1 (1971), 6-25.

[29] Held, M., Wolfe, P., & Crowder, H. D. "Validation of Subgradient Optimization." Mathematical Programming, 6 (1974), 62-88.

[30] Hogan, W. W., Marsten, R. E., & Blankenship, J. W. "The Boxstep Method for Large Scale Optimization." Operations Research, 23 (1975), 3.

[31] Lorie, J., & Savage, L. J "Three Problems in Capital Rationing." Journal of Business (1955), pp. 229-239.

[32] Mairs, T. G., Wakefield, G. W., Johnson, E. L., & Spielberg, K. "On a Production Allocation and Distribution Problem." Management Science, Vol. 24, No. 15, (1978) 1622-1630.

[33] Marsten, R. "The Use of the Boxstep Method in Discrete Optimization," Math. Progr. Study, 3 (1975) 127-144.

[34] Miliotis, T. "Integer Programming Approaches to the Travelling Salesman Problem." Math. Prog., 10 (1976), 367-378.

[35] Miliotis, T. "An All-Integer Arithmetic LP-Cutting Planes Code Applied to the Travelling Saleman Problem." London School of Economics Working Paper, 1976.

[36] Muckstadt, J., & Koenig, S. A. "An Application of Lagrangian Relaxation to Scheduling in Power Generation Systems." Operations Research, 25 (May-June 1977), 387-403.

[37] Mulvey, J. and H. Crowder, "Cluster Analysis: An Application of Lagrangian Relaxation," Man. Science, 25 (1979) 329-340.

[38] Nemhauser, G. L., & Weber, G. "Optimal Set Partitioning, Matchings and Lagrangian Duality." Talk delivered at the New York ORSA/TIMS Meeting, May 3, 1978.

[39] Poljak, B. T. "A General Method for Solving Extremum Problems." Soviet Mathematics Doklady, 8 (1967), 593-597.

[40] Ross, G. T., & Soland, R. M. "A Branch and Bound Algorithm for the Generalized Assignment Problem." Mathematical Programming, 8 (1975), 91-103.

[41] Schrage, L. "Implicit Representation of Variable Upper Bounds in Linear Programming." Math. Prog. Studies, 4 (1975), 118-132.

[42] Shapiro, J. F. "Generalized Lagrange Multipliers in Integer Programming." Operations Research, 19 (1971), 68-76.

[43] Shapiro, J. F. "A Survey of Lagrangian Techniques for Discrete Optimization." Annals of Discrete Mathematics 5 (1979), 113-138. 1977.

[44] Shapiro, J. F., Mathematical Programming: Structures and Algorithms, Wiley, 1979.

[45] Williams, H. P. "Experiments in the Formulation of Integer Programming Problems." Math. Prog. Study, 2 (December 1974), 180-197.

[46] Williams, H. P. "The Reformulation of Two Mixed Integer Programming Problems." Math. Prog., 14 (1975) 325-331.

AN ITERATIVE LINEAR PROGRAMMING ALGORITHM BASED ON THE MODIFIED LAGRANGIAN[*]

E.G. Gol'shtein

Central Economic Mathematical Institute
USSR Academy of Sciences
Moscow

Current LP solution algorithms are of two types: *finite* — such as the simplex method — and *iterative* — which after a finite number of iterations give only an approximate solution. The main shortcoming of iterative methods to date is their slow rate of convergence. This paper describes an iterative LP algorithm which seems to have a satisfactory practical convergence rate. Naturally, the ultimate conclusion regarding its computational efficiency can be reached only after its widespread use in practice.

[*] This algorithm has been developed at CEMI by E.P. Borisova, N.A. Sokolov and N.V. Tretyakov in conjunction with the author.

1. INTRODUCTION

Economic models developed to describe the processes of economic activity on various levels involve many problems concerning choice of an optimal decision from amongst possible alternatives. Such problems involve a wide range of mathematical concepts amongst which are static and dynamical formulations, continuous and discrete variables, the constraints of simple and very complicated structures, and stochastic and deterministic approaches. Nevertheless, in spite of all these complications, practical problems are usually given a general formulation which is linear. To a large extent this is due to our ignorance regarding the mechanisms of economic processes as well as to difficulties in obtaining reliable data. In any case *linear programming* (LP) remains one of the most important practical techniques with which to treat decision problems.

The LP algorithms of today look rather powerful and sophisticated as a result of the widespread experience of many research workers.

Basically, there are two types of LP methods: *finite* and *iterative*. Finite methods provide in principle the possibility of finding an *exact* solution of the problem (to a specified machine precision) after a finite number of operations, while generally speaking, *any* finite number of operations by an iterative method gives only an *approximate* solution. The typical--and most famous--finite method is the *simplex method*, which is the foundation of most modern LP algorithms. The product form of the simplex method, together with special computational schemes involving reinversions, the rules for choosing pivot elements, prescaling of the initial data, and numerous additional procedures, are presently used in all the commercial LP packages for solving sparse large-scale problems. The fundamental role of the simplex method in LP packages is due to an advanced level of computational efficiency reached after the thirty odd years of its algorithmic development. However, some shortcomings of this highly popular method are well known. They are as follows: numerical instability, inconsistency in, and complexity of, schemes for avoiding ill-conditioned bases and reducing the data representing the inverse matrix, and awkwardness in taking the specific structure of a problem into account.

Many attempts have been made to construct an efficient LP algorithm based on ideas different from the simplex method, in particular by using an iterative method. It is worth noting that algorithmic implementations of these iterative LP methods often do not require the computation of the inverse matrix, do allow compact representation and handy transformation of data, and are numerically stable--i.e., they obviate the shortcomings mentioned above. Why then are iterative algorithms not widely used for practical LP problems? The reason lies in the very slow convergence of all known iterative algorithms; this is their main shortcoming. This report describes an iterative LP algorithm which seems to have a satisfactory practical rate of convergence. Naturally the ultimate conclusion concerning the efficiency of the algorithm can be reached only after obtaining widespread experience in practical use.

The following research workers of CEMI have taken part in developing this algorithm along with the author: E.P. Borisova, N.A. Sokolov, N.V. Tretyakov.

2. PROBLEM FORMULATION AND ALGORITHM OUTLINE

We consider the general LP problem in *canonical form*, i.e. in the form

$$L(x) = \sum_{j=1}^{n} c_j x_j \to \max \; ; \qquad \sum_{j=1}^{n} a_{ij} x_j = b_i , \quad i = 1,2,\dots,m \tag{1}$$

$$x_j \geq 0, \ \forall j \quad .$$

The algorithm is based on using the simplest modified Lagrangian of problem (1):

$$\begin{aligned} F_\alpha(x,y) &= \sum_{j=1}^{n} c_j x_j^j + \sum_{i=1}^{m} y_i u_i(x) - \tfrac{1}{2}\sum_{i=1}^{m} \alpha_i u_i^2(x) \\ &= \sum_{i=1}^{m} b_i y_i + \sum_{j=1}^{n} x_j p_j(y) - \tfrac{1}{2}\sum_{i=1}^{n} \alpha_i u_i^2(x) , \end{aligned} \tag{2}$$

where $u_i(x) = b_i - \sum_{j=1}^{n} a_{ij}x_j$, $i = 1,2,\ldots,m$,

$$p_j(y) = c_j - \sum_{i=1}^{m} a_{ij}y_i, \quad j = 1,2,\ldots,n,$$

$$\alpha = (\alpha_1,\alpha_2,\ldots,\alpha_m) \text{ is a penalty vector, } \alpha_i > 0 \ \forall i$$

$$x = (x_1,x_2,\ldots,x_n), \quad y = (y_1,y_2,\ldots,y_m) \quad .$$

The values $u_i(x)$ and $p_j^+(y) = \max\{0,p_j(y)\}$ are called *residuals* of the corresponding contstraints of problem (1) and of the *dual* problem

$$(3) \qquad \tilde{L}(y) = \sum_{i=1}^{m} b_iy_i \to \min, \quad \sum_{i=1}^{m} a_{ij}y_i \geq c_j, \quad j = 1,2,\ldots n.$$

The vectors $u(x) = (u_1(x),u_2(x),\ldots,u_m(x))$ and $p^+(y) = (p_1^+(y),P_2^+(y),\ldots,p_n^+(y))$ are said to be the *residual vectors* of the primal and dual problems respectively.

The backbone of the algorithm is the well-studied (see [1 - 4]) dual method based on the modified Lagrangian (2). In this method the recursions

$$(4) \qquad x^{s+1} \cong \operatorname*{argmax}_{x \geq 0} F_\alpha(x,y^s), \quad y^{s+1} = y^s - \alpha_0 u(x^{s+1}), \quad s = 1,2,\ldots,$$

with $\alpha = (\alpha_0,\alpha_0,\ldots,\alpha_0)$, are used to construct the sequences $\{x^s\}$ and $\{y^s\}$, the first converging to a solution of (1) and the second converging to a solution of (3).

The implementation of the scheme (4) involves a number of questions such as the following.

Which optimization method should be used to determine x^{s+1} for the fixed $y = y^s$?

What accuracy is required to solve the "auxiliary" problem of maximizing $F_\alpha(x,y^s)$ over the positive orthant $x \geq 0$?

May one use the vector α with identical components as in (4), or should these components be different?

Should the vector α be changed during the process of computation and if so how?

The description of the algorithm presented below answers these, and some other, questions. From the outset it is worth noticing that for algorithm efficiency we must use penalty vectors with different components which must be changed from one iteration to another as the results of current computation. This requirement in particular distinguishes the present algorithm from earlier implementations of (4) (see [5]).

When solving (1) by means of the suggested algorithm three sequences are constructed recursively, namely $x^{k_s} \in E^n_+$, $y^s \in E^m$, $\alpha^s \in \text{int}\ E^m_+$ (we use the notation E^t, E^t_+ and int E^t_+ respectively for the t-dimensional Euclidean space, the positive orthant of E^t and the interior of E^t_+).

The vector $x^{k_{s+1}}$ is determined as the result of approximate solution of the *auxiliary* problem

$$(5) \qquad F_{\alpha^s}(x, y^s) \to \max, \qquad x \in E^n_+$$

with the starting point x^{k_s} which has been found at the previous iteration.

The vector y^{s+1} is computed by the formula

$$(6) \qquad y_i^{s+1} = y_i^s - \alpha_i^s\, h_s u_i(x^{k_{s+1}}), \qquad i = 1,2,\dots,m$$

where the parameter $h_s \in [0,1]$ is chosen to depend on the solution process of the auxiliary problem (5).

The penalty vector α is recomputed according to the rule

$$\alpha^{s+1} = \phi(x^{k_s}, x^{k_{s+1}}, y^{s+1}, \alpha^s)$$

where ϑ is a certain vector-function whose choice substantially influences the efficiency of the algorithm.

Consider now the implementation of the scheme (5-7) in more detail.

3. AUXILIARY PROBLEM OPTIMIZATION

The *alternating coordinate direction* method (which is often called *Seidel's* optimization method) was chosen for solving the auxiliary problem (5). This choice was made for the following reasons. First, the numerical implementation of the alternating coordinate direction method is very simple *and* fits naturally with data processing column by column (as used with the simplex method) --an important feature for large-scale LP problems (1) with n>>m. Secondly, the computational trials show that for the case of problem (5) this method is not much worse than methods (such as the conjugate gradient method) which are more efficient in general. A single iteration of the alternating coordinate direction method enables us to obtain the vector x^{t+1} from x^t by solving n one-dimensional problems involving optimization of the function (2) in coordinates $x_1, x_2, \ldots, x_n$ with fixed $y = y^s$, $\alpha = \alpha^s$. The solution of each problem may be computed from the simple recursion:

$$x_j^{t+1} = \max\{0, x_j^t + [p_j(y^s) + \sum_{i=1}^{m} \alpha_i^s a_{ij} u_i(x_1^{t+1}, \ldots, x_{j-1}^{t+1}, x_j^t, \ldots, x_n^t)] / \sum_{i=1}^{m} \alpha_i^s a_{ij}^2\}$$

where $t = k_s + \ell$ and ℓ is the current iteration number of the coordinate direction method used in solving (5).

Let ℓ_s be an integer such that $k_{s+1} = k_s + \ell_s$, that is $x^{k_s+\ell_s}$ is accepted to be $x^{k_{s+1}}$, an approximate solution of (5). The method of determining ℓ_s, which is of great importance for the algorithms's efficiency, is based on using two criteria: A and/or B.

Criterion A

$$\bar{p}^{+}(\tilde{y}^{s+1}) \qquad\qquad \leq c'_A\bar{u}(x^t)$$

$$\frac{2|F_0(x^t,\tilde{y}^{s+1}) - \tilde{\mathcal{L}}(\tilde{y}^{s+1})|}{|F_0(x^t,y^{s+1}) + \tilde{\mathcal{L}}(\tilde{y}^{s+1})|} \leq c''_A\bar{u}(x^t)$$

where: $\tilde{y}^{s+1}$ is derived from y^s according to (6), with $h_s = 1$ and $x^{k_{s+1}}$ replaced by x^t,

$$\bar{p}^{+} = \sum_{j=1}^{n} p_j/n \max\{i,c_j\} \quad ,$$

$$\bar{u} = \sum_{j=1}^{m} |u_i|/m \max\{i,|b_i|\} \quad ,$$

$$F_0(x,y) = F_\alpha(x,y) \quad , \quad \text{with } \alpha = (0,0,\ldots,0)$$

and c'_A and c''_A are specified positive numbers.

Criterion A stops the solution process for (5) when the relative average residual in the constraints of the dual problem and the relative difference between the objective functions of the perturbed primal and dual problems become comparable with the relative average residual in the constraints of the primal problem. Notice that when $t \to \infty$ the left-hand sides of the inequalities in A tend to zero, while $\bar{u}(x^t)$ converges to a positive number, since y^s is not a solution of (3).

Criterion B

$$\Delta\bar{u}(x^t) \leq c_B\bar{u}(x^t) \quad , \qquad \Delta\bar{u}(x^t) \leq \Delta\bar{u}(x^{t-1}) \quad ,$$

where $\Delta\bar{u}(x^t) = \sum_{i=1}^{m} |u_i(x^t) - u_i(x^{t-1})| \;/\; m \cdot \max\{1,|b_i|\}$

and c_B is a specified positive number.

Criterion B stops the solution process for (5) when the vector $u(x^t)$ which determines the direction for adjusting the vector y^s, becomes stable.

To avoid too many iterations in solving the auxiliary problems, the algorithm implimentation is also provided with an iteration count "cut-off" $\ell_{max} = \ell_{max}(n)$, which depends on the dimension n of the vector x.

The number ℓ_s of alternating coordinate direction method iterations performed to find an approximate solution of the auxiliary problem (5) is ℓ the minimal number of iterations after which at least one of the criteria A or B holds if $\ell < \ell_{max}$; otherwise $\ell_s = \ell_{max}$.

4. PENALTY VECTOR UPDATE

The details of the method for updating the penalty vector α, given in general form by (7), is also critical for efficiency of the suggested algorithm.

Set

$$u^s = u(x^s),\ \bar{u}^s = \bar{u}(x^s),\ p^s = p^+(y^s),\ \bar{p}^s = \bar{p}^+(y^s),\quad s = 1,2,\ldots$$

After termination of the auxiliary problem (5) solution process, we transform α^s into α^{s+1} according to the following formulae:

$$\beta_i^s = \alpha_i^s\, \varphi\left(\frac{|u_i^{s+1}|}{\bar{u}^{s+1}}\right), \qquad i = 1,2,\ldots,m, \tag{8}$$

$$\gamma_i^s = \frac{\beta_i^s \sum_{i=1}^{m} \alpha_i^s}{\sum_{i=1}^{m} \beta_i^s}\, \eta_s\, \chi\left(\frac{\bar{u}^{s+1}}{\bar{p}^{s+1}}\right) \psi\left(\frac{\bar{u}^{s+1}}{\bar{u}^s}\right), \qquad i = 1,2,\ldots,m \tag{9}$$

$$\alpha_i^{s+1} = \Pi_{[c_1,c_2]}(\gamma_i^s), \qquad i = 1,2,\ldots,m\ . \tag{10}$$

The function ϕ in (8) changes the components of the penalty vector proportional to the relative residuals in the constraints of the primal problem.

The role of formula (9) is to change the norm of the penalty vector. The factor $\eta_s \in [0,1]$ decreases the norm of α when too many iterations are required in solving the auxiliary problem (5) to the accuracy determined by the criteria A or B. Thus

$$\eta_s \begin{cases} = 1 \quad , & \text{if } \ell_s < \ell_{\max} \quad , \\ \in (0,1) \quad , & \text{if } \ell_s = \ell_{\max} \quad . \end{cases}$$

The functions χ and ψ change the norm of the penalty vector in relation to, respectively, the ratio of the current average relative residuals of the primal and dual problems, and the ratio of the average relative residuals of the primal problem provided by two successive iterations. Finally, the penalty coefficients are projected onto the closed interval $[c_1, c_2]$ denoted by the projection operator Π in (10), the positive numbers c_1 and c_2 being the minimal and maximal admissible values of α_i^s respectively.

The functions involved in (8), (9) were chosen as follows:

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 1 + k_\phi \cdot (t - 1) & 1 \leq t \leq m_\phi \\ \phi(m_\phi) & t > m_\phi \end{cases}$$

$$\chi(t) = \begin{cases} 1 + k_\chi \cdot (t - 1) & 1 \leq t \leq m_\chi \\ \chi(m_\chi) & t > m_\chi \end{cases}$$

$$\psi(t) = \begin{cases} 1 + k_\psi \cdot (t - 1) & 1 \leq t \leq m_\psi \\ \psi(m_\psi) & t > m_\psi \end{cases}$$

where

$$k_\phi,\ k_\chi,\ k_\psi \in (0,1) \quad , \qquad m_\phi,\ m_\chi,\ m_\psi > 1 \quad .$$

5. CURRENT DUAL VECTOR UPDATE

To complete the description of the algorithm implementation, a few comments are in order concerning the formula (6) for updating y^s to yield y^{s+1}. The parameter h_s entering (6) is determined according to the conditions at termination of the auxiliary problem (5) solution process. Namely, we set $h_s = 1$ if either criterion A or B is satisfied at termination (i.e., if $\ell_s < \ell_{max}$) and set $h_s = h \in (0,1)$ otherwise (i.e. when $\ell_s = \ell_{max}$). Thus the parameter h_s decreases the step-size in the direction $u(x^{k_{s+1}})$ when the solution accuracy for (5) is not high enough.

6. COMPUTATIONAL EXPERIENCE

Next we present the results of some trial computational experience with the suggested algorithm.

For all the test problems starting values of x,y and α were taken as follows:

$$x^1 = 0 \ , \quad y^1 = 0 \ , \qquad \alpha^1 = (\bar{\alpha},\bar{\alpha},\ldots,\bar{\alpha})$$

$$\bar{\alpha} = 1 \ / \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}| \ , \qquad (k_1 = 1) \quad .$$

A preliminary normalization of the test problems in the form (1) was also used. Basically it consisted of transforming each problem in the form (1) into an equivalent problem in the same form but having identical averages of the coefficients $|a_{ij}|$, $|b_i|$ and $|c_j|$.

Table 1 summarizes the results of solving 5 practical LP problems of the size given in the first column. Each of the next five columns of Table 1 presents the computational effort required for solving the problems from the initial values to within an accuracy of ε%, the values of ε being indicated in the upper positions of each column.

n × m	10 - 15%	5 - 8%	3 - 5%	2 - 3%	1 - 1.5%	Simplex Iterations
18 × 11	5(5)	14(12)	14(12)	15(13)	21(15)	12
59 × 13	13(7)	18(10)	31(15)	31(15)	44(17)	20
88 × 33	29(11)	29(11)	35(12)	35(12)	57(15)	33
113 × 83	90(23)	90(23)	97(24)	97(24)	103(25)	121
352 × 166	383(155)	457(170)	486(176)	490(178)	578(199)	684

Table 1. Computational Results for Five Practical LP Problems

The accuracy of the solutions has been estimated as follows:

$$\varepsilon = \max\ \{\varepsilon_1, \varepsilon_2, \varepsilon_3\}$$

$$\varepsilon_1 = \frac{|\ c'x^{k_s} - b'y^s\ |}{|\ c'x^{k_s} + b'y^s\ |} \cdot 200$$

$$\varepsilon_2 = 100 \cdot \max_i (u_i(x^{k_s}) \ /\ \max\{1,|b_i|\})$$

$$\varepsilon_3 = 100 \cdot \max_j (p_j^+(y^s) \ /\ \max\{1,|c_j|\} \quad .$$

The computational effort required to solve a test problem to the specified accuracy is presented in Table 1 in the intersection of the corresponding column and row. It is measured by the number k_s-1 of iterations required for the determination of x^{k_s} using the alternating coordinate direction method. The number s-1 of updates of the vector y is given in brackets. The last column of Table 1 contains the number of iterations required to solve the same test problems using a modern version of the revised simplex algorithm in product form. As is seen by inspection of Table 1 the suggested algorithm enables us to find sufficiently accurate solutions of the given problems in a number of iterations comparable to that required by the simplex method. (Note that an iteration of the simplex method is more complicated than one of our solutions of the given problems in a number of iterations comparable to that required by the simplex method. (Note that an iteration of the simplex method is more complicated than one of our algorithm.)

It is well known that the simplex method has worst-case exponential complexity. This arises from the fact that for certain LP problems it must look through all, or almost all, vertices of the feasible polyhedron. It is thus quite natural to try using the suggested iterative algorithm to solve such "difficult" LP problems.

In view of this a special family of LP problems depending on a positive integer parameter m has been considered. The problem corresponding to each fixed m involves 2m nonnegative variables subject to m equality constraints and its feasible polyhedron has 2^m vertices. The problems considered have the property that starting from the natural basis the simplex method will look through all feasible vertices. The results of applying the new algorithm to some problems of this family are given in Table 2 in the same format as in Table 1.

m	10 - 15%	5 - 8%	3 - 5%	2 - 3%	1 - 1.5%	Simplex Iterations
8	351(76)	351(76)	363(77)	363(77)	363(77)	256
10	687(84)	695(85)	695(85)	695(85)	695(85)	1024
12	594(116)	687(147)	764(156)	774(157)	774(157)	4096
15	2451(437)	2536(444)	2541(446)	3466(556)	>4600	32768

Table 2. Computational Results for Four Simplex Method Worst-Case Problems

7. CONCLUSION

In conclusion a few words should be said about various possibilities for using the suggested iterative algorithm in practice.

First of all the algorithm enables us to get approximate solutions of practical LP problems in reasonable time using an extremely small amount of computer memory.

Further, the new iterative algorithm is rather suitable for use *together with* the simplex method as an *initial* solution process. After acheiving a certain solution accuracy a simplex basis, close to the optimal one,may be constructed using information from the approximate primal and dual solutions obtained by the iterative algorithm; this basis is then improved by the simplex method. Computational experience shows that the iterative stage of the process should be performed with a rather low accuracy (no more than about 10 - 15%,as defined for the tables of the previous section) since even this small amount of preliminary work appears to reduce the number of simplex iterations by a factor of ten.

Finally, one could try to use the iterative algorithm for helping the simplex method out of the neighbourhood of a "bad" basis, but no computational experience with this idea has been obtained as yet.

REFERENCES

[1] Polyak, B.T. and N.V. Tretyakov. On an iterative method of linear programming and its economic interpretation. *Economics and Math. Methods* 8.5(1972) 740-751. (In Russian.)

[2] Rockafellar, R.T. The multiplier method of Hestenes and Powell applied to convex programming. *J. Optim. Theory and Appl.* 12.6(1973) 555-562.

[3] Golshtein, E.G. and N.V. Tretyakov. The gradient method of minimization and algorithms of convex programming based on modified Lagrangian functions. *Economics and Math. Methods.* 11.4(1975) 730-742. (In Russian.)

[4] Kort, B.W. and D.P. Bertsekas. Combined primal-dual and penalty methods for convex programming. *SIAM J. Control and Optim.* 14.2(1976).

[5] Belov, E.N. Solving linear and nonlinear programming problems by means of the conjugate gradient method together with the penalty prices method. In: *Routines and Algorithms.* 66(1976) CEMI,Acad. of Sci. of the USSR. (In Russian.)

EXPERIMENTS WITH THE REDUCED GRADIENT METHOD FOR GENERAL AND DYNAMIC LINEAR PROGRAMMING*

Markku Kallio
System and Decision Sciences, IIASA

William Orchard-Hays
*Energy Systems Program, IIASA***

This article deals with variations of the reduced gradient method for general and dynamic linear programming. Such methods generate a monotonically improving sequence of feasible solutions; examples are the simplex method and the standard reduced gradient method. A class of these methods and their convergence have been discussed in a recent article by Kallio and Porteus.

A version of these methods has been implemented in the SESAME system. This version resembles the standard reduced gradient method except that only a subset of nonbasic variables to be changed is considered at each iteration. We have tried out several modifications of this basic version, experimenting with moderate sized nonstructured as well as dynamic problems. Compared with the simplex method, the overall performance of these variants appears to be about equal in the case of linear programs with no particular structure.

For dynamic LP we have obtained some encouraging results. Although we have been able to experiment with only a few problems, it appears that using a specially defined starting basis and an initial nonbasic solution can lead to considerable savings; in one case, the number of iterations required by the reduced gradient method was reduced by a factor of 8. This starting basis is chosen so that its columns are also likely to appear in an optimal basis. For the initial solution, available information, such as current level of activities in real life, may be employed.

No fair comparison was made for dynamic LP between the simplex method and the reduced gradient method. However, our starting basis may be used also in the simplex method, and therefore the results obtained may be employed immediately in the simplex method as well, provided that an option for obtaining a vertex solution from a nonbasic starting solution is available.

*The authors wish to thank an anonymous reader for beneficial comments and many detailed suggestions which have significantly improved our paper.

**Currently at:
Energy Information Administration
U.S. Department of Energy, Washington, D.C.

1. Introduction

Consider the linear program (LP):

find $x \in R^n$ to

(LP1) maximize cx

(LP2) subject to $Ax = b$

(LP3) $0 \leq x \leq u$,

where $c, u \in R^n$, $b \in R^m$, and $A \in R^{m\times n}$ is of full row rank. For solving (LP) we shall consider methods, which can be characerized as follows: Like the simplex method [1], these methods move from one feasible solution to another at each iteration, thereby improving the objective function. Each feasible solution is also associated with a basis. However, this feasible solution need not be an extreme point and the basic solution corresponding to the associated basis need not be feasible. Nevertheless, as shown in [2], an optimal solution, if one exists, can be found in a finite number of iteration (under nondegeneracy).

In the following, we shall first review this class of methods as presented in [2]. Thereafter, we discuss an implementation of such methods in the SESAME system, an interactive mathematical programming system developed by Orchard-Hays [7]. In the last two sections we shall report experiments which we carried out both for nonstructured and for dynamic linear programs (LP).

2. The Class of Methods

We shall now review the methods in consideration as presented in [2]. We call x a system solution if it satisfies (LP2), a homogeneous solution if it satisfies $Ax = 0$, and a feasible solution if it satisfies (LP2) and (LP3). If x is feasible and z is a homogeneous solution, then $x + \theta z$ is feasible as long as $0 \leq x + \theta z \leq u$, for all $\theta \in R$. As θ increases, the objective function if and only if $cz > 0$. The simplex method chooses as z one of the homogeneous solutions corresponding to increasing the value of a nonbasic variable such that cz, the reduced cost, is positive. The methods considered here may choose as z a linear combination of such vectors, rather than just one. In particular, the direction may be chosen according to the reduced gradient method, (e.g. [10]). As in the simplex method, a new feasible solution is found by increasing θ (and the objective function) as much as possible without losing feasibility.

The Admissible Directions

Before stating the method, we shall discuss how an admissible direction is constructed. Let β denote the set of basic indices (indices for basic variables), and let α and γ be sets of variables at their lower and upper bounds at x, respectively; i.e.

$$\begin{aligned} \alpha &= \alpha(x) = \{i \mid x_i = 0\} \quad \text{and} \\ \gamma &= \gamma(x) = \{i \mid x_i = u_i\} \quad . \end{aligned} \tag{1}$$

In the simplex method, all nonbasic variables would be in $\alpha \cup \gamma$, but this is not necessarily the case here. For convenience, assume that the variables have been ordered so that $\beta = \{1,2,\ldots,m\}$. Let B be the corresponding basis matrix, and let a^j denote the j^{th} column of A. For each nonbasic variable $j \in \bar{\beta}$ (the complement of β) define a column vector $z^j \in R^n$ componentwise as follows:

$$z_i^j = \begin{cases} -(B^{-1}a^j)_i & \text{if } i \in \beta \ , \\ 1 & \text{if } i \in \bar{\beta} \text{ and } i = j \ , \\ 0 & \text{otherwise} \ . \end{cases} \tag{2}$$

Clearly, z^j is a homogeneous solution, since $Az^j=B(-B^{-1}a^j)+a^j=0$. As mentioned before, z^j serves as the direction of change in the simplex method, when changing the value of a nonbasic variable j. For the methods considered here, linear combinations of such vectors serve as such directions z; i.e., if $Z = z(z^j)$ is the $n\times(n-m)$ matrix having vectors z^j as its columns and w is an $(n-m)$-vector of weights, then

$$z = Zw \quad . \tag{3}$$

We shall index the components of w by nonbasic variables rather than the first n - m integers. Thus, reference to w_j always carries the convention that $j \in \bar{\beta}$. Taking (2) into account, the components w_j indicate the direction of change in the space of nonbasic variables while z is the direction in the space R^n of all variables.

In general, certain conditions are to be met by an admissible direction in order for the method to converge: (i)For the direction to be feasible, we require (for a nonbasic variable j currently at its bound) that $w_j \geq 0$ for $j \in \alpha$ and $w_j \leq 0$ for $j \in \gamma$. (ii)In order to improve the objective function, we must have $cZw > 0$. (iii)Finally, in order to prevent zig-zagging, we require that $cz^j w_j > 0$ if $w_j \neq 0$. If no $w \in R^{n-m}$ satisfies conditions (i) - (iii), then the current solution is optimal for (LP). (For a proof, see reference [2].)

In the simplex method, an admissible direction w is a unit vector for which cZw is positive or negative depending on whether the particular nonbasic variable is currently on its lower or upper bound. For the reduced gradient method, w is given by

$$w_j = \begin{cases} 0 & \text{if } j \in \alpha \text{ , and } cz^j < 0 \text{ , or} \\ & \phantom{\text{if }} j \in \gamma \text{ , and } cz^j > 0 \text{ ,} \\ cz^j & \text{otherwise .} \end{cases} \tag{4}$$

That is, nonbasic variables are adjusted in proportion to their reduced costs unless they are currently at a bound and a feasible movement off from the bound will not increase the objective function.

The Basis Change

Initially, any basis can be chosen independently of the initial solution. At an iteration, if a nonbasic variable moves to its bound, then we simply leave the basis unchanged. Otherwise, at least one basic variable reaches its lower or upper bound.

We may arbitrarily[1] select one of these to be the leaving variable ℓ. For the entering variable, there may be many candidates: any variable e is a candidate if it is currently off from its bounds (i.e. $0 < x_e < u_e$) and $\beta' = \beta \cup \{e\} - \{\ell\}$ is a legitimate set of basic variables. It has been shown in [2], that if (LP) is nondegenerate, then such a variable e always exists. Implementation of the basis change rule will be dicussed in Section 3 in detail.

The Method

The steps of the methods in consideration can be stated as follows:

1° Initialization: Specify an initial basis (set of basic variables β), an initial feasible solution x and the corresponding sets $\alpha = \alpha(x)$ and $\gamma = \gamma(x)$.

2° Specify direction: Determine a vector w of weights satisfying conditions (i) - (iii) above. If none exists, then stop (the current solution x is optimal).

3° Determine step size: Let $\bar{\theta}$ be the largest θ for which $x + \theta Zw$ is feasible. If $\bar{\theta} = \infty$, then stop ((LP) is unbounded).

4° Update: Replace x by $x + \bar{\theta}Zw$. Thereafter,

4.1° if any of the nonbasic variables moved to its upper or lower bound, update α and γ, and return to 2° (without a basis change);

4.2° otherwise, update α and γ , and pick any $\ell \in \beta\cap(\alpha\cup\gamma)$

[1] Actually, standard pivot selection rules are used.

(a basic variable on its bound) as leaving variable. Pick $e \in \overline{\beta} \cap \overline{\alpha} \cap \overline{\gamma}$ (a nonbasic variable off from its bounds) such that $\beta' = \beta \cup \{e\} - \{\ell\}$ is a legitimate set of basic variables. Replace β by β' and return to 2°.

3. Implementation: The Basic Version

The SESAME system was modified for adopting the features of the method described above. We shall describe an implementation which later will be referred to as the basic version. In subsequent sections we report computational experience with the basic version as well as with several of its modifications.

Shortly stated, the basic version is just the reduced gradient method modified so that only a certain subset of nonbasic variables is considered for changing at each iteration. We shall first give a brief overview of the SESAME system. Thereafter, following the steps listed for the method in Section 2, we shall discuss details of our implementation. Such a discussion ought to be useful when we consider alternative implementations for these particular steps in subsequent sections.

The SESAME System

The SESAME mathematical programming system is a large out-of-core MPS with simplex algorithms and supporting procedures in traditional style. Its grandparentage is partly IBM's MPS/360 and its parentage partly Management Science System's (now Ketron) MPS-III [8]. SESAME includes an elaborate data management extension, called DATAMAT, which has very similar external (but not internal) specifications to MPS-III's DATAFORM. Both these extensions are the outgrowth of several lines of development going back as far as 1959 [6].

SESAME was designed from the beginning for use only on an interactive host, namely an IBM/370 operating under VM/CMS. While this restricts its portability, specialization to one type of computer enhances efficiency as with all other large MPS's. Both SESAME and, particularly, DATAMAT have been enhanced and extended at IIASA, utilizing the IBM 370/168 at the CNUCE center in Pisa, Italy. SESAME is controlled by the user through and only through a remote terminal. There is no such thing as "submitting a job". Instead the user creates standard sequences of instructions--at various levels--in the form of files which are then invoked by a command at the terminal. The creation, modification and invocation of these "run" and "program" files are all performed interactively, as is ad hoc use of various system facilities.

The main simplex algorithm in SESAME combines the primal, dual, generalized upper bounding (GUB) and separable programming all in one procedure. It also includes bounds and ranges of all types, multiple and partial pricing, and a number of algorithm control switches. (Multiple pricing and suboptimization is permanently limited to seven columns, which becomes important below). Both standard MPS input and MPS-III extensions as well as another better but little-used format are accepted. Most models, however, are created with DATAMAT which enfiles them directly without an intermediate card-image form. Standard output of the various usual kinds is provided and, additionally, LP results may be enfiled directly for subsequent use with DATAMAT functioning as a report generator or master algorithm control. The system includes a number of other features which are of no particular pertinence here.

Initialization of the Method

We shall now turn our discussion to the implementation of our basic version of the reduced gradient method in the SESAME system. For the basic version, either an all logical starting basis (i.e. a basis consisting of slacks and artificials only) can be constructed or an advanced basis is loaded. The latter alternative is available if a basis from previous runs has been saved or if such a basis has been generated by other means. However, no crash algorithm has been employed.

The initial solution of the basic version is the basic solution corresponding to the initial basis. If this solution is not feasible, we start Phase I in the usual way for minimizing the sum of infeasibilities. Thus in this case, the objective function coefficient is set to -1 for all variables above their upper bound (including artificial variable at a positive value), 1 for all negative variables and to 0 in other cases.

Specifying Direction

At each iteration we consider at most $k = 7$ nonbasic variables to be changed simultaneously. In the following, this set is called the k-set. The maximum number of elements in the k-set was due to an implementation similar to one employed for a multiple pricing procedure in the SESAME system. In such a case, the alpha columns (the columns a^j premultiplied by the basis inverse) for nonbasic variables j to be moved are stored explicitly, and core limitation soon becomes prohibitive for larger k.

While choosing the k-set we cycle through the nonbasic variables in a similar manner to one of the standard partial pricing schemes in the simplex method. We need to find, if possible, a set of t (standard value of $t = 12$) nonbasic variables, called the t-set, for which formula (4) of the reduced gradient method yields a nonzero weight w_j. Among the t-set we choose, when possible, k variables with the largest weights in absolute value. The optimum for (LP) has been obtained if the t-set is empty.

After choosing in this way the k-set from the set of all nonbasic variables, compute the alpha-columns for the k-set (all in one FTRAN pass), we set the weights according to (4) and move in this direction. If a nonbasic variable (one or more) encounters a bound, we redefine its weight according to (4) and leave the k-set unchanged. Otherwise, a basic variable ℓ having moved to its bound is replaced by a variable e of the k-set. Thereby the size of the k-set is reduced by one element, and the alpha-columns and reduced costs are updated. We repeat such iterations until either the k-set becomes empty or the weights for all variables in the k-set are equal to zero. Therafter, a new k-set (of at most 7 variables) is chosen among the nonbasic variables as described above.

Remark. Alternatively, the composite direction may be computed applying FTRAN on the composite column $\sum_j w_j a^j$ (where summation is taken over the k-set). This approach has been adopted in the nonlinear programming system MINOS by Murtagh and Saunders [4,5]. The advantage is that the alpha-columns need not be stored nor computed for each j in the k-set. For large k, this is superior to the approach we have taken. However, for small k, this approach is likely to require mor work per iteration because normally a second FTRAN is needed to compute the alpha-column for the variable entering the basis. A fair comparison of these two alternatives remains a topic of future research.

Determining the step size

As indicated above, the alpha-columns for all nonbasic variables in the k-set are stored explicitly. When a new k-set is chosen, an FTRAN pass is needed to compute these alpha-columns. Otherwise, the existing alpha-columns are just updated in the usual way utilizing the alpha-column of the entering variable. Given the alpha-columns, a composite column is computed as a weighted sum of these vectors, the weights being those given by the direction w.

For Phase II, the minimum ratio test is carried out using the composite vector as usual to determine the step size. For Phase I, however, there are several alternatives. The rule adopted in our basic version is to move as far as (i) a currently feasible variable reaches its bound, or (ii) an infeasible variable, moving towards feasibility, reaches its farthest finite bound, whichever occurs first.

Updating the basis inverse

The basis inverse is stored in a product form and, given a leaving and an entering variable, updated exactly as in the simplex algorithm of the SESAME system. In our case, however, there is some freedom in choosing the entering variable. As shown by the following result,we may exclude from consideration all nonbasic variables which are not in the k-set.

Lemma. Let $\ell \in \beta$ be a basic variable becoming binding at the current iteration. Then there exists in the current k-set a variable e such that $\beta' = \beta \cup \{e\} - \{\ell\}$ is a legitimate set of basic variables, and such that the updated price vector corresponding to β' is (dual) feasible for column ℓ.

Proof: Let d_j be the reduced cost and α_ℓ^j the element of the alpha-column j in pivot row ℓ, for each j in the k-set. If basic variable ℓ is forced to its lower bound, then there must be a variable j in the k-set for which either $d_j > 0$ and $\alpha_\ell^j > 0$ or $d_j < 0$ and $\alpha_\ell^j < 0$. On the other hand, if ℓ is forced to its upper bound, there exists variable j, for which either $d_j > 0$ and $\alpha_\ell^j < 0$ or $d_j < 0$ and $\alpha_\ell^j > 0$. In each case one can readily check that the result follows.||

Among all candidates e implied by this Lemma, we choose as the entering variable the one off bound with the largest pivot element. If this element is within the range of a pivot tolerance (standard threshhold is 10^{-8}) the variable with the largest pivot element among all columns suggested by our Lemma is chosen. If both fail, this can only be due to digital difficulties, and no provision has been implemented to avoid this, except the possibility to change the tolerance.

4. Computational Experience: General LP

4.1 Test Problems

The following test problems were considered: a tiny oil refinery model (A), agricultural planning models (B), (C) and (D), an energy supply model (E), and dynamic forest sector models (F) and (G). All models (B) to (G) have been developed in conjunction with research projects at IIASA. The forest sector models (F) and (G), which have been tested more extensively in this paper, have been reported in [3]. Statistics concerning these test problems are given in Table 1 below.

Table 1. Summary of test problems.

Problem	Rows	Columns	Density (%)
A	23	23	18.1
B	451	507	1.0
C	476	532	1.0
D	152	218	6.1
E	50	165	12.1
F	521	612	0.6
G	2321	3188	0.14

4.2 Results with the basic version

Table 2 below shows computational results of our basic version compared with the simplex method (as implemented in the SESAME system).

In each case, we have started with an all logical basis and the initial solution is the corresponding basic solution. The initial number of infeasibilities is shown, and the number of iterations required for reaching a feasible solution as well as an optimal solution is given. Furthermore, a measure for primal degeneracy is given for the initial and optimal solution in terms of the number of basic variables equal to zero. We shall refer to this measure in subsequent sections.

As a measure for computational efficiency, the number of iterations, or rather the number of basis changes, may be used. For the reduced gradient method we did not count the minor iterations when a nonbasic variable moves to its lower or upper bound (the case without a basis change). On the other hand, an iteration is counted for the simplex method, when a nonbasic variable is moved from one bound to another. A set of experiments

Table 2. Experience with the basic version of the reduced gradient method compared with the simplex method of SESAME.

	Reduced gradient method					
Problem	A	B	C	D	E	F
Initialization:						
Infeasibilities	4	58	0	32	13	81
Basic variables equal to zero	13	266	48	93	21	362
Feasible solution:						
At iteration	26	-	1700+	288	47	976
Optimal solution:						
At iteration	28	400*		444	106	1462
Basic variables equal to zero	0	3		16	10	20
Nonbasic variables not on bound	1	15		4	1	25

	Simplex method					
Problem	A	B	C	D	E	F
Feasible solution:						
At iteration	23	-	1175	171	40	818
Optimal solution:						
At iteration	25	360*	1688	293	105	1085

*the problem was found to be infeasible.
+run was interrupted without finding a feasible solution.

was carried out on Problem F, which showed that the average CPU time per iteration for the reduced gradient method is .8 times that for the simplex method. Thus, to make the number of iterations comparable measures for computational efficiency, the iteration numbers in Table 2 for the reduced gradient method should be multiplied by a factor of .8.

In order to explain this factor we may consider two types of iterations: first, those where the k-set is selected from among the nonbasic variables, and second, the rest of the iterations (i.e., those where only the variables in the k-set are considered). The first type of iteration occurs in the simplex method when a multiple pricing pass is carried out. Obviously, the second type of iteration is cheap compared with the first type, because FTRAN and BTRAN are unnecessary (the alpha columns and the reduced costs of nonbasic variables in the k-set can be updated in a more simple and straightforward manner).

The reason for an RGM iteration (in our implementation) to be cheaper on the average than a simplex iteration result from the observation that the proportion of the second type of iterations is larger for the reduced gradient method than for the simplex method. This in turn results from the strategies implemented. In the simplex method a new k-set having *at most* 7 columns is chosen when the reduced costs in the current k-set are equal to zero within a tolerance. The actual number of columns chosen to the k-set is determined by a heuristic rule. In the reduced gradient method we choose always 7 columns to the k-set (if possible), and a new set is chosen when the k-set is empty or when the reduced costs in the current k-set are equal to zero (within zero tolerance, which is much smaller than the tolerance used for the simplex method).

According to Table 2, the overall performance of the basic version of the reduced gradient method is about equal compared with the simplex method of the SESAME system. (The difficulty in finding a feasible solution to problem C is unexplained. The source of the model is obscure and no investigation was possible).

4.3. Choosing a Nonbasic Starting Solution

Because the right hand side vector b normally is a relatively sparse vector, the initial solution is highly degenerate, when an all logical starting basis is chosen. This in turn results in a large number of iterations with a step size equal to zero. The ratio of such iterations for problems B and D, for instance, was more than 50 percent, most of which occured during the early iterations for both of the methods. In the following we report a little study, where we consider an approach for avoiding this phenomenon and investigate whether something can be gained in doing so.

Basically, our approach is to start the reduced gradient method with a nonbasic solution. We try to provide some motivation for this approach through an example, which has been presented in Figure 1.

The origin $(x_1, x_2) = (0, 0)$ in the picture corresponds to the basic solution for an all logical starting basis which is comprised by the (columns of the) slacks s_i. This solution is highly degenerate as nine out of ten of the basic variables are equal to zero. There is only one infeasibility $(s_1 = -10)$. When the standard simplex method or our basic version is used, either 2,3,4,5,6, or 7 iterations are required, depending on the choice of alternative pivot paths, to reach the optimal solution $(x_1, x_2) = (10, 10)$. For all the iterations, except the last one, the step size is equal to zero and the resulting solution is the same as the starting solution.

$$
\begin{array}{lrcl}
\text{minimize} & -x_1 \qquad + s_1 & = & -10 \\
\text{subject to} & -5x_1 + x_2 + s_2 & = & 0 \\
 & -4x_1 + x_2 + s_3 & = & 0 \\
 & 3x_1 - x_2 + s_4 & = & 0 \\
 & 5x_2 - 2x_2 + s_5 & = & 0 \\
 & 2x_1 - x_2 + s_6 & = & 0 \\
 & 5x_1 - 3x_2 + s_7 & = & 0 \\
 & 3x_1 - 2x_2 + s_8 & = & 0 \\
 & x_1 - x_2 + s_9 & = & 0 \\
 & 2x_1 - x_2 + s_{10} & = & 0 \\
 & \multicolumn{3}{l}{x_1,\ x_2 \geq 0 \quad ,\ s_i \geq 0 \text{ for all i.}}
\end{array}
$$

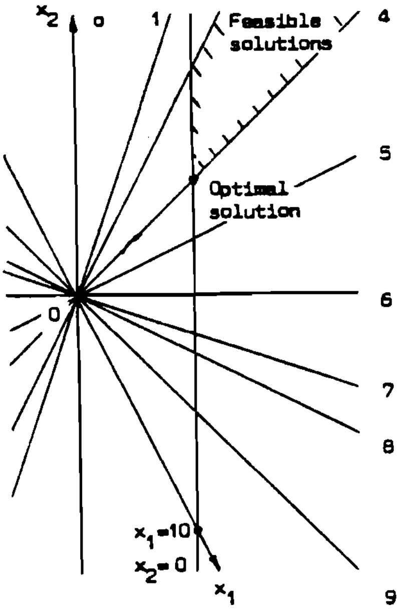


Figure 1. An example of a degenerate, all logical starting basis.

For the reduced gradient method, we may choose a nonbasic starting solution. For instance, we may choose the starting basis as above, set the nonbasic variables to any nonnegative value, and solve (LP2) for the basic variables to obtain a nonbasic system solution to start with. In particular choosing any such point, other than the origin, the number of iterations to reach the optimum is either 2 or 3, depending on the choice. Thus, it seems likely that starting with a nonbasic solution results in a decrease in the number of iterations in this example. Notice, that the number of infeasibilities at such a starting solution ranges between 0 and 7. (For brevity, we shall not discuss the possible pivot paths here).

We shall now add to our basic version the possibility of setting nonzero values to the nonbasic variables at the starting solution (given that the initial basis has already been chosen). Because, in general, no indication may be available as to which values should be used, we have implemented the possibility of setting the same arbitrarily chosen nonnegative value for all nonbasic variables.

Table 3 below shows the effect of starting with such nonbasic solutions. As a general observation, we may conclude that setting all nonbasic variables initially to a given nonzero value indeed yields a slight improvement (but not in that degree which might be suggested by our example). The number of iterations with a stepsize equal to zero was decreased dramatically, and thereby the functional value both in Phase I and in Phase II improved smoothly.

Table 3. Starting with a nonbasic solution and an all logical basis.

	Initial solution			Feasible solution		Optimal solution	
Problem	N.b. value	Degeneracy	Infeasibilities	Iteration	Functional	Iteration	Functional
D	0	93	32	288	-7.3	444	6.0
D	1	2	115	233	-7.6	366	6.0
D	10	2	115	217	-7.4	367	6.0
D	100	3	115	220	-7.2	411	6.0
F	0	362	81	976	-21	1462	-0.6
F	1	9	322	986	-21	1475	-0.6
F	10	9	354	1101	-5	1264	-0.6
F	100	7	338	961	-6	1064	-0.6

N.b. value = initial value for nonbasic variables; Degeneracy = initial number of basic variables equal to zero; Feasible solution = number of iterations for feasibility and the functional value; Optimal solution = number of iterations for optimality and the functional value.

4.4 Improving the Functional Value in Phase I

The fact that the feasible solution generated in Phase I is often a relatively poor solution, led us to try to take into account also the functional when choosing the direction in Phase I. We shall report such an experiment as well as another attempt aimed at improving Phase I in the following.

Our intention now is to specify the vector of weights w for the direction z = Zw in such a way that, in Phase I, improvement is made for the functional value cx as well as for the sum of infeasibilities.

Let c^1x denote the objective function of an ordinary Phase I. We shall now replace this objective by $(c^1 + \lambda c)x$, where λ is a positive parameter. Each time, when optimality has been reached with this objective function, and there are still infeasibilities left, we switch back to the ordinary Phase I routine and stay there as long as the solution remains optimal subject to the modified objective. Thus, the technique is one version of the "composite objective" option available in some of the commercial MPS's.,

The results of our experiments were negative: our general observation was that the total number of iterations for reaching optimality increased considerably; e.g., by fifty per cent for Problem F when the standard version was used. Typically, the primal objective function improved well along the Phase I iterations, even reaching the neighborhood of the optimal value, but then a switch to the ordinary Phase I resulted in a large degradation in the functional value.

As another attempt to improve Phase I we implemented a procedure for choosing the step size at each iteration in such a way that the sum of (the values for) infeasible variables is minimized. For the simplex method such a step-choosing technique is uncommon, but not new. (It has been implemented in MPS III, for instance.) We denote the sum of infeasible variables as a funciton of step size θ by $\zeta(\theta)$. A typical picture of such a function is shown in Figure 2. It is a convex, piece-wise linear function whose derivative is discontinuous at points θ_0, θ_1, θ_2, et cetera. At each of these points one or more variables become either feasible or infeasible. The minimization of this function, subject to the requirement that the nonbasic variables are not allowed to become infeasible, can be done easily because the information needed to compute the slope changes at each of the points θ_i, is readily available in the composite vector $z = Zw$.

Somewhat surprisingly, the approach was also a setback compared with the basic version: suboptimization over θ caused an increase in the number of iterations for reaching feasibility.

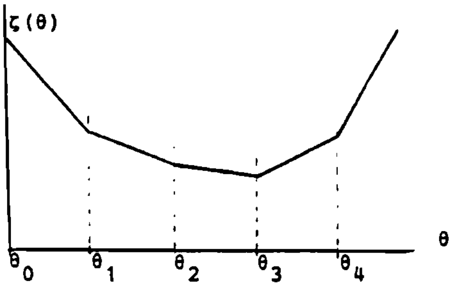


Figure 2. Sum of infeasible variables as a function of step size.

5. Specialization for Dynamic Linear Programming

In this section, further elaboration is made on choosing an initial nonbasic solution as well as an initial basis in the case of dynamic linear programming.

5.1 The Dynamic Linear Programming Problem

The dynamic linear programming problem (DLP) is an important special case of (LP). At the same time, it is known as a particularly difficult class of LP problems. The problem can be stated as follows [9]:

find $x(t)$ and $u(t)$, for all t, to

$$\text{(DLP1)} \quad \text{maximize} \sum_{t=0}^{T-1} (a(t)x(t) + b(t)u(t)) + a(T)x(T)$$

s.t.

$$\text{(DLP2)} \quad x(t+1) = A(t)x(t) + B(t)u(t) + s(t) \quad , \quad 0 \leqslant t \leqslant T-1$$

$$\text{(DLP3)} \quad G(t)x(t) + D(t)u(t) = f(t) \quad , \quad 0 \leqslant t \leqslant T-1$$

$$\text{(DLP4)} \quad u(t) \geqslant 0, \ x(t) \geqslant 0 \quad , \quad \text{for all } t$$

$$\text{(DLP5)} \quad x(0) = x^0 \quad .$$

Here $x(t) \in R^{n_t}$ is the vector of state variables at the beginning of period t, for $t = 0, 1, \ldots, T$, and $u(t) \in R^{r_t}$ is the vector of control activities during period t, for $t = 0, 1, \ldots, T-1$. For each t, $a(t) \in R^{n_t}$, $b(t) \in R^{r_t}$, $s(t) \in R^{m_t}$ and $f(t) \in R^{k_t}$ are externally given vectors, and $A(t)$, $B(t)$, $G(t)$ and $D(t)$ are externally given matrices of appropriate dimension. The initial state of the system is described by the vector $x^0 \in R^{n_0}$. The objective function in (DLP1) is a linear function of state variables $x(t)$

and control variables u(t). Constraints (DLP2) may be called the state equations, as they determine the state x(t+1) at the end of a period t (beginning of the subsequent period t+1) given the initial state x(t) and the control action u(t) for that period.

Clearly, (DLP) is a special case of (LP). The constraint matrix A for (DLP) has been illustrated in Figure 3 for T = 3.

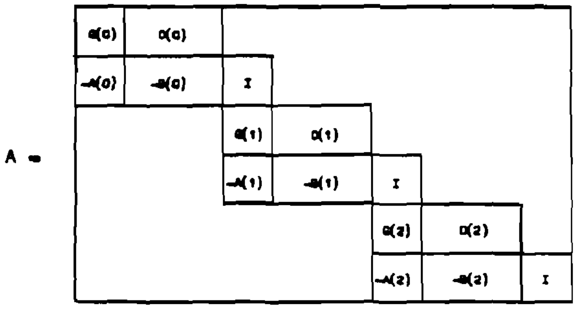


Figure 3. A dynamic LP with three time periods

In the following, we shall experiment with ideas of choosing an initial basis and an initial solution, when the reduced gradient method is applied to (DLP).

5.2. An Advanced Basis for Dynamic LP

For dynamic linear programs, it may seem intuitively appealing that most of the state variables appear in the optimal basis. In fact, for various versions of DLP Problems F and G, over 90% of the state variables appear in the optimal basis. Furthermore, we believe that in a typical dynamic LP formulation, besides the state equations (DLP2), there are only a relatively small number of constraints of equality type; i.e., most of the constraints (DLP3) are just inequalities which have been converted to equalities through adding the slack variables. For Problem F, 95% of constraints (DLP3) are converted inequalities. For problem G this ratio is 80%.

These remarks led us to construct an advanced triangular basis which consists of (i) columns of all state variables, (ii) columns of slacks for inequality type constraints in (DLP3), and (iii) artificial columns for equality type constraints in (DLP3). An example of such a basis corresponding to our example in Figure 3 is given in Figure 4.

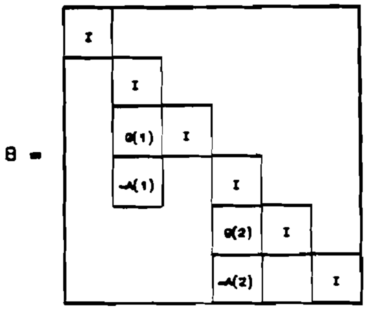


Figure 4. An advanced basis for dynamic LP.

When the basic version was used for Problem F and the above constructed basis was used as a starting basis, the number of iterations was reduced from 1462 corresponding to an all logical starting basis to 583. When the same basis was used for the simplex method, only 363 iterations were needed. However, when the constructed initial basis was combined with an initial nonbasic solution where all the nonbasic variables were set to one, the number of iterations was reduced to 260. For the nonbasic variables equal to 10 and 100, the respective numbers of iterations were 313 and 399. This may support our earlier conjecture in Section 4.3 concerning possible advantages in starting with a nonbasic solution. In any case, the result seems promising as the total number of iterations was reduced by a factor of four to five.

5.3 Initial Solutions for Dynamic LP

We already obtained a relatively encouraging result while using initially the constructed basis and setting the nonbasic variables to a constant value. We shall now experiment further with some straightforward ideas for setting initial values to the controls.

Setting Controls to the Same Level at Each Period

Typically in a DLP the same or almost the same set of control variables (as well as state variables) repeat from one period to another. Let us concentrate on those controls which are common to all periods. Initially, we may set these controls to the same level at each period and the rest of the controls to zero.

At least the following two approaches may be used to determine an initial value for the joint set of controls: (i) We adopt the real current levels for those controls (provided that the system described by DLP already exists), or (ii) we solve first a one-period problem (perhaps with appropriate bounds for the final state variables) and adopt the values for the joint set of controls from this optimal solution.

For the two dynamic problems F and G, exactly the same set of controls appear at each time period. As both of the models describe a real forest sector, the current rates for controls were easily available. When the constucted basis was used initially and all the controls were set to their current values it took 240 iterations to solve Problem F representing a reduction by a factor of about 6 compared with the basic version. We should note that the initial solution constructed this way was not feasible: there were 34 infeasibilities for Problem F initially.

The other approach (ii) for constructing initial values for controls was applied as well. For the first period model we require the final state to be at least as good as the initial state; i.e., for each state variable for which a large value is desireable (e.g. wood in the forest, production capacity, etc.) the initial value sets a lower bound for the final value, and for other state variables (e.g. amount of long term external financing) the initial value sets an upper bound for the final value. Starting with the constructed basis for DLP and the controls set to the optimal level of the one period model resulted in 213 iterations for Problem F, thus yielding a slight improvement over the previous approach. Again the initial solution was infeasible. This approach was also applied to the larger DLP model G. The optimal solution was found in 3050 iterations.

Constructing a Feasible Solution

A relative drawback was notable in both of the previous attempts in trying to construct an initial nonbasic solution. As the initial solution was not feasible, it appeared that the relatively good initial functional value got substantially worse during the Phase I procedure. Thus we concluded that it would be desirable to construct an initial solution which is also feasible. Indeed, as described below, we were easily able to carry out this task for the two test problems F and G. Of course, the generality of such an approach may be doubtful. However, it is the authors' belief that a similar approach is applicable to most dynamic linear programs.

We shall now turn to a case of constructing a feasible starting solution. For Problem F, we first set the controls of all periods to the optimal level of the one period model. The printout of this solution indicated only two types of infeasibilities: one state variable, cash, became negative for most time periods, and the only equality type of constraint--other than the state equations--was violated for all except the first time period, i.e., the corresponding artificial variable appeared at a nonzero level. This equality constraint defines the profit (for each time period). Taking into account the objective function it became clear that a profit as large as possible was desired for an optimal solution. This allowed us to replace the equality by an inequality, and consequently the artificial variable in the constructed basis was replaced by a slack variable. For bringing the negative cash to a feasible range we simply adjusted a control variable determining the level of external financing. After these changes, the cash was brought to a feasible range, all the new slacks, corresponding to the rowdefining profit were nonnegative, and no new infeasibilities appeared; i.e., the initial solution was feasible.

Starting with this feasible (nonbasic) solution for Problem F, and with the advanced basis, it took 161 iterations for finding an optimal solution. A similar process was carried out for Problem F to construct a feasible initial solution based on the current levels of controls. The resulting number of iterations for finding an optimal solution was 180.

Thus, when the advanced starting basis was used together with a feasible initial solution, the number of iterations for finding an optimal solution by the reduced gradient method was reduced by a factor of eight to nine compared with starting with an all logical basis and the corresponding basic solution.

As was noted above, an initial basis can provide a good starting point for the simplex method. We should point out that the nonbasic values could be used with the simplex method as well, in some cases. This is true for some commercial MPS's that have an option for obtaining a vertex solution from a given nonbasic solution. Thus our initialization strategies could be employed immediately by the users of such existing systems. The SESAME system, however, did not allow us to experiment with the simplex method when a nonbasic starting solution was used.

6. Summary and Conclusions

This paper may be seen as consisting of three parts: First, details of a variation of the reduced gradient method and its implementation is discussed in Sections 2 and 3. Second, computational experience as applied to general linear programs is reported in Section 4. Third, specialization to dynamic linear programs is presented in Section 5. In the following we shall briefly discuss each part in turn.

(i) The SESAME system was adopted as a basis for implementation. A basic feature of our implementation is to compute explicitly the alpha-columns for each nonbasic variable being charged. Because of core limitations and for the sake of computational efficiency, we restrict to seven, the number of nonbasic variables allowed to change simultaneously. If a larger number is desired, an alternative approach to implementation would be preferred, where the weighted sum of the nonbasic columns is computed prior to the composite alpha-column. This in turn can be inefficient for a small number of nonbasics variables being changed simultaneously. As a topic for future study remains the question, which one of the two approaches is more efficient, taking into account both the average computational effort per iteration and the number of iterations (which may be influenced by the number of nonbasics being changed).

(ii) In the first part of computational experiments we present a comparison with the simplex method for general linear programs starting with an all-logical basis. The test problems being used are mainly medium sized sectoral economic models (energy, agricultute, etc.) developed at IIASA. According to our results, the overall performance of both methods is approximately the same. We have tested also some further modifications concerning the choice of an initial (nonbasic) solution and strategies for Phase I. Even though these did not, in general, yield an improvement for the reduced gradient method we felt that it would be of interest to report briefly our negative experience as well. In fact, further tuning of such strategies could well reverse the conclusions.

(iii) The most interesting practical results have been obtained in the final part where the special structure of dynamic linear programs is taken into account in starting the reduced gradient method. We observe first that the state variables in practical problems are likely to appear in optimal bases. This suggests to initiate with all state variables in the bases. Our experience shows that, when such a basis is completed with logical variables, considerable savings can be obtained indeed.

According to another abservation, most control variables in practical problems appear in all time stages. Thus we might start with such controls being set to the same value for each time period.

Suitable values may easily be obtained from empirical knowledge, or from a single-period model (e.g., a steady-state model). A few experiments have been reported, where such strategies are combined with the initial basis mentioned above. Also these results were encouraging in that considerable further gains in computational effort were obtained. Using an example, we have also demonstrated that the controls may actually easily be chosen to yield a feasible initial solution. Because Phase I is not needed, still further gains can be achieved. Of course, the procedure of generating such feasible controls is model-specific and may not always be possible.

No comparison is given with the simplex method in the case of dynamic LP. However, an example demonstrates a good performance of the simplex method when the initial basis involving the state variables is employed. It is likely that further improvement is achieved, as above, when the simplex method is initiated with a nonbasic solution as described. This, of course, would require an option to obtain a basic solution from a given nonbasic solution. Because such an option is available in some commercial MPS's, the strategies we have suggested for dynamic LP may be immediately employed by the users of such systems.

REFERENCES

[1] Dantzig, G.B. (1963) Linear Programming and Extensions. Princeton, N.J.: Princeton University Press.

[2] Kallio, M., and E. Porteus (1978) A class of methods for linear programming. Mathematical Programming 14:161-169.

[3] Kallio, M., A. Propoi and R. Seppälä A Model for the Forest Sector. WP-80-34. Laxenburg, Austria: International Institute for Applied Systems Analysis.

[4] Murtagh, B.A., and M.A. Saunders, MINOS User's Guide. Report SOL 77-9, Department of Operations Research, Stanford University, 1977.

[5] Murtagh, B.A., and M.A. Saunders, Large-Scale Linearly Constrained Optimization. Mathematical Programming 14 (1978), 41-72.

[6] Orchard-Hays, W. (1968) Advanced Linear-Programming Computing Techniques. New York: McGraw-Hill.

[7] Orchard-Hays, W. (1978) Anatomy of a mathematical programming system. Design and Implementation of Optimization Software, edited by H. Greenberg. Netherlands: Sijthoff and Noordhoff.

[8] Orchard-Hays, W. (1978) Scope of mathematical programming software. Design and Implementation of Optimization Software, edited by H. Greenberg. Netherlands: Sijthoff and Noordhoff.

[9] Propoi, A., and V. Krivonozhko (1978) The Simplex Method for Dyanamic Linear Programming. RR-78-14. Laxenburg, Austria: International Institute for Applied Systems Analysis.

[10] Wolfe, P. (1967) Methods of nonlinear programming. Nonlinear Programming, edited by J. Abadie. New York: Wiley.

NETWORK PROBLEMS

A SUCCESSIVE LINEAR OPTIMIZATION APPROACH TO THE DYNAMIC TRAFFIC ASSIGNMENT PROBLEM*

James K. Ho**

Applied Mathematics Department
Brookhaven National Laboratory
Upton, N.Y.

A dynamic model for the optimal control of traffic flow over a network is considered. The model, which treats congestion explicitly in the flow equations, gives rise to nonlinear, nonconvex mathematical programming problems. It has been shown for a piecewise linear version of this model that a global optimum is contained in the set of optimal solutions of a certain linear program. This paper presents a sufficient condition for optimality which implies that a global optimum can be obtained by successively optimizing at most N + 1 objective functions for the linear program, where N is the number of time periods in the planning horizon. Computational results are reported to indicate the efficiency of this approach.

*Published in *Transportation Science* 14(4), 1980. Reproduced with permission. Copyright © 1980 The Operations Research Society of America.

The submitted manuscript has been authored under contract DE-AC02-76CH0016 with the U.S. Department of Energy. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

**The author wishes to thank George Nemhauser and two anonymous referees for helpful comments.

The Problem

We consider a traffic network represented by a directed graph, as in Fig. 1. One of the nodes is designated as the destination. The planning horizon is divided into a finite number of discrete time periods. For each time period, external inputs are allowed at any node except the destination. For each arc, there is an exit function which relates the amount of traffic entering and leaving the arc during a time period. Congestion is modelled ((cf.[5]) by assuming the exit functions to be nondecreasing, continuous and concave, as in Fig. 2. The object is to find the feasible traffic flow that minimizes a cost function which, to express the disutility of congestion, is assumed to be the sum of nonnegative, nondecreasing, continuous convex functions in the arc flows. To formulate the problem, we use the following notation:

$G = (\mathcal{N},\mathcal{Q})$ a directed graph;

$\mathcal{N}$ = set of nodes of G;

$\mathcal{Q}$ = set of directed arcs of G;

N = planning horizon;

i = index of time period; i = 0, 1,...,N;

j = index of arc in $\mathcal{Q}$; j = 1,...,a;

q = index of node in $\mathcal{N}$; q = 1,...,n;

n = index of destination node;

$A(q) = \{j\epsilon\mathcal{Q}|$ arc j leaves node $q\}$;

$B(q) = \{j\epsilon\mathcal{Q}|$ arc j enters node $q\}$;

$F_i(q)$ = external input at node q in period i;

x_{ij} = amount of traffic (or flow) on arc j at the beginning of period i;

$h_{ij}(x_{ij})$ = cost of x_{ij} (the sum of these terms is to be minimized);

d_{ij} = amount of traffic admitted to arc j in period i;

$g_j(x_{ij})$ = amount of traffic to exit from arc j in period i.

The basic flow equations in the model are then

$$x_{i+1,j} = x_{ij} - g_i(x_{ij}) + d_{ij},\ i = 0,\ldots, N-1,\ \forall j \in \mathcal{A} \tag{1}$$

$$\sum_{j \in A(q)} d_{ij} = F_i(q) + \sum_{j \in B(q)} g_j(x_{ij}),\ i = 0,\ldots, N-1,\ \forall q \neq n. \tag{2}$$

For a piecewise linearization[1], we partition the nonnegative segment of the real line by K(j) grid points for each arc $j \in \mathcal{A}$. Denote these grid points by c_j^k so that $c_j^1 = 0$ and $c_j^{K(j)} = \infty$. See Fig. 3 . Let λ_{ij}^k be the interpolation weight on grid point k in period i, and g_j^k, h_{ij}^k the values of g_j and h_{ij} at that point respectively. Then redefining $c_j^{K(j)} = 1$, and taking each of $g_j^{K(j)}$ $h_{ij}^{K(j)}$ to be the slope of the last segment of the approximation to the corresponding function, we can express

[1] In [5], a sum-of-intervals represention of the piecewise linearization is used. The grid-point-interpolation representation in this paper is equivalent but is preferable for data generation and LP solution considerations. For example, explicit upper bounds on all variables are necessary in the former but none appears in the latter formulation.

$$x_{ij} \equiv \sum_{k=1}^{K(j)} c_j^k \lambda_{ij}^k$$

$$g_j(x_{ij}) \equiv \sum_{k=1}^{K(j)} g_j^k \lambda_{ij}^k$$

$$h_{ij}(x_{ij}) \equiv \sum_{k=1}^{K(j)} h_{ij}^k \lambda_{ij}^k \qquad (3)$$

for some $\lambda_{ij}^k \geq 0$, $k = 1,\ldots, K(j)$, with

$$\sum_{k=1}^{K(j)-1} \lambda_{ij}^k = 1$$

and $\lambda_{ij}^k > 0$ for at most two k's which furthermore are consecutive.

Substituting (3) into (1) and (2) we arrive at the following problem (P)

$$\text{minimize} \quad \sum_{i=0}^{N} \sum_{j=1}^{a} \sum_{k=1}^{K(j)} h_{ij}^k \lambda_{ij}^k = G_{N+1} \qquad \text{(LP.0)}$$

$$\text{subject to} \quad \sum_{k=1}^{K(j)} c_j^k \lambda_{i+1,j}^k = \sum_{k=1}^{K(j)} (c_j^k - g_j^k) \lambda_{ij}^k + d_{ij}$$

$$i = 0,\ldots, N-1,\ j = 1,\ldots, a \qquad \text{(LP.1)}$$

$$\sum_{j\epsilon A(q)} d_{ij} = F_i(q) + \sum_{j\epsilon B(q)} \sum_{k=1}^{K(j)} g_j^k \lambda_{ij}^k ,$$

$$i = 0,\ldots, N-1,\ \forall q \neq n \qquad \text{(LP.2)}$$

$$\sum_{k=1}^{K(j)} c_j^k \lambda_{0j}^k = R_j \text{ given, } j = 1,\ldots, a \qquad \text{(LP.3)}$$

$$d_{ij} \geq 0;\ i = 0,\ldots, N-1;\ j = 1,\ldots, a \qquad \text{(LP.4)}$$

$$\left.\begin{array}{ll} \lambda_{ij}^k \geq 0, & i = 0,\ldots, N \\ & j = 1,\ldots, a \\ \sum_{k=1}^{K(j)-1} \lambda_{ij}^k = 1; & k = 1,\ldots, K(j) \end{array}\right\} \qquad \text{(LP.5)}$$

and $\lambda_{ij}^k > 0$ for at most two k, which furthermore are consecutive. (OSP)

To obtain theoretical results, the following assumptions are made.

(A1) The arcs are not explicitly capacitated. This is modelled in (P) by letting $c_{ij}^{K(j)} = 1$ and excluding $\lambda_{ij}^{K(j)}$ from the convexity constraint in (LP.5).

(A2) $dg_j(x)/dx = 0$ for "large" x and all arcs, i.e. saturation is modelled by letting $g_j^{K(j)} = 0$.

(A3) For each j, the grid points are chosen such that the non-negative slopes of the piecewise linear approximation to g_j are strictly decreasing.

(A4) For the convex cost functions, we assume that

(a) $0 \leq h_{ij}^k \leq h_{ij}^{k+1}$ for all i, j and k;

(b) For each simple path containing arcs j_1 and j_2 such that

j_2 is the arc closer to the sink, $h_{ij_2}^{K(j_2)} \leq h_{ij_1}^{1}$ for all i.

Such cost functions provide incentive to move traffic efficiently toward the sink. Although condition (b) may be quite restrictive, it does accomodate for arbitrary network topology two cost functions likely to be useful in practice, namely

(i) $h_{ij}^{k} = 1$ for all i, j and k,

which gives the total cost as the amount of traffic in transit summed over the planning horizon; and

(ii) $$h_{ij}^{k} = \begin{cases} 1, & i = N \\ 0, & \text{otherwise} \end{cases}$$

which gives the total cost as the amount of traffic remaining in the network by the end of the planning horizon.

Agreement of Global Optima

Except for the last constraint (OSP) which is called the ordered solution property, (P) is a linear programming (LP) problem (cf. [1]). However, due to (OSP) the problem is nonlinear and nonconvex. If (LP.0-LP.5) is solved as a LP but the optimal solution does not satisfy (OSP), then it might appear that one must resort either to branch and bound techniques (see e.g.[2]) which are computationally very exp~~a~~ensive, or to ordered basis entry procedures in the simplex method (see e.g. [3]) which do not guarantee global optimality. This paper presents a more efficient approach.

Based on assumptions A1-A4, the following results are inferred from repeated application of Lemma 1 in [5].

Lemma A. If $y = \{\lambda^{k}_{ij}, d_{ij}\}$ is a feasible solution to (LP.1 - LP.5) that violates (OSP) for i = r and j = s, then there exists a feasible solution $\bar{y} = \{\bar{\lambda}^{k}_{ij}, \bar{d}_{ij}\}$ to (LP.1 - LP.5) that differs from y only for $i \geq r$ and for arcs j on paths beginning with arc s. x_{rs} given by λ in (3) equals $\bar{x}_{rs}$ given by $\bar{\lambda}$. $\bar{y}$ satisfies (OSP) for i=r and j=s as well as the conditions in Lemma B and Lemma C.

Lemma B. For all $q \in \mathcal{N}$ and i = 0,1,..., N, the total flow that reaches node q on or before period i is at least at great for $\bar{y}$ as for y in Lemma A.

Lemma C. The cost (LP.0) for $\bar{y}$ is no greater than that for y in Lemma A.

From repeated applications of Lemmas A and C, it follows that there exists an optimal solution to (LP.0-LP.5) that satisfies (OSP), hence is an optimal solution to (P). This agreement of the global optimum value for the two problems shows, in particular, that (P) attains a global optimum.

A Sufficient Condition for Optimality

We now show how a solution to (P) can be obtained by successive optimization of at most N+1 objective functions subject to (LP.1 - LP.5).

Let S be the set of optimal solutions to (LP.0 - LP.5). S is nonempty because the arcs are not explicitly capacitated and the cost is bounded from below by zero. S has in general more than one element because without enforcing (OSP) the grid point interpolation is not unique. Let $S_N \subseteq S$ be those solutions in S which maximize

$$G_N(\lambda) \equiv \sum_{i=1}^{N} \sum_{j=1}^{a} \sum_{k=1}^{K(j)} g_j^k c_j^k \lambda_{ij}^k .$$

In general, let S_t consist of those solutions in S_{t+1} which maximize

$$G_t(\lambda) \equiv \sum_{i=1}^{t} \sum_{j=1}^{a} \sum_{k=1}^{K(j)} g_j^k c_j^k \lambda_{ij}^k .$$

$G_t(\lambda)$ represents the total amount of traffic leaving all the arcs up to the end of time period t. Equivalently, it gives the total amount of traffic reaching all the nodes during or before period t. By the linearity of (LP.0-LP.5) and G_t we have

$$\emptyset \neq S_t \subseteq S_{t+1} \cdots \subseteq S_N \subseteq S, \; t = 1, \ldots, N .$$

Theorem. If $y = \{\lambda_{ij}^k, d_{ij}\} \varepsilon S_1$, then y is a solution to (P).

Proof. It suffices to show that $y \varepsilon S_1$ implies that $\{\lambda_{ij}^k\}$ satisfies (OSP). Suppose not, and that $\{\lambda_{ij}^k\}$ violates (OSP) for $i = r$, $j = s$.

Then by Lemma A, there exists $\bar{y} = \{\bar{\lambda}_{ij}^{k}, \bar{d}_{ij}\}$ that satisfies (OSP) for $i = r$, $j = s$, such that $x_{rs} = \bar{x}_{rs}$ where

$$x_{rs} = \sum_{k=1}^{K(s)} c_s^k \lambda_{rs}^k ,$$

$$\bar{x}_{rs} = \sum_{k=1}^{K(s)} c_s^k \bar{\lambda}_{rs}^k .$$

By Lemmas B and C, $\bar{y} \epsilon S_1 \subseteq \ldots S_r \subseteq \ldots S_N \subseteq S$.

Since $\sum_{k=1}^{K(j)} g_j^k c_j^k \lambda_{ij}^k$ is piecewise linear concave, violation of (OSP) under assumption (A3) always strictly underestimates its value. Therefore

$$\sum_{k=1}^{K(s)} g_s^k c_s^k \bar{\lambda}_{rs}^k > \sum_{k=1}^{K(s)} g_s^k c_s^k \lambda_{rs}^k . \tag{4}$$

Let q_s be the node to which arc s points. By Lemma A, y and $\bar{y}$ do not differ on other arcs pointing to q_s in period r, nor on any arc for $i < r$. Hence

$$G_r(\bar{\lambda}) > G_r(\lambda) , \tag{5}$$

contradicting the hypothesis that $y \epsilon S_1 \subseteq S_r$.

An Example

Consider the following numerical example of (P) with

$$N = 2$$

$$a = 3$$

$$n = 3$$

$$A(q) = \begin{cases} (1,2) & ; \quad q = 1 \\ (3) & ; \quad q = 2 \end{cases}$$

$$B(q) = \begin{cases} \emptyset & ; \quad q = 1 \\ (1,2) & ; \quad q = 2 \\ (3) & ; \quad q = 3 \end{cases}$$

$$K(j) = \begin{cases} 2 & ; \quad j = 1,2 \\ 1 & ; \quad j = 3 \end{cases}$$

$$h_{ij}^{k} = 1 \quad ; \quad \text{all } i,j,k$$

$$g_{j}^{k} = \begin{cases} 1 & ; \quad k = 1,\ j = 1,2,3 \\ 0 & ; \quad k = 2,\ j = 1,2 \end{cases}$$

$$c_{j}^{k} = \begin{cases} 50 & ; \quad k = 1,\ j = 1,2 \\ \infty & ; \quad k = 2,\ j = 1,2 \\ \infty & ; \quad k = 1,\ j = 3 \end{cases}$$

$$R_{j} = 0 \quad ; \quad j = 1,2,3$$

$$F_{i}(q) = \begin{cases} 100 & ; \quad i = 0,\ q = 1 \\ 0 & ; \quad \text{otherwise.} \end{cases}$$

Three solutions to (LP.0 - LP.5) are tabulated below, where an asterisk denotes an optimal value.

Solution	Y_1	Y_2	Y_3
x_{11}^1	50	50	0
x_{11}^2	0	50	50
x_{12}^1	50	0	0
x_{12}^2	0	0	50
x_{13}^1	0	0	0
x_{21}^1	0	50	50
x_{21}^2	0	0	0
x_{22}^1	0	0	50
x_{22}^2	0	0	0
x_{23}^1	100	50	0
G_3	200*	200*	200*
G_2	200*	150	100
G_1	100*	50	0
OSP	YES	YES	NO

We remark that all three solutions optimize (LP.0). $y_1 \varepsilon S_1$, hence satisfies OSP. $y_3 \notin S_1$ and violates OSP. However, $y_2 \notin S_1$ but still satisfies OSP. This illustrates that the condition of the theorem is sufficient but not necessary.

A Successive Linear Optimization Algorithm

To obtain a global minimum of (P), the following algorithm can be used. An asterisk denotes an optimal value.

Step 1. Minimize G_{N+1} subject to (LP.1 - LP.5) to obtain G^*_{N+1}

Set $t = N+1$.

Step 2. Test for (OSP), stop if satisfied.

Step 3. Add new constraint

$$G_t = G^*_t; \qquad \text{(c.t)}$$

maximize G_{t-1} subject to (LP.1 - LP.5) and (c.t - c.N+1) to obtain G^*_{t-1}. Set $t = t-1$ and return to Step 2.

Note that the solution obtained at the end of each step provides a feasible starting solution for the next step in the above algorithm.

Computational Experience

Implementation of the algorithm is very simple, as any available LP code can be adapted to perform the successive optimizations. The efficiency of the algorithm can be measured by the amount of computation in Steps 2 and 3 relative to that in Step 1. Admittedly, even with good starting feasible solutions, the solution of N additional LP's may still be costly when N is large. To gain some insight into this aspect of the algorithm we report computational experience on a test problem with the following characteristics:

number of nodes	$n = 7$
number of arcs	$a = 12$
number of periods	$N = 10$
number of grid points	$K(j) = 4, \; j = 1, \ldots, 12$

$$h_{ij}^{k} = 1, \; i = 1, \ldots, 10, \; j = 1, \ldots, 12, \; k = 1, \ldots, 4.$$

The network for the test problem is depicted in Figure 1. The exit functions for the arcs are given in Table 1. Five cases are considered by varying the external inputs to the nodes as given in Table 2. Each case gives rise to a LP with

311	rows
791	columns
3683	nonzero coefficients, and
1.5%	density.

A Fortran implementation of the revised simplex method with inverse in product form (see e.g. [4]) has been adapted to test for (OSP) and to control the successive optimizations. Table 3 records the number of steps required before a solution with (OSP) is obtained. Table 4 records the number of simplex iterations and CPU time involved in each step. In each case, step 1 was initiated with an all-logical (or artificial) basis. All CPU times reported are on a CDC 7600, excluding data input but including (OSP) testing.

Based on these computational results, the following observations are made. It is only when the network is extremely overloaded that a significant number of successive optimizations is required to obtain a solution with (OSP). In such cases (e.g. I and II for our test model), so much traffic never reaches the sink that it matters little whether certain arcs move their charges along according to the exit functions, or let them stall thus violating (OSP). In more realistic cases, even with substantial congestion in the arcs over various time periods, very few steps are required. Typically, maximization of the total traffic throughput (G_N) suffices, as with cases III, IV, and V for our test problem. In any case, one can expect the total computational effort to be very significantly less than N + 1 times that for the initial LP. Experience with other more complex test problems (up to 25 nodes, 65 arcs and 10 periods) agrees with the above observations and suggests that successive linear optimization is an efficient approach to the dynamic traffic assignment problem.

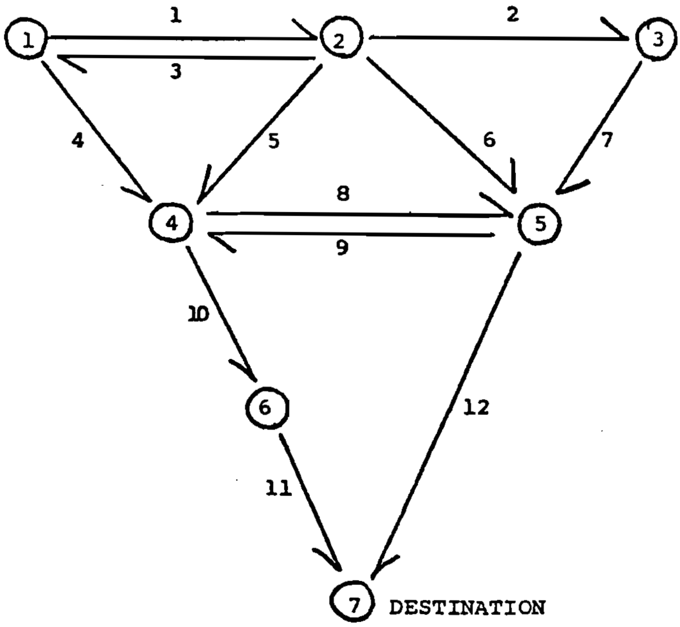


Fig. 1. Network for the Test Problem

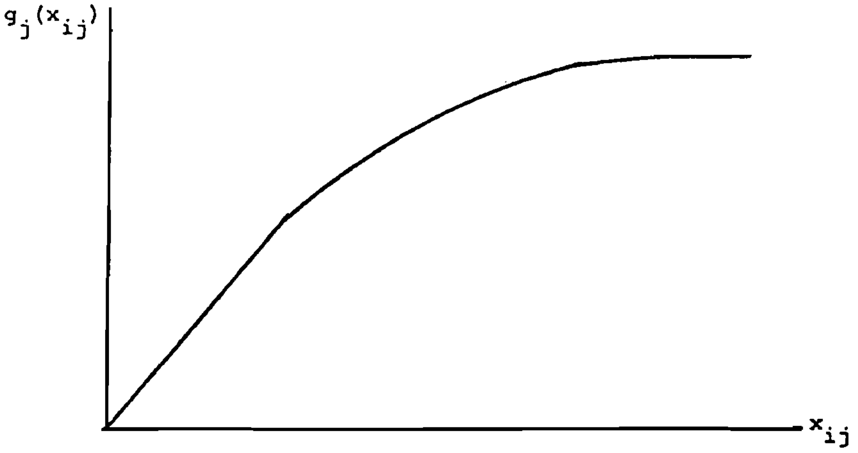


Fig. 2. A typical exit function for an arc

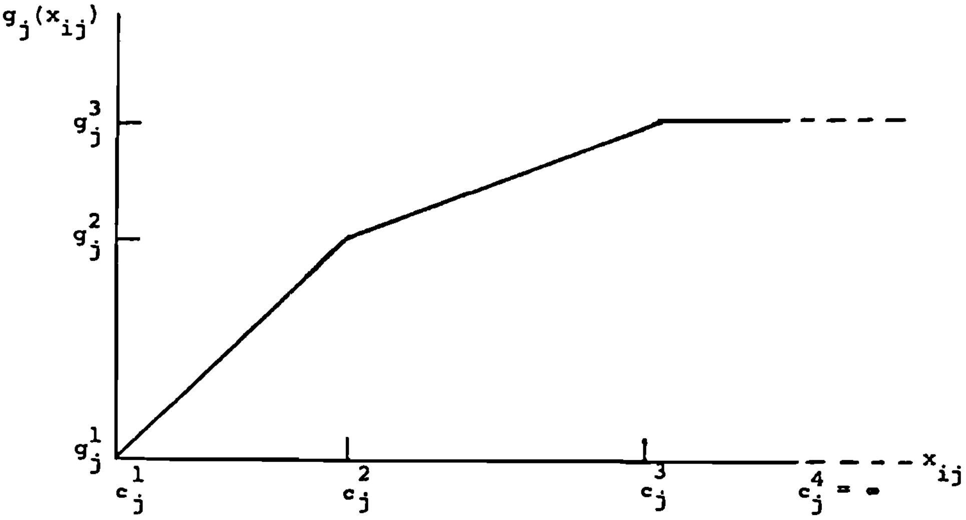


Fig. 3. Piecewise linearization of an exit function

TYPE	c_j^2	g_j^2	c_j^3	g_j^3	ARC j
1	10	10	20	15	1,3,6,7,10
2	15	15	25	18	2,4,5,11
3	30	30	40	33	8,9
4	50	50	80	65	12

$\{(c_j^1 \equiv 0,\ g_j^1 \equiv 0,\ c_j^4 \equiv \infty,\ g_j^4 \equiv 0$, for all j)}

Table 1

Exit Functions for Test Problem

CASE	$F_i(q)$, i=1,..., 10, q=1,..., 6
I	30
II	20
III	17.5
IV	15
V	10

Table 2

Node Input for the Five Cases.

CASE OSP Violations After Optimizing	I	II	III	IV	V
G_{11}	25	38	41	36	12
G_{10}	9	8	0	0	0
G_9	9	7			
G_8	7	3			
G_7	5	2			
G_6	4	1			
G_5	2	0			
G_4	2				
G_3	0				
G_2					
G_1					

Table 3

The Number of Steps Required to Obtain OSP

CASE Iterations (CPU Seconds)	I	II	III	IV	V
G_{11}	409 (7.1)	464 (8.9)	495 (10.2)	362 (7.2)	381 (6.1)
G_{10}	117 (2.7)	378 (9.9)	264 (7.2)	173 (4.6)	46 (1.0)
G_9	3 (0.4)	26 (0.8)			
G_8	15 (0.7)	8 (0.6)			
G_7	12 (0.6)	4 (0.6)			
G_6	12 (0.6)	34 (1.3)			
G_5	11 (0.7)	4 (0.6)			
G_4	48 (1.5)				
G_3	6 (0.6)				
TOTAL	633 (14.9)	918 (22.7)	759 (17.4)	535 (11.8)	427 (7.1)

Table 4.

Solution Statistics

REFERENCES

1. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

2. Garfinkel, R. S. and G. L. Nemhauser, Integer Programming, Wiley, New York, 1972.

3. Hadley, G., Nonlinear and Dynamic Programming, Addison-Wesley, Reading, Mass., 1964.

4. Lasdon, L. S., Optimization Theory for Large Systems, Macmillan, New York, 1970.

5. Merchant, D. K. and G. L. Nemhauser, "A model and an algorithm for the dynamic traffic assignment problem", TRANSPORTATION SCIENCE, 12, 183-199, 1978.

AN EFFICIENT ALGORITHM FOR UPDATING THE BASIS IN BICOMPONENT LINEAR PROBLEMS

K.V. Kim, B.R. Frenkin, B.V. Cherkassky

Central Economic Mathematical Institute
USSR Academy of Sciences
Moscow

In this report we consider the bicomponent problem of linear programming. In such problems each variable enters not more than two constraints. A basis updating procedure of maximal efficiency is suggested for the problem. Special list structures used in the procedure enable us to scan only those basis elements whose characteristics are updated.

1. INTRODUCTION

This paper reports on an efficient algorithm for basis manipulation in generalized transshipment problems of linear programming (LP). These *bicomponent* problems form a special class containing not only flow problems but also LP problems having not more than two non-zero elements in each column of the constraint matrix. The paper suggests an algorithm for updating the basis for these problems with maximal efficiency.

The development of basis-updating algorithms is based on the assumption that the matrix of constraints is sparse. At first we tried to develop a basis-updating procedure requiring $0(m)$ instead of $0(m^2)$ operations, where m is the number of constraints. If the size of a problem increases, the percentage of nonzero elements of its matrix usually decreases, and only a small part of the basis variables change during one simplex iteration. Thus in solving large-scale problems it would be desirable to have a basis-updating procedure requiring fewer operations than $0(m)$. It is however evident that a lower bound for the number of operations in basis-updating procedures exists. Let X,Y denote primal and dual variables and ΔX, ΔY denote increments of these variables during one simplex iteration. Let d be the number of nonzero elements in vectors ΔX and ΔY. It is evident that we need no less than $O(d)$ operations to update the basis. An algorithm requiring $O(d)$ operations may be called *maximal-efficient*. It is interesting to know whether maximal-efficient algorithms for different classes of linear problems exist. The efficiency of the algorithm depends of course on the data structure used. Thus if ΔX and ΔY are stored as simple vector arrays, the algorithm with such data structure cannot be maximal-efficient because we need no less than $O(m)$ operations to find nonzero elements of these arrays.

Bicomponent problems form a class of LP problems for which a maximal-efficient algorithm for updating the basis exists. In this paper we present this algorithm.

2. THE STRUCTURE OF THE BASIS GRAPH

The design of efficient algorithms is based on the graphic representation of bicomponent problems. The nodes of the basis graph correspond to the rows of the basis matrix and the edges to the columns. Thus all basis information may be presented as basis graph characteristics. The endpoints of the basis edge are the row numbers of nonzero elements of the corresponding column. Vectors X and ΔX also correspond to edges. Dual variables Y and ΔY correspond to nodes of the basis graph. Scanning the vectors and all computations connected with these vectors may be interpreted as scanning the nodes and edges of the basis graph.

Updating the basis information during one simplex iteration begins with computation of the vector ΔX. Its nonzero elements form the set of edges we must scan; we shall call them *working edges*. At the end of the simplex iteration we compute the vector ΔY for changing the vector Y. The nonzero elements of ΔY form the set of nodes we must scan; we shall call them *working nodes*.

We need to have direct access to working nodes and edges, and can realize this direct access using the special structure of the basis graph. Generally, the basis graph of a bicomponent problem contains m nodes and m edges. The basis graph does not contain isolated nodes and the endpoints of each edge belong to our set of nodes. The basis graph thus contains one or more connected components, and each connected component contains just one cycle. The case where we have a one-element column in the source problem (for example, the unit column of the artificial basis) may be reduced to the general case. For the purpose of this reduction we must take an artificial node i_0 and form an artificial cycle with the help of the loop (i_0,i_0). In this case one of the connected components will always contain the artificial node. In a pure transshipment problem the artificial node is usually the root of the basis tree. Figure 1 shows the general structure of the basis graph.

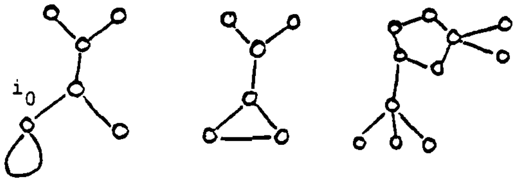


Figure 1. The structure of the basis graph

3. THE SET OF WORKING EDGES

The set of working edges is defined in a unique way after finding the new basis column. The new basis column corresponds to the new basis edge (k,j). In the basis graph the working edges form a *bicycle* structure closed by the new edge. The bicycle is a connected subgraph containing two cycles and the new edge and not having terminal nodes (Figure 2).

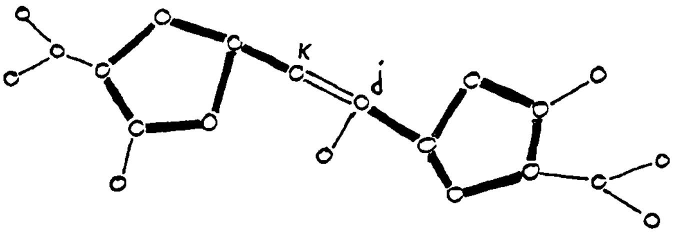


Figure 2. Forming a bicycle

Nonzero elements of X must be computed and the vector X must be changed during a linear scan of the bicycle edges. For performing such a scan, the structure of the basis graph is represented by a *forward list* (also called a *predecessor list*). This is an array of references p(i), where i and p(i) are the endpoints of the basis edge. If i does not belong to the cycle, then p(i) is the nearest node on the path from i to the cycle. If i belongs to the cycle, then p(i) is the next node on the cycle according to a particular cycle orientation. The reference p(i) points out the particular orientation of the basis edge (i,p(i)) (Figure 3).

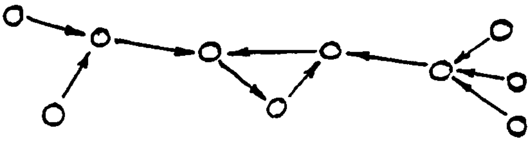


Figure 3. The orientation of basis edges

We are interested in the algorithmic sense of this orientation. Storing the basis graph with the help of the forward list, we point out directly only one endpoint of the basis edge, i.e., p(i). The other endpoint i is indicated by the i-th position in the array p. The edge (i,p(i)) thus becomes connected with the i-th node; that is, all information connected with this edge is addressed by the index i. This indication ensures direct access to the bicycle edges when we go down the references of the forward list. Let (k,j) be a new edge added to the basis graph. We begin to scan the bicycle from the node k. We label the node k equal to 1:$\ell(k):=1$. Going down the forward list, we execute the statement i:=p(i). The node i=p(k) is thus the next node that we should scan. This node gets a label equal to 2:$\ell(i):=2$, and so on, until we try to scan a node labeled earlier. We shall call this node s the *first closure.* We then begin to scan nodes from the node j going down the forward list up to the first labeled node denoted by t. The node t is the *second closure.* In this case the nodes being scanned get negative labels: $\ell(j):=-1$; $\ell(p(j)):=-2$, and so on. Depending on the relation between $\ell(s)$ and $\ell(t)$, we have three variants of the bicycle structure (Figure 4).

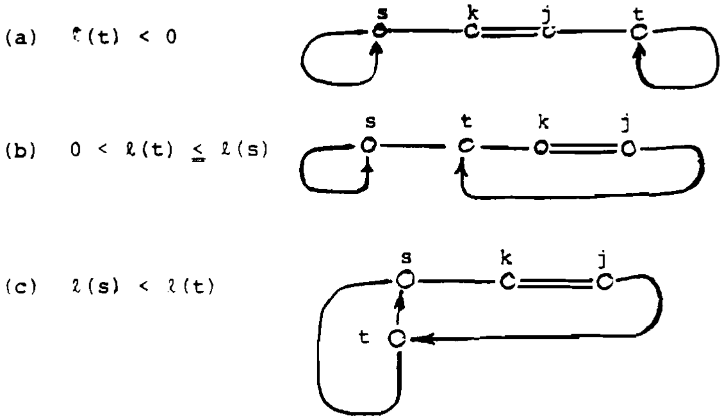


Figure 4. Three variants of the bicycle structure

Before constructing the bicycle, the new basis variable is given unit value. Other nonzero elements of ΔX are computed from the constraint equations when we scan the bicycle edges. These equations are not satisfied at the closures. Hence we must correct the vector ΔX on the edges of the cycle. To perform this correction, we must scan the cycles again. The remaining work necessary for changing basis variables and deleting an old edge from the basis graph may be performed by repeatedly going down the forward list from nodes k and j to closures s and t. It can easily be shown that for all the work described we need no more than $O(d_1)$ operations, where d_1 is the number of bicycle edges. To update the forward list, we must change the orientation of basis edges on the path from the new edge to the edge that has been deleted (Figure 5).

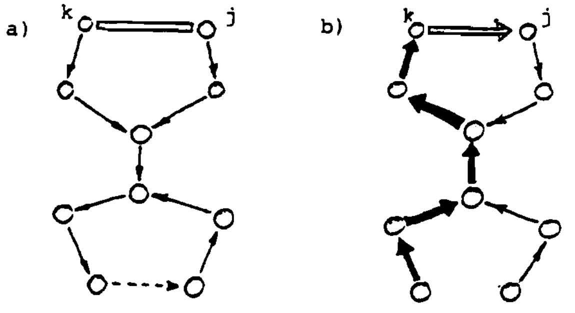


Figure 5. The reorientation of the basis edges

4. THE SET OF WORKING NODES

At the end of a simplex iteration we must change the vector Y. The elements of ΔY may differ from zero only in the nodes preceding the new edge; that is, going from these nodes down the forward list, we must pass the new edge. If we want to change the vector Y efficiently, we must store information about the predecessors in each node. Storing the information in direct form does not solve the problem because we need too much time to update this information. Special information about the predecessors incorporated in a *backward list* is used in efficient algorithms. Let us consider the backward list for one connected component. At first it contains an array of references q(i), which enables us to scan all the nodes of the component. Beginning from a node i and executing the statement i:=q(i), we should first scan the predecessors of i; this must be effected for each node. Figure 6 shows an admissible order of scanning. One can construct this order by moving in the basis graph according to the labyrinth rule and deleting the nodes being passed for the second time on the way back.

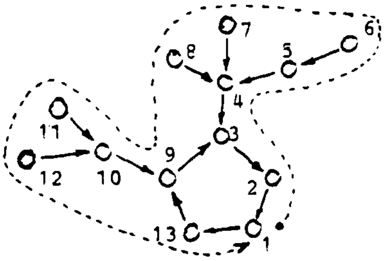


Figure 6. Going down the backward list

Let (k,p(k)) be a new edge in the updated basis graph. The set of working nodes precedes this edge. Thus we must start from node k to scan all working nodes. If the new edge belongs to the cycle, then all nodes of our connected component are working. In this case, we must go down the backward list back to the starting point k. If the new edge does not belong to the cycle, then the set of working nodes forms a branch rooted in k. In this case, we will eventually be able to stop going down the backward list before reaching k. To solve this problem, the backward list must contain not only the array of references, but also an auxiliary array f(i). There are many variants of information stored in this array, including, for example,

(a) f(i) -- the number of the last node of the branch rooted at i

(b) f(i) -- the number of nodes in this branch

(c) f(i) -- the distance from i to the cycle.

In our algorithm, f(i) is the degree of node i decreased by two. It can easily be shown that the sum of f(i) on the branch rooted at i is greater than or equal to zero if the branch is not completely scanned and becomes less than zero if the last node of the branch is scanned. Going down the backward list, we must thus sum up the values of f(i) and stop when the sum becomes less than zero. Significantly, updating this backward list does not take more than $0(d_1)$ operations during one simplex iteration. Let d_2 denote the number of working nodes. Changing the vectors X and Y and updating the forward and backward lists thus takes $0(d_1) + 0(d_2)$ operations. Hence our algorithm may be called maximal-efficient.

5. AN AVERAGE OPERATION NUMBER HYPOTHESIS FOR PIVOTING IN LARGE-SCALE PROBLEMS

The time required to solve large-scale problems depends on the number of simplex iterations. Generating a good starting solution for a bicomponent problem is not a very important difficulty because values of d_1 and d_2 for an artificial basis are very small and starting iterations are performed quickly. One simplex iteration consists of selecting an incoming variable by pricing and updating the basis. By using efficient algorithms, the time required to update the basis decreases considerably. The choice of pricing strategy is also important. As pointed out in [4], the choice of a good pricing strategy is an art for large-scale problems. We believe that this problem requires special research. For example, everyone knows that selecting the "most negative" variable to enter the basis is bad because we spend too much time on pricing.

It is interesting to estimate the average number of operations involved in a pivot. We suppose the number of operations depends on the branching of the basis graph. If the basis graph is a cycle, then it has no branches and $d_1 + d_2 = 0(m)$, i.e., the estimation is bad. If the basis graph is a star, then $d_1 + d_2$ does not depend on m and the estimation is good. In speaking about "average branching", we imagine a complete binary tree. If we suppose a complete binary tree to be a basis graph, then an average number of working edges is 0(log m). It can easily be shown that an average number of working nodes is also 0(log m). Thus we suggest the hypothesis that an average number of operations for pivoting in maximally efficient algorithms grows like log m.

6. THE NONCYCLING MODIFICATION OF THE ALGORITHM

We can easily exclude the possibility of cycling in flow problems. Transshipment and generalized transshipment problems are flow problems. The bicomponent problem is a flow problem if in each column one nonzero element is positive and the other is negative. In this case, the variables may be interpreted as a network flow with gains. The nonzero elements of ΔX may be interpreted as the correcting flow on the bicycle edges. The orientation of the correcting flow depends on the orientation of the new edge. The source and the sink of the correcting flow are the possible algorithm closures. The closures come at the source or the sink because of circulation in the basis cycle. If we go down the entire cycle with a certain orientation, then the flow increases and the closure is at the source. If we go down the entire cycle with another orientation, then the flow decreases and the closure is at the sink. As shown in Figures 7(a) and 7(b), if the new edge is oriented from k to j, then the closure s must be the source and the closure t must be the sink. Let us move around the bicycle from the source to the sink. Then cycling is impossible if two rules are followed:

1. In the starting basis, all degenerate flows are oriented to the cycle;
2. If we have several edges that we may delete, then we delete the first edge encountered in moving around the bicycle from the source of the correcting flow to its sink (Figures 7(a) and 7(b)).

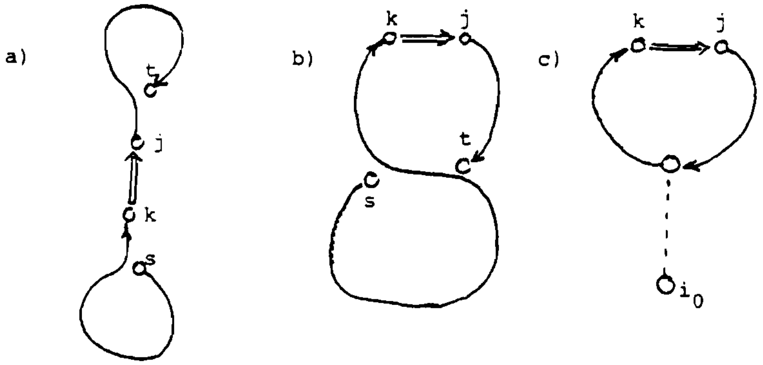


Figure 7. The order of search for the edge deleted from the basis graph

Rule 2 can be simplified for pure transshipment problems. In such cases, the basis graph is a tree and the correcting flow circulates in the cycle formed by the new edge. Thus we have just one closure, which is simultaneously the source and the sink of the correcting flow. In this case, Figure 7 shows the order of search for the deleted edge.

7. CONCLUSION

The algorithms described for the pure transshipment problem were developed in the USSR in 1972 for solving large-scale problems in the development and placement of plants, taking into account the cost of transportation. Linear and nonlinear problems were considered. Sometimes it was necessary to modify supplies, demands, and cost coefficients and to reoptimize many times. The maximal-efficient primal code was good for these applications. In 1974 the general description of maximal-efficient algorithms for bicomponent problems was presented in [1]. These algorithms are now used in many codes. In particular, they are implemented in a code library for solving transportation problems in PL/1 and FORTRAN. This library is popular in the USSR. The design of a noncycling algorithm [2] is more interesting from the theoretical than from the practical point of view; however, it guarantees the finiteness of the code execution. Similar efficient algorithms for pure transshipment problems have been presented independently in other papers, as, for example, [3] and [4].

REFERENCES

[1] Kim, K.V. 1974. About the efficiency of algorithms for solving bicomponent problems of linear programming. Economica i matematicheskiye metody 10(3):621-631. In Russian.

[2] Bronshtein, M.L., and K.V. Kim. 1979. Economica i matematicheskiye metody 15(2):408-412. In Russian.

[3] Glover, F., D. Karney, and D. Klingman. 1974. Implementation and computational comparison of primal, dual, and primal-dual computer codes for minimal cost network flow problems. Networks 4:191-212.

[4] Bradley, G.H., G.G. Brown, and G.W. Graves. 1977. Design and implementation of large-scale primal transshipment algorithms. Management Science 24(1):1-34.

SOME TECHNIQUES TO IMPROVE THE EFFICIENCY OF SOLVING LINEAR PROGRAMMING PROBLEMS*

U.H. Malkov, G.G. Padchin, N.A. Sokolov

Central Economic Mathematical Institute
USSR Academy of Sciences
Moscow

Further improvements to the simplex algorithm with the multiplicative form of the inverse and in obtaining greater efficiency in solving LP problems are possible in the following directions:

> Reducing the required iterations by using new fast algorithms to obtain an initial solution. We use an iterative algorithm which seeks a saddle point of the augmented Lagrangian and uses a vector of updated penalty coefficients.
>
> Taking into account the specific features of particular problems.

In this paper we first construct a special algorithm for obtaining an initial solution to an irrigation model. Second, in the framework of the multiplicative form of the inverse we implemented a specific simplex algorithm for the problem.

*Paper presented to the Workshop by K.V. Kim.

The simplex method is currently a rather efficient technique (in terms of running time, reliability, and the size of problems to be solved) for solving linear programming (LP) problems. According to specialists, the best algorithm (as far as running time, compactness, and accuracy are concerned) is based on triangular (LU) decomposition of the basis matrix and on the triangular multiplicative factorization of the inverse basis matrix.

We believe that it is possible to improve the multiplicative algorithm and to solve LP problems more efficiently by:

1. reducing the number of iterations required through using more powerful algorithms to obtain initial solutions
2. taking into account the specific features of a problem
3. increasing the reliability of the algorithm through using more efficient techniques to handle "ill-conditioned" problems.

One promising approach for obtaining a good initial solution is based on iterative algorithms. Professor E. Gol'shtein and his colleagues are studying iterative methods of solving LP problems at the Central Economical Mathematical Research Institute (CEMI) of the Academy of Sciences of the USSR.

Good results are achieved by using an iterative algorithm which seeks the saddle points of the augmented Lagrangian

$$F(x,y,\alpha) = \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} y_i u_i(x) - \tfrac{1}{2}\sum_{i=1}^{m} \alpha_i u_i^2(x) \quad ,$$

where

$$u_i(x) = \sum_{j=1}^{n} a_{ij} x_j - b_i \quad ,$$

x, y are primal and dual solutions, and $(\alpha_1,\alpha_2,\ldots,\alpha_m)$ are vectors of penalty coefficients.

An important detail in this algorithm is the use of a penalty coefficient vector α instead of an ordinary scalar coefficient. The penalty vector is recomputed during the iterations using information about the current solution of the primal and dual problems.

The steps of the iterative algorithm follow. A vector x^{s+1} is determined as an approximate solution of an auxiliary problem

$$F(x,y^s,\alpha^s) \rightarrow \max \quad , \qquad x \geq 0$$

by means of the alternative coordinate directions algorithm. The vector y^{s+1} is recomputed from

$$y^{s+1} = y^s - \alpha^s u(x^{s+1}) \quad .$$

The vector α^s is then recomputed. A more detailed description of the algorithm is given in [1].

The algorithm in question enables us to find an approximate solution to the primal and dual problems with an accuracy of up to 1% by a number of iterations comparable to that of the simplex method.

The iteration of an iterative algorithm is much simpler than that of a multiplicative algorithm; there is no need to store the inverse in the iterative algorithm.

Table 1 compares the iterative algorithm with the simplex method.

Table 1.

Size of problem	The Number of Iterations			
	Iterative algorithm*			Simplex Method
	$\varepsilon f = 10\%$ $\varepsilon R = 15\%$	$\varepsilon f = 3\%$ $\varepsilon R = 3\%$	$\varepsilon f = 1\%$ $\varepsilon R = 1.5\%$	
83 × 33	29	35	57	33
113 × 83	90	97	103	135
352 × 166	383	486	578	558

*εf and εR are the objective function and constraint tolerances, respectively.

The number of iterations for the simplex method is obtained starting from an all-slack basis.

To obtain an exact solution, it is possible to use the multiplicative simplex algorithm starting from the point given by the iterative one.

The question of how to construct an initial basis associated with the iterative solution for the multiplicative algorithm is not yet completely solved, but there is no need for a more accurate iterative solution. Improving an initial point by means of the iterative algorithm does not however at present always give fast convergence to an optimal solution in the multiplicative algorithm.

Evidently the subroutine for obtaining the initial solution must be fast and efficient. We have used a simple method to obtain an initial (usually infeasible) solution. We first bring all slack variables of a model into the basis. Then we pivot in the columns having nonzero elements in the iterative solution.

Even in such cases, the number of iterations required to obtain the optimal solution may be reduced to half that reouired using standard simplex pivot rules from the initial all-slack basis.

The special features of a problem may be taken into account in two ways. First, it is possible to construct a special algorithm to obtain a fast initial solution. Second, in the framework of a multiplicative algorithm we can use special features of a problem--generating the columns, for instance, instead of storing them.

These ideas were implemented in an optimal irrigation model. Generating the columns enabled us to increase the size of the solvable problems by four times and to decrease the running time by a similar amount. A special approximate algorithm for finding an initial solution decreased the number of iterations by an order of magnitude.

Let us consider the rectangular field with a mesh on it (see Fig. 1).

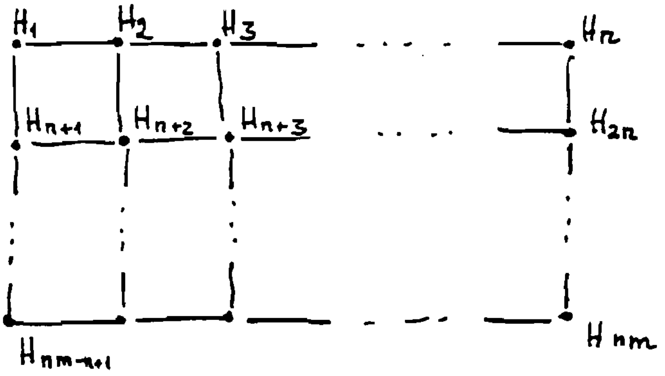


Figure 1

The values H_j are heights (evaluations) of the initial surface.

It is possible to find a feasible surface that gives the minimum total amount of ground transportation work (from the points of cutting to the points of filling) by solving the following mathematical programming problem.

The objective function

$$F = \sum_{i=1}^{nm} \sum_{\substack{j=1 \\ j \neq i}}^{nm} c_{ij} t_{ij}$$

must be minimized subject to the following constraints:

--longitudinal (vertical) slope constraints

$$\varepsilon_1 \leq z_j - z_{j+1} \leq d_1, \ j = 1,\dots,nm, \ j \not\equiv 0 \pmod n$$

--transversal (horizontal) slope constraints

$$\varepsilon_2 \leq z_j - z_{j+n} \leq d_2, \quad j = 1,\dots,nm-n$$

--transport variable value constraints

$$z_j - H_j = \begin{cases} \frac{1}{\gamma} \sum\limits_{\substack{i=1 \\ i \neq j}}^{nm} t_{ij} \ , & \text{if } z_j - H_j \geq 0 \\ - \sum\limits_{\substack{i=1 \\ i \neq j}}^{nm} t_{ji}, & \text{otherwise} \end{cases}$$

H_j are initial height marks, z_j are project height marks, the values ε_1, d_1, ε_2, d_2 define feasible intervals of the longitudinal and transversal slopes, and γ defines the balance of cuttings and fillings.

The field may be partitioned into several subfields, each subfield having its projection parameters ε_1^i, d_1^i, ε_2^i, d_2^i.

The problem under consideration comes up in projecting irrigated fields and building sites.

This linear programming problem has the structure

$$\min \left\{ ct \;\middle|\; \begin{array}{ll} Ax + y = a \ , & 0 \leq y \leq \bar{y} \\ \Lambda x = Dt & t \geq 0 \end{array} \right\}$$

where x: = z - H.

Here A has two nonzero coefficients in each row (+1 or -1), the matrix Λ is diagonal, and D is a matrix of the transportation problem.

Practical difficulties are of considerable size. For example, a field of 100 acres gives rise to the problem of about 4 000 rows and 500 000 columns.

A good initial solution may be obtained in the following manner [4], taking into consideration the specifics of the problem.

First, we solve an auxiliary problem

$$\min \left\{ \sum_{i=1}^{nm} \lambda_i x_i^2 \;\middle|\; Ax + y = a, \quad 0 \le y \le \bar{y} \right\},$$

by the group balancing algorithm [2] to define the initial values of the variables x and y.

The group balancing algorithm consists of the following steps. We scan the field cyclically, building the group for the next j-th point of the field. All the nodes around the j-th one are included in the group if they satisfy the slope constraints as equalities. Then we attempt to move the j-th node with its group upward if $z_j-H_j<0$ or downward if $z_j - H_j > 0$. If a new constraint comes into equality and does not allow further movement (upward or downward), a corresponding node is included in the group. We continue to move the j-th node with its group until a balance of cuttings and fillings in the group is reached. During the first scanning of the field, we include in the group the nodes that violate the constraints. The given algorithm converges fast, in 3-6 scannings. Initial values of transport variables are determined using quantities of t obtained by solving the following problem:

$$\min \{ct \mid \Lambda x = Dt \;, \quad t \ge 0\} \quad .$$

The approximate solution obtained is usually within 5% of the optimal one. The less rigid the constraints, the better the solution is. Further optimization by the multiplicative algorithm requires fewer iterations (by a factor of ten) than if we started from the "zero", i.e. all-slack, basis.

A specialized multiplicative algorithm, with a coefficient matrix and cost vector all kept algorithmically, is used to obtain the optimal solution. The matrix columns are split into four priority subsets. We look for the candidate to enter the basis first among the y-columns; if there is none, we then look among the transport columns t_{ij} such that $x_i (x_i < 0)$ and $x_j (x_j > 0)$ are in a current basis, among the x-columns, and finally among all the transport columns. Such regulation of the column generating procedure has produced significant gains,as searching for candidates to enter the basis is the most expensive operation in the multiplicative algorithm.

The code has a low running time and the process is reliable in obtaining a feasible solution. These properties are especially important when dealing with an ill-conditioned problem, where one could easily get out of the feasible solution set. Our experience has demonstrated that the algorithm where variables of an intermediate basis can be infeasible is more reliable. Such an algorithm is not much more complex than an ordinary one; it differs from the latter only in the pivoting rules.

We have used the following variant of the simplex method [5]. Let J_3, J_4 be sets of variables that are negative or over the upper bound, respectively. When a current basis is infeasible, we use the following objective function:

$$\max \sum_{j_k \varepsilon J_3} x_{j_k} - \sum_{j_k \varepsilon J_4} x_{jk}$$

instead of the initial one and the following rule for choosing the pivot row:

$$\theta = \min \{\theta_1, \theta_2, \ell_k\}$$

$$\theta_1 = \min_i \left\{ \frac{x_i}{z_{ik}} \mid z_{ik} \neq 0, \frac{x_i}{z_{ik}} > 0, \ j_i \not\subseteq J_4 \right\}$$

$$\theta_2 = \min_i \left\{ \frac{x_i - \ell_{ji}}{z_{ik}} \mid z_{ik} \neq 0, \frac{x_i - \hat{x}_{ji}}{z_{ik}} > 0, \ j_i \not\subseteq J_3 \cap J_4 \right\}$$

j_i is the number of a basis variable, x_i is the value of a basis variable, and $\hat{x}_j$ is the upper bound of the value of a variable x_j.

$$z_{.k} = \begin{cases} B^{-1}\, a_{.k} & \text{if } x_k = 0 \\ -\, B^{-1}\, a_{.k} & \text{if } x_k = \hat{x}_k \end{cases}$$

is the transformation of a column $a_{.k}$ entered in the basis.

References

[1] Sokolov, N.A. (1980) An Iterative algorithm for solving the primal and dual linear programming problems using an augmented Lagrangian. Numerical Methods of Mathematical Programming, edited by V.A. Skokov. Moscow: CEMI. In Russian.

[2] Korobochkin, M.I. (1974) A group balancing algorithm for solving special models of quadratic programming. Software Systems for Solving Optimal Planning Problems, edited by U.H. Malkov. Moscow: CEMI. In Russian.

[3] Borisova, E.P., E.G. Golstein, N.A. Sokolov and N.V. Tretiakov (1980) An iterative algorithm of linear programming in package "PAOEM ES." Software Systems for Solving Optimal Planning Problems, edited by U.H. Malkov. Moscow: CEMI. In Russian.

[4] Malkov, U.H., G.G. Padchin (1980) About solving the optimal irrigation model by the quadratic and specialized algorithms. Software Systems for Solving Optimal Planning Problems, edited by U.H. Malkov. Moscow: CEMI. In Russian.

[5] Shikhov, V.V. (1980) An experimental investigation of two techniques for obtaining feasible solutions in LP problems. Software Systems for Solving Optimal Planning Problems, edited by U.H. Malkov. Moscow: CEMI. In Russian.

MODEL GENERATION AND STRUCTURE IDENTIFICATION

TOWARD SUCCESSFUL MODELING APPLICATIONS IN A STRATEGIC PLANNING ENVIRONMENT

Johannes Bisschop and Alexander Meeraus

*Development Research Center**
The World Bank
Washington, D.C.

The role of models and their success in a strategic planning environment are evaluated, and current limitations on modeling applications are examined. In order to relax many of these limitations we propose a basic change in modeling technology. Although some of the underlying ideas are currently implemented at the Development Research Center of the World Bank, a unified effort by the entire modeling community and software industry is needed to finally deliver a significant improvement in the quality and success of strategic modeling applications.

*Views expressed are thosa of the authors and do not necessarily reflect those of the World Bank.

1. Introduction

The audience at which this paper is directed consists of both the modeling community and the developers of algorithms and software. Experienced model builders will acknowledge the frustrations they have suffered practising their trade, while students of model building will become better aware of some of the resource constraints encountered in applied modeling exercises. All of them will have a definite interest in recognizing ways to relax these resource constraints. Algorithm and software developers are also an important part of the audience as their joint but unified contributions will be required for the eventual resolution of the fundamental issues raised in this paper.

The main focus of the paper is on modeling in a strategic planning environment. Section 2 will elaborate on what we mean by such an environment, and what is meant by "success" in the application of models in strategic planning. In section 3 we examine the current limitations on modeling, emphasizing not only the extensive resource requirements in terms of technical skills, money and time, but also such intangible issues as the low reliability associated with our present model generating software, and the awesome communication problems associated with the dissemination of models and their results. Section 4 sets out some fundamental steps that will be required for a significant increase in successful modeling, while sections 5 and 6 elaborate on some of the developments that we have begun in taking these steps.

2. The Role of Mathematical Models in a Strategic Planning Environment

In order to have a common understanding of what we mean by "mathematical models" and a "strategic planning environment," we need to classify models in relation to the environment in which they are used. We view mathematical models essentially as mappings: each model transforms a set of input data into a set of output data. With such a general definition in mind, it is no easy task to classify models and their use. Anyone punching a calculator is using a model. A linear program to run a refinery is a model. A management information system is a model. A mathematical program capable of evaluating water-related investments in a third-world country is a model. These examples represent models that are used in different application environments, and each model has its own characteristics.

One can identify a wide spectrum of models with operational models on the one extreme, and strategic planning models on the other. Operational models can be characterized as "black boxes." Their users are not interested in the model itself. Only the results produced by the model are of interest. Operational models are used over and over again, each time with different parameter inputs. No structural changes are ever made to these models, which makes them essentially static in nature. Strategic planning models, on the other hand, can be characterized as "open boxes." Their users are primarily interested in how the model is constructed. Strategic planning models are used only once, and their results serve to further the understanding of the model. Structural changes are continually made, which makes these models dynamic in nature.

The calculator is clearly an example of an operational model. It is an hard-wired device (usually contained in a black box), which can perform a set of well-defined tasks. A typical user is interested in the results it produces, and wants to use it over and over with different input data. No modifications are even made to the calculator itself. The linear program to run a refinery is mostly an operational model. Its structure is fixed most of the time, and it is used over and over again to determine the operation of the refinery. The management information system is somewhere in the middle of the spectrum. Whenever it is used to provide factual information to management it represents an operational model. Whenever it functions as a decision support system, capable of analyzing information, it represents a strategic planning model. The mathematical program capable of evaluating water-related investments in a third-world country is clearly an example of a strategic planning model. The understanding of the model is much more important than the results it produces. Structural changes to the model will be made as a result of enhanced insights into both the model itself and the real world it is designed to capture.

In this chapter we want to focus on the role of models in a strategic planning environment. Such an environment is characterized by long-term, often ill-defined and poorly understood issues which require near immediate decision making. It is the long-term impact of the decisions that make them important. Examples of strategic planning environments are government planning agencies, corporate planning offices and international organizations.

These planning environments have common characteristics. The issues under consideration are usually extremely complex, and need to be sorted through. The amount of possibly relevant information is vast. In addition, the consequences of any decision are not necessarily limited to one person or one institution. Nor are all other aspects of the decision necessarily under the jurisdiction of one person or one organization. In such an environment, mathematic models play a special role. They are used as a framework for analysis, for data collection and for discussion. They are created to improve one's conceptual understanding of the problem. If several decision makers and/or institutions are involved in a final decision or set of recommendations, models can be used as neutral moderators to guide the discussions. Different viewpoints can be tested and examined. In such an environment the actual values of model results are not so important, but the relative values resulting from testing different scenario's are of interest. The model is a learning device, and should never be expected to produce final decisions. Because of this indirect importance of a model in a strategic planning environment, there is no clear way to measure the benefits, although it is not too difficult to keep track of the (usually high) costs. It is precisely this lack of well-defined monetary benefits and the fact that planning models are continuously changing that distinguish them and their environment from the operational models discussed previously.

Having characterized models and their roles in a strategic planning environment, we can now define the meaning of success of a modal. An operational model is successful if it produces raliable results, and it is easy to operate. A strategic planning model is successful if it is easy to understand the model, if its structure and content can be communicated effectively to others, if the results produced by the model can be explained, and if model experiments can be easily repeated or verified by experts other than the original model builders. Referring to two of our previous examples, the calculator should be easy to use, and the refinery model should be able to control the refining process effectively for them to be successful models. The requirements for the success of strategic planning models are much higher than the ones for the success of operational models. This has undoubtedly contributed to the limited role that mathematical models have played thus far in a strategic planning environment. The next section will highlight some selected aspects of our current modeling technology to illustrate this point.

3. Current Limitations on Modeling Applications in a Strategic Planning Environment

In the early days of mathematical modeling, large applications were mostly of a military or industrial nature. Models were used to describe and solve well-defined problems in the areas of production and distribution, and they were employed on a routine basis. In many instances it was considered cost-effective to establish a small group of technical people whose sole responsibility was to maintain and to improve the existing package of models. In recent years the scope of mathematical modeling applications has widened, and modeling environments different from those described above have emerged [1], [2], and [3]. The U.S. Government, for instance, has supported the development of a large number of strategic models, and many planning agencies around the world use mathematical models as their major tool for analysis. In these planning oriented environments we have observed that the cost of building and maintaining mathematical models is high, while the benefits are not always clearly defined.

A study by the National Science Foundation on the development and use of mathematical models within the U.S. Government provides some interesting figures [2]. The total development cost of the 650 models surveyed was US$100 million ($154,000 per model), and it took on the average 17 months to make a model operational. It was observed that 75% of all models can be operated only by the original development team, despite strong efforts in model and program documentation. Actual policy use of these models by groups other than the model designers has been minimal. Given the median size of 25 equations (only 6 models had more than 1,000 equations), the above figures look rather depressing as it takes 3 weeks and $6,000 to develop one equation on the average.

Our own experience in the World Bank indicates that a large portion of total resources currently spent on large modeling exercises is for the generation, manipulation, and reporting of these models. It is evident that this percentage must be reduced significantly if models are to become effective tools in planning and decision making.

Besides these extensive resource requirements we have encountered several other problem areas, most of them stemming from attempts to disseminate previous and ongoing research in a planning environment. The documentation of large models and their modifications is one such problem. If a project is large, and continues for one or two years, the cost of complete documentation becomes horrendous. A decision is usually made to maintain a few versions of a model. In practice this means that some basic experiments can be repeated. In the long run, however, the value of the available software becomes essentially zero as people change jobs, and any changes to existing versions require extensive set-up time.

A related problem is the communication of models to interested persons that are not part of the development team. As there are no standards in notation, it is often difficult to judge from any write-up what exactly the model is. Experimentation with the model may enhance one's understanding, but this requires the use of both the model and report generators. As these programs are nontrivial, they in turn require the use of a technical person. The extensive time and money requirements prohibit many outsiders from even attempting to satisfy their own curiosity with regard to the model. No effective dissemination of knowledge can therefore take place.

Another major obstacle to successful modeling in a planning environment is that there does not exist a common interface with the various solution routines modelers can use for their family of models. As each solution package usually requires different data structures, it becomes both time and money consuming to switch back and forth between solution algorithms. As a result models tend to get locked into one solution package which at times limits their development. There is also no general-purpose software for the linking of models, an activity that has become more prevalent with the increased use of models.

The heart of the problem is the fact that solution algorithms need a data structure and a problem representation which is impossible to comprehend by humans. At the same time, problem representations that are meaningful to humans, are not acceptable to machines. The two translation processes required can be identified as the main source of difficulties and errors. With today's technology, each translation process is broken down into a number of interrelated steps where most of the coordination and control has to be done by humans, and is therefore subject to error. That's why extensive time, skill and money resources are required for the completion of large-scale modeling excercises. In addition, it is not surprising that because of this extensive human input the overall reliability (the probability of no mistakes) of our modeling practice is embarrassingly low.

We would like to illustrate the above paragraph by using linear programming as an example, surely the most widely used and best developed tool available today. Most students involved in quantitative studies are exposed to linear programming and its applications through textbook examples, which can be comprehended quite easily. Still, many of them find tremendous difficulties in handling real-world linear programming applications! The reason for this is simply size. If one uses textbook methodology, one finds that the complexities associated with the generation and manipulation of models grow astronomically with size. Consider a small problem with 10 equations and 10 variables. This can be neatly printed in matrix form on one page. We can directly inspect each of the 100 numerical entries and their position relative to each other. To print the matrix of a "standard" size agricultural sector model with 1,000 equations and

2,000 variables, on the other hand, would require 3,200 pages of computer paper. Realizing that of the 2 million possible entries only 80,000 or so are different from zero, we could label rows and columns and print these labels together with the non-zero entries. Although this way of representing the matrix reduces the required pages to 1,230, we are essentially left with a list of seemingly random numbers, unable to discover any meaningful model. Unfortunately, this is exactly the way we have to communicate with today's software.

Table 1 is a further elaboration on linear programming technology. The solution process is broken down into 12 different *tasks or processes* and 15 classes of associated *documents or data files*. As an illustration of how to interpret the table, consider the third row. The task is described as "design computer program to generate column/row/value records corresponding to model in matrix form." It can only be performed by a human, and it requires three inputs and one output for its completion. One necessary input is the description of data and model in conventional notation. On the basis of this input, one has to design MPS naming conventions that will be used in the naming of rows and columns of the linear programming tableau. Added to this input will be a data set coded in a program acceptable form. With these inputs the task can be executed, and the final output, a matrix generator program written in some language, will result. Remember that each input and output in the table requires human intervention. The final goal, of course, is the solution report while the 13 intermediate documents are an expensive and error-prone detour. It is important to note that the first 7 tasks are performed by humans. The last 5 tasks are performed by the machine, but need additional

Table 1: Tasks and Documents Associated with Current Technology to Solve LP Models

Documents (files) / Task (process)	Data and model in conventional notation	MPS Naming conventions	Code book	Matrix generator *	Report generator *	Basic data *	Matrix generator (object)	Report generator (object)	MPS Matrix file *	LP control program *	Restart basis	Revision file *	LP-report	Solution file	Solution report
Formulate coding rules for row and column labels (human)	INPUT	OUTPUT													
Formulate rules for data representation (human)	INPUT		OUTPUT												
Design computer program to generate column/row/value records corresponding to model in matrix form (human)	INPUT	INPUT	INPUT	OUTPUT											
Transcribe data (human)	INPUT					OUTPUT									
Design solution strategy (human)	INPUT									OUTPUT					
Design report program (human)	INPUT	INPUT	INPUT		OUTPUT										
Generate revision files (human)	INPUT	INPUT	INPUT	INPUT	INPUT	INPUT						OUTPUT			
Compile matrix generator (computer)				INPUT			OUTPUT								
Compile report generator (computer)					INPUT			OUTPUT							
Execute matrix generator (computer)								INPUT	OUTPUT						
Execute LP-system (computer)									INPUT	INPUT	OUTPUT INPUT	INPUT	OUTPUT	OUTPUT	
Execute report generator (computer)						INPUT								INPUT	OUTPUT

* Readable by both human and machine.

control instructions to coordinate input and output. This again is a source of error, even though the execution of these 5 tasks is for all practical purposes error free. Many errors that are made are usually of a very intricate nature, and do not become apparent immediately after they have been committed. They are often carried on throughout the process without having fatal effects on solution procedures.

Although our present modeling technology has proven to be adequate in many modeling environments, it is clearly inadequate in meeting the needs of model builders in a strategic planning environment. A new generation of modeling technology is needed in order to make a significant improvement in the thus far limited success of planning models.

4. Relaxing Current Boundaries on Successful Modeling in Strategic Planning

Mathematical models are a potentially powerful tool in a strategic planning environment, but their effective use and dissemination have been hampered seriously thus far by excessive resource requirements in terms of time, money and technical skills, and by the difficulties associated with the communication and repeatability of model experiments. Existing modeling technology is a major limiting factor in this environment where models are always subject to structural changes, and at most one run of any model version is of interest. As we noted in the previous section, the heart of the problem is the fact that two representations of the same model are needed before any solutions can be obtained, and that the required translations are labor intensive and error prone. In our opinion there are two basic requirements that will eventually lead to the success of strategic modeling.

First of all we need a universally accepted, highly sophisticated notation, a formal modeling language, which is easily understood by both humans and the machine. Secondly we need a universally available software system with a low level internal data structure so that, at least in principle, it can interface with any data base and solution algorithm that is available. It goes almost without saying that these are necessary and not sufficient conditions for successful modeling, although their fulfillment should greatly enhance our modeling capability.

Determining a general modeling language is not an easy task, but its eventual definition should be guided by observed needs. Most problems associated with model building can be reduced to a basic question concerning communication. How can one communicate data and its associated complex mathematical structures when the human mind is limited in its power to grasp and comprehend many issues simultaneously. The only tool available to us is our power of abstraction which aids us in understanding the complexity of real world phenomena. It allows us to define partitionings, mappings, nestings, and short-hand notation. A modeling language should provide us with such a short-hand notation, allowing for the specification of partitionings, mappings and nestings in a unifying but easy to use manner. As almost everyone has been exposed to some algebraic notation during their formal education, the language should adopt as much as possible the existing algebraic conventions, with perhaps a few additions to handle the inherent complexities of large models. Since the language will only be used to represent models, it should not be an algorithmic programming language, but instead resemble more the language of a sophisticated data base capable of handling symbolic algebraic relationships.

The existence of such a general purpose modeling language carries several important implications with it. As it is machine readable, only one document is needed. This document serves also as the complete documentation of the model. Since it is easy to inspect, commmunication between persons is greatly enhanced. In addition, the machine can be of great help to the model builder in discovering misspecifications, especially at the syntactical level.

A universally available software system which can interface with data bases and solution algorithms also carries important implications for the modeling community. All of a sudden, many tasks that are currently performed by humans during the model building process, are automatically performed by the machine. This leads to a dramatic improvement in the reliability of our modeling software. Solutions and reports will be automatically generated as soon as a complete model representation has been specified. No more highly skilled computer technicians are needed. The time that is required to accomplish any structural changes in a model representation is suddenly becoming negligent. In addition, as many human errors are prevented by this new machine-intensive technology, the overall cost per model should decrease.

An important byproduct of an algebraic modeling system is that the algebraic representation contains structural information about the model which can be recognized by the system. The automatic detection of linear and nonlinear equations is one example. The automatic detection of block structures in the incidence matrix is another example. A general modeling system will also aid the development of algorithms when it comes to testing and comparing them. In addition, the system can be used as a marketing device for well-implemented algorithms, thereby reducing the distance between software developers and model builders.

It is evident that collaboration by various professionals and organizations is needed to ultimately accomplish the two basic tasks outlined in this section. At the Development Research Center we have made the first steps toward the development of a general algebraic modeling language and a general algebraic modeling system which we refer to as GAMS. We hope that this development, which has been under way for several years, will become a take-off point for future collaboration and possibly standardization in modeling software.

5. The Development of a General Algebraic Modeling System (GAMS)

The modeling environment within the World Bank can in general be characterized as a strategic planning environment, where models are built and used as a learning device and a framework for analysis [1]. Over the years we have been associated with many modeling exercises, and have recognized the limitations of our current modeling software. It is the need for a basic change in modeling technology that has led to the inception of the GAMS project. This project has progressed to the point where we represent complete models within GAMS, and execute all data manipulations. Links with selected solution algorithms are still under development.

The system is currently used to formulate and document several of our models. It guides and checks the user in the specification of a complete model, thereby accelarating the formulation process. In addition, all data manipulations can be performed by the system, and their results can be displayed in the form of well-documented reports. The layout of these reports are automatically generated by the system. An extensive analysis of the input data can therefore be made prior to any attempt to solve the model. In our experience the data preparation and analysis have always been the most time consuming aspects of any modeling exercise. The efforts required for this phase are reduced substantially with the use of the system. For those solution routines that are not linked to GAMS as yet, we use the GAMS notation as a guide for writing the model generator (i.e., the program that generates the model representation as it is required by a specific solver). This has resulted in an increased reliability of the model generators.

The choice of notation in GAMS grew from an increased understanding of both the needs of the model builder and the shortcomings of available computer languages. Most common programming languages are designed to implement algorithms, and do not allow for a minimal and easy to read representation of large and complex models. There are, however, some specialized model generating languages, usually designed around a specific algorithm. The DYNAMO language, for instance, was designed around an efficient algorithm to integrate dynamic systems. The TROLL system was designed around a Gauss-Seidel process to solve econometric models. Languages such as MAGEN and MGG were designed around the simplex method for linear programs. Both MAGEN and MGG are essentially a short-hand notation for generating the MPS tape (the industry-wide standardized input for linear programming software), and, as such, are limited in scope. They are rightfully called "matrix generators," as they are only suited for linear problems. In addition, since their main task is to manipulate characters (strings) to put together names for an MPS tape, they also do not allow for a minimal and easy to read representation of large models. They are, however, a step forward when compared to ordinary programming languages, and have been used successfully in environments other than strategic planning.

In our experience, the typical model builder wants to be freed from any burdens that are imposed by solution algorithms. His basic needs are served by both a notation that allows him to write down a model in a straightforward manner, and a system that takes over the steps required for the generation of the model and its results. Knowing that persons from many disciplines have been introduced to some mathematical notation, and that elementary data base notions are easy to understand, we have choosen for an algebraic language with data base concepts interwoven. The result is a flexible and easy to use notation, that is powerful enough to handle complex models. In addition, it can be used for many types of mathematical models both linear and nonlinear.

The data model used in GAMS is designed to take advantage of sparseness. Only the nonzero or the explicitly defined elements are stored internally. With this data structure the efforts required to perform any logical and algebraic data manipulations can be reduced to the sorting and merging of multidimensional files.

There are several tasks that we envision the system to perform. Besides the capability to interface with outside data bases and solution routines, the system should facilitate the coupling of models. Simulations over time may be such that the solution of a model in one period is used to determine a parameter of the model in the next period. This capability to link models will reduce the extensive set-up time that is required with current modeling technology. Other tasks that we envision the system to perform are automatic unit analysis and automatic scaling. Especially the latter one will become important as the distance between the model builder and the solution algorithm will grow with time.

Figure 1 gives a simple schematic overview of the structure of GAMS. Central in this figure is the GAMS translator. Following the translation the system either creates or continues a GAMS Project File. This allows a user to add or modify an existing model without requiring the system to repeat previous operations. The responsibility for the project data file lies therefore with the user. The responsibility for the GAMS Data Base, on the other hand, lies with the system. It contains information that is available and of interest to a variety of users and models. The technical norms that exist for each process in the fertilizer industry describing the input quantities required for one unit of output form one example. The GAMS Executor interfaces with both the Project File and a large variety of stranger systems. Examples of stranger systems are the GAMS evaluator (which evaluates all data expressions), the GAMS Model Analyser and the Linear Programming Executor. Each of them may in turn interface with other such stranger systems. The overall GAMS system is set up such that it can always expand as the need arises.

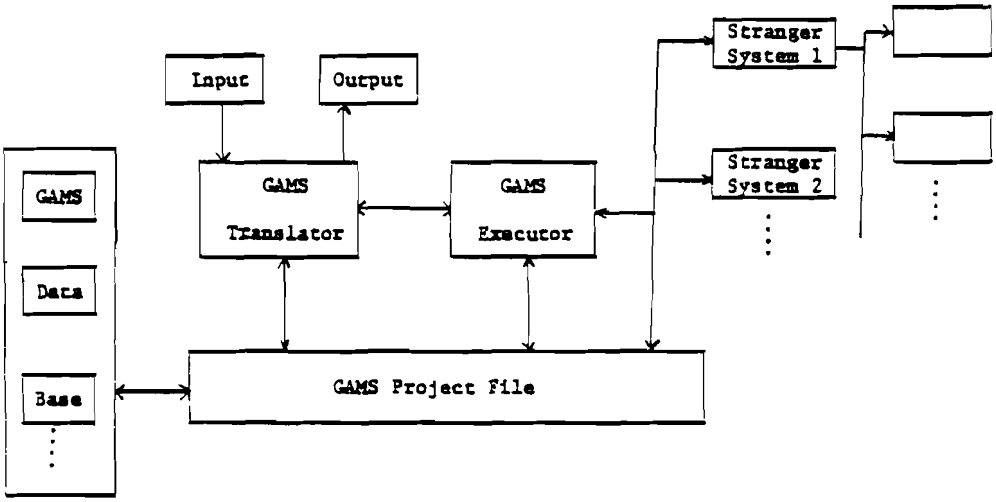


Figure 1: Schematic Overview of GAMS

Following this general discussion on the development of GAMS, we will devote the next section to some more detailed aspects of the modeling language in GAMS.

6. Selected Aspects of the Modeling Language in GAMS

This section is different in flavor from the previous sections in that it does not concern itself with generalities. Instead it concentrates on language details to provide the reader with some intuitive feel of what the language is all about. For illustrative purposes, consider the cannery transportation problem taken from the book, Linear Programming and Extensions by G. B. Dantzig. A company desires to supply its three warehouses from two canneries with given inventories in each, and wants to minimize the total shipping cost. The computer readable GAMS representation of this problem is stated in Figure 2.

As can be noted from the model description, we have restricted ourselves to a small character set which is available on most computers. In addition, we have assumed that there is no carriage control available (i.e., no subscripts or superscripts), and that there are only capital letters. Within these few limitations, we have adhered as much as possible to existing mathematical conventions.

This model statement can be viewed as an integrated data base. In addition to the data tables and assignment statements, there are the symbolic equations which represent data that can only be obtained via some solution algorithm. Both data and symbolic equations are needed for a complete model representation in GAMS.

There are several key words used in the model description. They are (in order of occurrence) SET, PARAMETER, TABLE, VARIABLE(S), EQUATION(S), SUM, MODEL, SOLVE..USING..MINIMIZING, and DISPLAY. We will comment on each of them.

```
SET  C  CANNERIES / SEATTLE, SAN-DIEGO /;
SET  W  WAREHOUSES /

     NEW-YORK
     CHICAGO
     KANSAS  KANSAS CITY /;

PARAMETER  A  AVAILABLE INVENTORIES (CASES OF TINS PER DAY) /

     SEATTLE    350
     SAN-DIEGO  650 /;

PARAMETER  R  REQUIRED INVENTORIES (CASES OF TINS PER DAY);

     R(W) = 300 ;

TABLE  UTCOST  UNIT TRANSPORT COST (DOLLARS PER CASE)
*              FROM CANNERY  C  TO WAREHOUSE  W

                 NEW-YORK        CHICAGO        KANSAS

SEATTLE            2.5             1.7            1.8
SAN-DIEGO          2.5             1.8            1.4

VARIABLES  X        SHIPMENTS (CASES OF TINS PER DAY);
           TRCOST   TOTAL TRANSPORT COST (DOLLARS PER DAY);

EQUATIONS  SUPPLY   AVAILABILITY CONSTRAINT (CASES OF TINS PER DAY)
           DEMAND   REQUIREMENT CONSTRAINT (CASES OF TINS PER DAY)
           COST     COST ACCOUNTING EQUATION (DOLLARS PER DAY) ;

*   AVAILABILITY CONSTRAINT IMPOSED ON EACH CANNERY

    SUPPLY(C)..

        SUM(W, X(C,W))             =L=          A(C);
*   - - - - - - - - - - - -                  - - - - - - - -
*   TOTAL SHIPMENTS LEAVING                  AVAILABILITY AT
*   CANNERY  C                               CANNERY  C
*   - - - - - - - - - - - -                  - - - - - - - -

*   REQUIREMENT CONSTRAINT IMPOSED BY EACH WAREHOUSE

    DEMAND(W)..

        SUM(C, X(C,W))             =G=          R(W);
*   - - - - - - - - - - - - - -              - - - - - - - -
*   TOTAL SHIPMENTS ARRIVING                 REQUIREMENT AT
*   AT WAREHOUSE  W                          WAREHOUSE  W
*   - - - - - - - - - - - - - -              - - - - - - - -

*   COST ACCOUNTING EQUATION REFLECTING TRANSPORT COST

    COST..

*           SUM((C,W), UTCOST(C,W) * X(C,W)) =E=  TRCOST;
*           - - - - - - - - - - - - - - - - -
*                  TOTAL TRANSPORT COST
*           - - - - - - - - - - - - - - - - -

MODEL  CANNERY  THE CANNERY TRANSPORTATION MODEL / ALL / ;

SOLVE  CANNERY  USING  LP  MINIMIZING  TRCOST ;

DISPLAY  X.L, SUPPLY.M ;
```

Figure 2: The Cannery Model in GAMS

Sets are used as driving indices in many mathematical models. They usually have a short name followed by a description. Following the description is a listing of the set elements contained between two "slashes." Each name can have an associated description if needed (e.g., the element KANSAS has a description KANSAS CITY).

A parameter can be defined in a similar fashion, with a number following each label as we did for parameter A. An algebraic definition using an assignment statement is also possible, and this was done for parametar R (each warehouse requirement is 300 units). A third way to define a parameter is via some tabular arrangement as was done for the parameter UTCOST. Both row and column descriptions of the parameter are required. As we shall see later this two-dimensional framework can be used to represent parameters with more than two dimensions attached to them.

Variable and equation names must be defined first before they can appear in any symbolic equations. One can recognize a symbolic equation by the two dots following the equation name. Note that the availability constraint SUPPLY is defined over the domain (set) C. It is a short-hand notation for two availability constraints, namely one for each cannery. The summation in the SUPPLY equation is indicated by SUM, and followed by the set name W to which the summation operation is to be applied. Each symbolic equation in GAMS has a type. In the example we have =L= (a less than or equal to constraint), =G= (a greater than or equal to constraint), and =E= (an equality constraint).

A model in GAMS is the selection of a subset of the symbolic equations. In the cannery example all equations are included in the model. Once a model is defined, a particular algorithm must be chosen. In this case linear programming (LP) is selected to minimize the variable TRCOST in the model CANNERY. Display statements can be used to get selected pieces of data. Here we have asked for the activity levels associated with the variables (X.AL), and the shadow prices (marginal costs) associated with the availability constraints (SUPPLY.MC).

The cannery model serves as a quick overview of several important aspects of the language in GAMS. The example does not portray some of the complexities associated with the representation of large-scale models. That is why a more extensive description of the notation in GAMS is presented next.

6.1 Sets and Set Mappings

A simple (one-dimensional) set in GAMS is a finite collection of labels. These sets play an important role in the indexing of algebraic statements. The cannery example contains two such simple sets (namely, C and W), and both their syntax and use are illustrated there. Several one-dimensional sets can be related to each other in the sense that there is a correspondence between them. As an example consider the correspondence between countries and regions. Depending on one's viewpoint, this is a one-to-many or one-to-one correspondence. To each country corresponds a specific set of regions, while each region corresponds to one specific country only. As we shall see, these correspondences play an important role in GAMS since they can be used to control the domain of definition in assignment statements and symbolic equations.

The syntax for set correspondences is much like the one for single sets. Consider the following illustration.

```
SET  CR  COUNTRY-REGION CORRESPONDENCE /

     INDONESIA.N-SUMATRA
     INDONESIA.N-JAVA
     MALAYSIA.W-MALASIA
          .
          .                   /;
```

or,

```
SET  CR  COUNTRY-REGION CORRESPONDENCE /

     INDONESIA.(N-SUMATRA, E-JAVA), MALAYSIA.W-MALAYSIA, .../;
```

Note that the period is used as an operator to relate the elements of the different sets, and that the order of the elements in the correspondence is fixed (in this case country first, region second). In order to reduce unnecessary repetition, the parentheses can be used when several elements in one set correspond to a single element of the other set. There can be any number of sets in a correspondence. The following few lines illustrate a 3-dimensional set mapping.

```
SET  RZD  REGION ZONE DISTRICT MAPPING /

     NORTH.IRRIGATED.(W-NORTH, C-NORTH, E-NORTH)
     CENTRAL.(IRRIGATED.(NW-UPPER, NE-UPPER)
              RAINFED.(S-UPPER, W-LOWER, E-LOWER))
                .
                .                                   /;
```

There are ways to change the information contents of sets and set mappings. This can be done via algebraic assignment statements, which require all sets to be indexed. Assume that a set R of regions has been defined, and that a copy of this set is desired. Then one can write the following GAMS statements.

```
SET  R  COPY OF SET R ;  RR(R) = R(R) ;
```

The next example is a redefinition of RR on the basis of the above set correspondence RZD. Assume that the new set RR should contain all regions that are not rain-fed. The instruction SUM, already mentioned in the cannery example, denotes a union instead of a summation when applied to sets.

```
RR(R) = R(R) - SUM(D, RZD(R, 'TROPICAL', D)) ;
```

Note that the 3-dimensional correspondence RZD requires 3 driving indices. Since the middle index is invariant, we have used the quotes to indicate a specific element rather than the entire set.

6.2 Data Tables

Tabular arrangements of data are a very convenient way to describe multi-dimensional parameters. The unit cost table in the cannery model is an example of a 2-dimensional parameter. The following table illustrates a 4-dimensional parameter, where 3 dimensions are captured in the row descriptions, while the fourth dimension is contained in the column label.

```
          TABLE  L  LABOR COEFFICIENTS IN HOURS PER RAI
*                   BY REGION, CROP ROTATION, TECHNOLOGY AND MONTH

                                   JANUARY  FEBRUARY  MARCH  APRIL

NORTH-UPP.SUGARCANE.TRAD-BUFF         2         2       2      12
NORTH-UPP.SUGARCANE.MOD-TRACT         1         2       2      10
              .
              .

+                                    MAY      JUNE     JULY  AUGUST

NORTH-UPP.SUGARCANE.TRAD-BUFF        12        35       30     45
NORTH-UPP.SUGARCANE.MOD-TRACT        12        30       30     40
              .
              .
```

Note that we have specified the units for the entire table in the table heading. As it stands at the moment, unit analysis has to be done by the model builder, although one of our goals is to make automatic unit analysis an integral part of the data base system in GAMS. The order of the sets used in the row and column descriptions in the table statement must be maintained in later references to the parameter. For the above example this will be L(R,C,T,M) where R, C, T and M refer to the simple sets.

6.3 Assignment and Equation Statements

Most of the syntax used in assignment statements and equations are the same, although it is straightforward to detect if a GAMS statement is an assignment or an equation.

An assignment statement in GAMS is an instruction to perform some data manipulation and store the result. It can be compared to a FORTRAN statement where the result of the operations performed is stored under the name that appears on the left side of the equal sign. As an example consider the parameter DIST(I,J) indicating the distance from location I to location J, where the elements in the sets I and J are identical. Assume that initially only the lower triangular part of DIST was specified in a TABLE statement, and that we are interested in specifying the entire matrix. We can write the following sentence

```
DIST(I,J) = DIST(I,J) + DIST(J,I) ;
```

The right-hand side is defined for each 2-tuple of the Cartesian product of the sets I and J. A copy of DIST(I,J) is stored in a temporary work array, and the entries in DIST(I,J) are replaced with the results from the additions for all pairs (I,J) in a parallel fashion. Note that all values of DIST(I,J) that were not defined in the TABLE statements are assumed to be zero. An alternative but equivalent GAMS statement for the above replacement is as follows.

```
DIST(I,J) = MAX(DIST(I,J), DIST(J,I)) ;
```

Here the MAX operator selects the largest of the two values inside the parentheses.

An equation in GAMS is a symbolic representation of one or more constraints to be used as part of a simultaneous system of equations, or an optimization model. It always begins with the equation name, possibly indexed, followed by two dots (periods). We again refer to the equations in the cannery example.

6.4 The $ Operator

Partitioning large models by using driving indices provides an elegant short-hand notation. Complexities, however, are introduced when there are restrictions imposed on the partitionings. As these complexities arise continually in large-scale models, we have strived for an elegant and effective way to incorporate them in a model statement.

Let us begin with an example. Define the sets R and D as regions and districts respectively. Assume that for each district in a region we know the level of income YD(R,D), and that we want to determine the regional income YR(R) for each of the regions. Writing the assignment statement

```
YR(R) = SUM(D, YD(R,D)) ;
```

is meaningless as not every district is contained in each region. We need to use, therefore, the relationship between the sets R and D. Let RD be the set correspondence between these two sets. Then we can write the following assignment statement

```
YR(R) = SUM(D$RD(R,D), YD(R,D)) ;
```

Here the dollar sign is used as a conditional operator. For each specific region R it restricts the sum to be over those elements of D for which the correspondence RD(R,D) is defined.

Let A be a name or an expression in GAMS, and let B be a name or a true-false expression. Then the phrase A $ B is a conditional statement in GAMS where the name A is considered or the expression A is evaluated if and only if the name B is defined or the expression B is true.

When the dollar operator is used in an assignment statement, it can appear both on the right and on the left of the equal sign. When it appears on the left, it controls the domain over which the assignment is defined. Whenever the condition following the name on the left is not true,the existing data values contained under that name remain unaffected. If on the other hand, that same condition is applied to the right of the equal sign, the existing values contained in the name on the left will be set to zero whenever the condition is not true.

In order to illustrate the conjunctive use of the dollar operator and logical phrases contained in an assignment statement, consider the next example. Let the sets P, I and M denote processes, plants and machines respectively. The parameter K(M,I) denotes the number of units of available capacity of machine M in plant I, while the parameter B(M,P) describes the required number of units of capacity of machine M per unit level of process P. We want to define a zero-one parameter, PPOSS(P,I), indicating which processes P need to be considered for plant I. We can write the following set of logical relations always resulting in either a zero or one.

```
PPOSS(P,I) = SUM(M $ (K(M,I) EQ 0), B(M,P) NE 0) EQ 0 ;
```

Here the expression B(M,P) NE 0 will contain a value 1 if process P is dependent on machine M, and 0 otherwise. These values are summed over all machines M that are not available in plant I. If the resulting sum is zero for process P then the process is not dependent on unavailable machines, and should therefore be considered. Note that PPOSS is one in this case. If the resulting sum is not 0, the process is dependent on at least one unavailable machine, and should therefore not be considered. The parameter PPOSS is set to zero in this case.

When the dollar operator appears in an equation statement, it is used to control the generation of equations and/or variables. As an illustration let CAP be an equation name referring to capacity constraints, and let Z be a variable name referring to levels of process operation. Using the notation of the previous paragraph, we can write the following symbolic equation.

```
CAP(M,I) $ (K(M,I) GT 0)..
     SUM(P $ PPOSS(P,I), B(M,)) * Z(P,I)  =L=   K(M,I) ;
```

In this example the system will generate an equation for a specific pair of machines and plants only when the capacity of that machine in that plant is strictly positive. Similarly, only those variables that refer to processes which can be operated at a positive level will be generated.

6.5 The Lag and Lead Operators

Most sets employed in mathematical models are collections of labels whose only purpose is to identify objects, properties or events that are relevant to the model description. There are sets, however, for which the order of the elements is crucial. One frequently used example is any set expressing some notion of time. For these sets it is often important to reference elements relative to each other. A forward reference is usually referred to as a "lead" while a backward reference is referred to as a "lag." In GAMS it is possible to perform lag and lead operations on any set whenever the elements in that set have never been used in a previous set definition. The order of entry is then the relevant order. In the case that they have been used in previous sets, their mutual order should be unaltered. Whenever a set is generated or modified via an assignment statement, the system will not execute any lag or lead operations using this set.

In the language we make a distinction between two types of lead and lag operations. If the lead operator is + and points to an element beyond the last element in the set, the corresponding operation is not performed. If the lead operator is ++ , it will act as a circular operator, and consider the k^{th} element beyond the last element in the set to be the k^{th} element in the set. The lag operators - and — are defined in a similar manner. We will give an example of each.

```
   SET  M  MONTHS  /  JANUARY, FEBRUARY, MARCH, APRIL, MAY, ....
          DECEMBER / ;

   PARAMETER NSALE PROJECTED CUMULATIVE SALES OF NITROGENOUS FERTILIZERS;
*         (IN 1000's OF KILOGRAMS)

   NSALE('JANUARY')  =  100.  ;

   LOOP(T, NSALE(T+1)  =  1.05 * NSALE(T) ;
```

In this example a forward projection is made on the basis of the starting value of the first month. The term NSALE('DECEMBER' + 1) will be considered as vacuous. Note that the looping device is necessary for the above assignment statement. Without it, all operations will be performed in a parallel fashion, which will result in a proper definition of the parameter NSALE('FEBRUARY') only. All other values of NSALE will be equal to the implied default value of zero.

In some agricultural models, the constant set of months has been used in a circular fashion, where JANUARY is the one-period lead of DECEMBER and DECEMBER is the one-period lag of JANUARY. As an example assume that we want to determine the five-dimensional parameter CLAB denoting the labor requirement coefficient by district, crop, technology, month and planting date. Assume also that the planting dates are EARLY and LATE, and that the coefficient values for both are the same except that they differ by a month. Let the parameter LABREQ be the labor requirement coefficients by district, crop, technology and month, obtained via a TABLE statement. The CLAB can be generated from LABREQ as follows.

```
CLAB(D,C,T,M,'EARLY')  =  LABREQ(D,C,T,M)  ;
CLAB(D,C,T,M,'LATE')  =  CLAB(D,C,T--1,'EARLY')  ;
```

7. Summary and Conclusion

In this paper we have described the limitations of our current modeling technology when employed in a strategic planning environment. For modeling to become successful, we have proposed the following two basic changes in modeling technology. First we need a universally accepted, easy to use, general purpose modeling language which is readable by both man and machine. Secondly we need a modeling system that can readily interface with data bases, solution algorithms and report generators, and that can perform such tasks as the linking of models, unit analysis and automatic scaling.

Following this discussion, the paper describes the development of a General Algebraic Modeling System (GAMS), which we view as a first step toward a new technology in modeling. Although modelers in the

Research Center of the World Bank have greatly benefited from a system such as GAMS, a unified effort by the entire modeling community and software industry is needed to bring about a universal change in modeling capabilities.

Our prediction is that there will be a tremendous increase in model-building activities over the next decade or so if software manufacturers provide the technology for modeling exercises to become successful in a strategic planning environment. It is our sincere wish that they will all share the same modeling language. Without such a standard, models will not be portable, which limits their success. In addition, the burden of having to learn many different notations is imposed on the growing group of model builders. We hope that this paper will become a stepping stone for the development of a universally available modeling language, and that the general topic will gain the attention of the modeling community as a whole.

PART OF GRAMMAR OF GAMS

Equations and Assignments

```
<equation>            ::=   <left part> .. <expression> <type> <expression> ;
<assignment>          ::=   <left part> = <expression> ;
<left part>           ::=   <left> | <left> $ <primary>
<left>                ::=    ident | ident ( <index list> )
<index list>          ::=   <index expression> | <index list> , <index expression>
<index expression>    ::=   <simple index> | <simple index> <lag operator> <primary>
<simple index>        ::=    ident | 'index'
<primary>             ::=   <variable> | number | SUM ( <con control> , <expression> ) |
                             function ( <expression list> ) | ( <expression> )
<variable>            ::=    ident | ident ( <index list> )
<expression list>     ::=   <expression> | <expression list> , <expression>
<con control>         ::=   <control> | <control> $ <primary>
<control>             ::= _ ( <ident list> ) | ident
<expression>          ::=   <term> | <unary operator> <term> |
                              <expression> <binary operator> <term>
<term>                ::=   <primary> | <term> $ <primary>
<unary operator>      ::=   - | + | NOT
<binary operator>     ::=   - | + | * | / | ** |EQ|GT|GE|LT|LE|NE|AND|OR|XOR
<type>                ::=   =L= | =G= | =E=
<lag operator>        ::=   + | - | ++ | --
```

Set Definitions

```
<set definition>      ::=   SET <declaration list> ;
<declaration list>    ::=   <declaration> | <declaration list> , <declaration>
<declaration>         ::=   <set name> | <set name> / <value list> /
<set name>            ::=    ident | ident text
<value list>          ::=   <element text> | <value list> , <element text> | ALL
<element text>        ::=   <element> | <element> text
<element>             ::=   <simple element> | <element> . <simple element> |
                            <element> . ( <element list> ) | ( <element list> )
<element list>        ::=   <element> | <element list> , <element>
<simple element>      ::=    index | 'index'
```

Table Definitions

```
<table definition>    ::=   TABLE  <table name> <table body> ;
<table body>          ::=   eol <columns> <table values> |
                               <table body> eol + <columns> <tables values>
<columns>             ::=   <elements>| <columns> <elements>
<table values>        ::=   <row> | <table values> <row>
<row>                 ::=   eol <element> | eol <element> <row values>
<row values>          ::=   <signed number> | <row values> <signed number>
<table name>          ::=   <identifier> | <identifier> <text>
```

REFERENCES

[1] H. B. Chenery and R. Weaving, "An Overview of Research at the World Bank," World Bank Research News, Vol. 1, No. 1, January 1980, pp. 1-9.

[2] G. Fromm, W. L. Hamilton and D. E. Hamilton, Federally Supported Mathematical Models: Survey and Analysis, U.S. Government Printing Office, Washington, D.C., 1974.

[3] P. W. House and J. McLeod, Large Scale Models for Policy Evaluation, John Wiley and Sons, Inc., New York, 1977.

[4] R. E. Pugh, Evaluation of Policy Simulation Models, Information Resources Press, Washington, D.C., 1977.

AUTOMATIC IDENTIFICATION OF GENERALIZED UPPER BOUNDS IN LARGE SCALE OPTIMIZATION MODELS

Gerald G. Brown and David S. Thomen

Naval Postgraduate School
Monterey, California

To solve contemporary large scale linear, integer and mixed integer programming problems, it is often necessary to exploit intrinsic special structure in the model at hand. One commonly used technique is to identify and then to exploit in a basis factorization algorithm a generalized upper bound (GUB) structure. This report compares several existing methods for identifying a GUB structure. Computer programs have been written to permit comparison of computational efficiency. The GUB programs have been incorporated in an existing optimization system of advanced design and have been tested on a variety of large scale real life optimization problems. The identification of GUB sets of maximum size is shown to be among the class of NP-complete problems; these problems are widely conjectured to be intractable in that no polynomial-time algorithm has been demonstrated for solving them. All the methods discussed in this report are polynomial-time heuristic algorithms that attempt to find, but do not guarantee, GUB sets of maximum size. Bounds for the maximum size of GUB sets are developed, in order to evaluate the effectiveness of the heuristic algorithms.

1. INTRODUCTION

Contemporary mathematical programming models are often so large that direct solution of the associated linear programming (LP) problems with the classical simplex method is prohibitively expensive, if not impossible in a practical sense. It has been found that most of these problems are sparse, with relatively few non-zero coefficients, and usually possess very systematic structure. These problems exhibit inherent structural characteristics that can be exploited by specializations of the simplex procedure.

Methods to exploit special model structure can be categorized generally as *indirect* (e.g., decomposition), where a solution to the original problem is achieved by dealing with related models which are individually easier to solve, or as *direct*, when the original problem is solved by a modified simplex algorithm. Among the direct methods, the most frequently used technique is called *basis factorization* [7], where the reflection of special problem structure appears and is used to good benefit in the intermediate LP bases. Basis factorization can be *dynamic*, where the algorithm deals with each basis sequentially and/or independently in an attempt to extract as much specialized basis structure as possible, or *static*, where the algorithm depends upon certain types of special structure being present in *all* bases.

Static basis factorizations include *simple upper bounds*, *generalized upper bounds* (GUB), and *embedded network rows*, among many others. Simple upper bounds are a set of rows for which each row has only one non-zero coefficient. Generalized upper bounds are a set of rows for which each column (restricted to those rows) has at most one non-zero coefficient. Network rows are a set of rows for which each column (restricted to those rows) has at most two non-zero coefficients of opposite sign.

Each of these factorizations permits the simplex algorithm to deal with the static subsets of the rows (and columns) of all bases encountered with prior knowledge that they will satisfy *very* restricted rules. Most of these methods work best when *logic* can be substituted for arithmetic (as is the case with the coefficients ± 1). For this reason, static factorizations often restrict the special structure to possess only ± 1, or to be *scaled* so as to produce an equivalent result.

The concept of generalized upper bounds was introduced in 1964, the result of work by Dantzig and Van Slyke [5]. The name is derived from analogy to the simple upper bound structure. Graves and McBride [7] refer to *Static Signed Identity Factorization* as a term more suggestive of the implied basis structure. Since their introduction, some form of GUB has been implemented in many commercial LP systems. There is often confusion between the mathematical characterization of GUB and these various, widely used implementations of GUB, in that the latter often restrict the GUB set membership rules to permit uncomplicated simplex logic. All of the methods reported here address the full generality of GUB sets but can be modified to produce restricted GUB sets as necessary.

The details of how GUB can be exploited to reduce the computations of the simplex algorithm are not discussed here. See [1,5,7,11,13]. The underlying concept is that the GUB structure enables the simplex algorithm to manipulate the GUB rows implicitly, with logic rather than floating point arithmetic, thus reducing the effective size and solution time for the problem. The more GUB rows one is able to identify, the fewer rows one has to carry explicitly through the simplex operations. In large problems there exists a huge number of subsets of rows that satisfy the GUB criteria. It is generally regarded that those subsets with more rows are "better" GUB sets since they imply a more contracted explicit basis. The implied problem, then, is to find the *maximum* GUB set.

Optimal algorithms to find a maximum GUB set do exist. These entail enumeration schemes and cannot be guaranteed to be efficient in a practical sense. Conceivably, 2^m-m sets of rows might have to be searched before a maximum GUB structure is found: as the problem size grows, the number of possible sets that need to be checked increases *exponentially*. As will be shown later, the hope of finding an efficient algorithm to find the *maximum* GUB set for any general problem is dim.

Therefore, researchers and practitioners have concentrated on constructing efficient *heuristic* algorithms that attempt to identify, but do not guarantee, a maximum GUB set. A few of these methods showing great promise have been reported, but they have not been tested with large scale problems.

This report (abstracted from [4]) outlines several automatic heuristic GUB-finding procedures that have been developed and published in the recent literature. These procedures are tested on a suite of large scale, real life optimization problems, and are modified to improve their behavior. Comparative performance of the methods is given both in terms of the computational effort to identify a GUB set, as well as the size of the GUB set achieved.

Identification of GUB sets of maximum row dimension is shown in Section 7 to be among the class of NP-complete problems. However, easily computed *upper bounds* on the size of the maximum GUB set are developed and used to evaluate objectively the quality of heuristic GUB algorithms, showing that very nearly maximum GUB sets are routinely achieved.

2. PROBLEM DEFINITION AND REPRESENTATIONS

The Linear Programming problem is defined as

$$
\begin{array}{lll}
\text{(L)} & \text{Min } c^t x & \\
 & \text{s.t. } \underline{r} \leq Ax \leq \bar{r} & \text{(range constraints)} \\
 & \underline{b} \leq x \leq \bar{b} & \text{(simple bounds)}
\end{array}
$$

where $\underline{r}$ and $\bar{r}$ are m-vectors, x, c, $\underline{b}$ and $\bar{b}$ are n-vectors and A is an m × n matrix. The constraints are sometimes defined as equations, but for the general case of GUB treated here constraints can be equations, inequalities or a mixture. The immediate discussion will be directed at (L); integer and mixed integer problems are treated later.

Two rows of A are said to *conflict* if there exists at least one column with non-zero coefficients in both rows. The GUB problem can be restated as that of finding a subset of the rows that do not conflict.

There are several ways one can model the maximum GUB problem. Three approaches are presented to aid in the understanding of the theoretical context of the heuristic methods examined and to highlight the formal complexity of the original problem.

Graph Theory Representation

A graphical representation of the matrix A can be constructed through the following mapping rule, f. Let each row of A be a vertex of the graph. Should two rows of A conflict then the two vertices of the graph are joined by an edge. This mapping retains all the necessary conflict information. If two vertices, a and b, are joined by an edge, e, then a and b are adjacent, and a (or b) is *incident with* e. If a and b are not adjacent, this indicates that the corresponding two rows in A do not conflict.

This introduces the notion of *independence*. Given a graph $G = (V,E)$, a subset $V' \subset V$ is said to be an *independent set* if no two of its elements are adjacent. It follows that if an independent set of vertices can be found in G then the corresponding rows of the matrix A do not conflict and thus define a GUB set. Conversely, a GUB set for A defines an independent set for the graph G. It is also clear that an independent set for G is maximum if and only if the corresponding GUB set for A is maximum.

Consider the set A_m, the set of all A-type matrices having m rows. The above mapping factors this set into a definite number of *classes*. Two matrices, A_1 and A_2 are said to belong to the same class, C, if and only if each is mapped into the same graph, G_c.

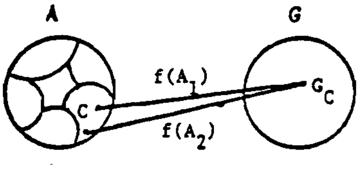


Figure 1

Thus, an independent set of vertices of G_c correspond to a GUB row set for every matrix in the class C.

The incidence matrix N is defined with $n_{ij} = 1$ if vertex i is incident with edge j, and $n_{ij} = 0$ otherwise. There exists one, and only one incidence matrix for each graph of G, where G is the set of all graphs having m vertices.

Since the set of all N-type matrices with m rows is a subset of A_m, every class of A_m contains one and only one incidence matrix. In general, for the GUB problem, every m row matrix is equivalent to one of a finite number of incidence matrices. Superficially this may seem to be a simplification. But as shown in Section 7 the GUB problem on N is as difficult as the independent set problem on G. The equivalent statements of the GUB problem do, however, offer different views of the problem which are helpful in considering algorithms for and analysis of the problem. (NOTE: In Garey and Johnson [6] it is shown that two other graph problems, the "vertex cover" and the "clique" problem, are equivalent to the independence problem, and hence the GUB problem. These problems do not seem to offer any additional insight for the GUB problem.)

Conflict Matrix Representation

The *conflict matrix* M is defined with $m_{ij} = 1$ if row i conflicts with row j in (L), and $m_{ij} = 0$ otherwise. Note that this matrix is symmetric. The sum for any row (or column) indicates the number of other rows it is in conflict with, plus one. This sum indicates for any particular row how many other rows would be subsequently excluded from a GUB set by its addition.

The rows of a GUB structure can be rearranged to form an embedded identity matrix in M.

Vector Space Representation

Yet another heuristic approach can be modelled using vectors in an n-dimensional vector space, where n is the number of variables in the problem (L). Consider each row of A as a vector in this space, having unit length in those "dimensions" corresponding with its non-zero coefficients.

R, the resultant vector from the sum of all vectors of the rows of A, indicates the number of conflicts, plus one, associated with each variable of (L). A hypercube in n-space situated in the first orthant at the origin with length 1 in all positive directions denotes the feasible GUB region. Should R extend beyond this area, then the set of rows corresponding to the vectors determining R does not constitute a GUB structure.

A gradient vector can be calculated indicating the direction of the shortest distance to the *feasible region*. It can be used to determine which row to remove from the set to obtain the largest movement in the desired direction. When R falls within the feasible region, the set of rows determining R constitutes a GUB set.

3. EARLIER LITERATURE

Two papers dealing with efficient GUB finding methods are worthy of special note.

Brearley, Mitra and Williams [2] establish a very useful framework for study of methods for finding GUB structure, as well as an insightful discussion of these methods and a taxonomy for their classification.

They define three sets consisting of the rows of the technological matrix A. The first set, the *eligible set*, is made up of every row of A that is individually eligible to belong in the GUB set. The *structure set* is a subset of the eligible set and includes all those rows currently considered as members of the GUB set. The *candidate set* consists of those rows of the eligible set that are candidates for inclusion (or re-inclusion) in the GUB set. Every one of the methods examined in [2] is described in terms of manipulation of these sets.

Each method of building a GUB set employs one of two basic strategies. The *row-addition* strategy begins with an empty structure set. Then, based on a particular criterion for inclusion, rows are removed from the candidate set and either added to the structure set or dropped from further consideration. This procedure continues until the candidate set is empty. The rows in the structure set form an admissible GUB structure.

The *row-deletion* strategy takes the opposite approach and is divided into two phases. Methods of this type initially place all eligible rows in the structure set. This normally leads to an infeasible GUB set with many conflicting rows. Based upon the particular decision rules, rows are removed from the structure set and placed in the candidate set. The first phase of this strategy ends when a feasible structure is obtained.

A second phase involves examining the removed rows in the candidate set. Those that do not conflict with any of the members of the current structure set are taken from the candidate set and reincluded in the structure set. Those that do conflict are deleted from the candidate set and dropped from further consideration. The second phase ends when the candidate set is empty. At this point the rows of the structure set constitute an admissible GUB set.

Brearley, Mitra, and Williams examine over 18 different methods. These approaches differ in the primary and secondary decision criteria for including (or removing) a row in the GUB structure set. The heuristic decision rules examined are based on the following model entities and combinations thereof: Include or remove a row based upon:

a) the number of non-zero elements in the given row,

b) the number of rows in conflict with the given row,

c) the number of non-zero elements in rows that conflict with the given row,

d) the row's relative weight obtained by the inner product of a vector representation of the row and a directional gradient.

These methods were implemented with an ALGOL program run on an ICL 4130 computer. Twelve linear programming problems ranging in size from 12 rows up to 166 rows were used for computational tests. The results show that those row-addition methods using heuristic (d) above "consistently performed very well" [2]. Similarly, those methods using heuristic (b) were found to perform nearly as well as (d).

McBride [15] compares the directional gradient method (d) with an approach suggested but not tested by Greenberg and Rarick [8]. The latter method uses the conflict matrix as does heuristic (b). However, it focuses on finding a maximal embedded identity matrix within the conflict matrix, rather than using the conflict matrix to determine conflict counts, applying a specialization of the preassigned pivot procedure (P^3) normally used for reinversion [9]. McBride's results indicate that heuristic (d) is significantly faster. However, neither method consistently achieves a larger GUB set.

McBride also comments on the notion of a "good" GUB set. He finds merit in selecting a set of GUB rows that minimizes the non-zero build-up in the representation of the inverse transformation of the explicit basis during actual optimization. Results are also given for a restricted GUB set selection that gives priority to equality constraints. Since equality constraints are always binding in feasible solutions, the subset of the basis associated with binding constraints, or kernel [7], is expected to have fewer explicit non-zero elements.

Based upon the results in these papers, and on independent computational experience with automatic GUB factorization reported by Brown and Graves [3], the present research initially concentrated on those approaches utilizing the two most successful heuristics based on conflict and directional gradient (i.e. methods I.2, II.2, II.9 and II.10 of [2]).

The models studied in this report are of much larger scale and include mixed integer problems as well as models for which prior GUB row sets have been manually specified.

4. DETERMINATION OF THE ELIGIBLE SET

The implementation of GUB in simplex algorithms usually admits only ± 1 as non-zero coefficients in the GUB rows. In linear programming, a column scaling can make each non-zero element in a GUB row ± 1. For variables of an integer or mixed integer programming problem, the columns of matrix A that correspond to integer variables cannot be scaled without inconvenience for other optimization functions depending upon the integrality condition. Therefore, non-zero elements in columns corresponding to integer variables will be modified by row scaling. If it is impossible to obtain the necessary ± 1 non-zero coefficients by row scaling and column scaling of columns corresponding to continuous-valued variables, the row is deemed not eligible for inclusion in a GUB set.

It is an objective of this research that the procedures examined for locating a GUB set in a linear programming problem be designed to be incorporated as an automatic, integral part of a contemporary optimization system of advanced design.

Each method is implemented as a feature of the read routine (written to accept input in the standard MPS format, as well as editing information indicating integer variables, scaling, and known prior GUB structure). Each method automatically examines the rows of the input and specifies a GUB set. The appropriate rows and columns are then scaled as necessary to obtain the proper GUB structure, and passed on to the optimizing portion of the system. (Note that the editing information places conditions that must be satisfied for any achievable GUB set.)

In determining the set of eligible rows, the following factors have to be considered.

a. Through the editing process, have some of the rows been dropped from the problem? If so, these "masked" rows are not eligible for inclusion in the GUB structure and are thus dropped from the set of eligible rows.

b. Through the editing process, have any rows been predesignated to be in the GUB structure? Large segments of the constraints can often be selected for the GUB set either visually or by recognition of a member of a convenient class of models. Any rows that conflict with these rows are not eligible for subsequent inclusion.

c. All rows designated "nonconstrained" (which include the objective function) are ineligible for inclusion in the GUB structure.

d. If there are any integer-valued variables, an additional check is performed. A row in the GUB set must eventually be capable of being scaled to ± 1 non-zero coefficients. This is achieved, if necessary, through a combination of row and column scaling. However, with integer variables, column scaling is no longer advisable. Therefore any row with a non-zero element in integer columns that is not a +1 or -1, or capable of being rendered into a ± 1 in those positions through row scaling alone, must be marked as ineligible for inclusion in the GUB structure.

Once the above restrictions have been considered, the resulting set of eligible rows is then available for search in order to construct the desired GUB structure.

5. IMPLEMENTATION OF AUTOMATIC GUB HEURISTICS

Conflict Methods

These employ the notion of a conflict measure for each row. Consider the conflict matrix, M, of the corresponding technological matrix A, for which a GUB set is to be found. An individual element, m_{ik} is 1 if row i and row k of the original matrix have at least one column j such that $a_{ij} \neq 0$ and $a_{kj} \neq 0$. If the two rows have no non-zero coefficients in a common column then the corresponding m_{ik} of the conflict matrix is 0. Summing across a row of the conflict matrix can thus give the measure of the number of rows plus one that are in conflict with a given row. For a given row, this sum less one, called the row's *deletion potential*, indicates exactly how many other rows would be immediately excluded from the GUB set by inclusion of this row.

Conflict row-addition places all the eligible rows on a candidate list. From the candidate list, individual rows are selected by *minimum* deletion potential and removed to be added to the structure set. Other rows that are in conflict with the selected row are immediately removed from the candidate list and discarded. The selection of rows for the structure set and the discarding of conflicting rows continues until the candidate list is exhausted. The resulting structure set forms a GUB set.

A modification to the above heuristic is possible which *breaks ties* among rows sharing the minimum deletion potential by (for instance) selecting the row having the most non-zero elements for inclusion with the GUB structure set.

The program used to test this heuristic approach is adapted from an earlier version made available by Graves.

Conflict Row-Addition

Step 1. Identify Eligible Rows. Set $\beta_i = 1$ if row i is an eligible row, and equal to 0 otherwise.

Step 2. Determine Deletion Potential. Scan each eligible row i and increment β_i by the number of other eligible rows k where a_{ij} and a_{kj} are both non-zero for at least one column j. (β_i is the deletion potential, plus one.)

Step 3. Stopping Condition. If any β_i is greater than 0, go to the next step. Otherwise, stop. At termination, the structure set is a GUB row set.

Step 4. Row Selection. Select row i having the minimum positive ("deletion potential") β_i and add it to the structure set.

Step 5. Exclude Rows in Conflict with Selected Row. Locate the (β_i-1) rows in conflict with the selected row. For each of these rows k, locate the (β_k-1) rows that they are in conflict with and decrement β_i for those rows by one.

Step 6. Marking Selected and Excluded Rows Ineligible for Further Consideration. Set β_i and the β_k's equal to zero. Go to step 3.

Conflict Row-deletion is a two-phase method which initially places all the eligible rows in the structure set. From this set individual rows are selected during Phase 1 and placed on the candidate list by *maximum* deletion potential. During Phase 2, remaining candidate rows that do not conflict with the structure set can be reconsidered in LOFI order [2]. A modification of phase 2 is used in this research which simply excludes from further consideration all conflicting rows, reincludes any remaining candidate rows, and repeats phase 1, until no further nonconflicting candidates remain.

Gradient Methods

Gradient row-deletion employs a heuristic method suggested by Senju and Toyoda [17] for approximate solution of certain linear programming problems with

0,1 variables. The objective is to obtain a maximum number of rows in the GUB structure while satisfying the stipulation that the GUB rows be disjoint.

$$\text{(S)}\quad \text{Max } Z = x_1 + x_2 + \cdots + x_m$$

$$\text{s.t.}\quad \sum_{i:a_{ij}\neq 0} x_i \leq 1,\ j = 1,\ldots, n$$

where $x_i \in \{0,1\}$

m is the number of candidate rows in (L),

n is the number of variables in (L),

x_i is the variable which determines whether row i is in the GUB set or not, and

Z is the objective function.

Using the vector space viewpoint outlined earlier, consider each row of (S) as a vector in n-space. A resultant vector R is determined by the sum of all the included rows and, in general, extends beyond the feasible space denoted by the unit hypercube. A gradient vector is calculated from this infeasible point in the direction of the shortest distance to the feasible region. An inner product of this gradient with each of the row vectors results in a relative weight for each row, which can be viewed as indicating the relative contribution that removal of the row would have towards obtaining a feasible structure set.

Rows are removed from the structure set according to their relative weight, the largest weight being removed first. This process is continued until a feasible set of GUB rows has been obtained. (The gradient vector is not recomputed as the method proceeds.)

Next, a phase 2 procedure examines each of the initially removed rows to see if any can be reincluded into the structure set without violating the GUB restrictions. Upon completion of phase 2, the selected rows constitute a GUB set.

A variation on the above procedure recalculates the shortest distance to the feasible region after the removal of each row. With the new gradient, a new set of relative weights for the remaining rows is then calculated and used, if necessary, to determine which of the subsequent rows will be removed.

Another modification is possible whenever two rows are found with equal weights. As a tie-breaking rule, the row found to have the least number of non-zero coefficients may be discarded first.

Gradient Row-Deletion

Phase 1: Deletion of Infeasible Rows

Step 0. Initialize Sets. Add all eligible rows to the structure set. The candidate set is empty.

Step 1. Determine the Vector R. For each column j, define ρ_j as the number of rows in the structure set having non-zero elements in column j.

Step 2. Determine Relative Weight of Each Row. For each row i, define ν_i as the sum of (ρ_j-1) of every column j, for which $a_{ij} \neq 0$.

Step 3. Feasibility Condition. If for every column, $\rho_j \leq 1$, then go to step 6; else find a column j such that $\rho_j > 1$.

Step 4. Determine Row for Exclusion. Examine the rows in the structure having non-zero elements in column j. Select the row i with the largest ν_i.

Step 5. Remove Selected Row. Remove row i from the structure set, decrementing ρ_j by one for every column j with $a_{ij} \neq 0$. Add row i to the candidate set and return to step 3.

Phase 2: Improve Feasible GUB Set Found by Re-including Excluded Rows

Step 6. Eliminate Rows in Candidate Set that Conflict with the Feasible Set. For every row i of the candidate set that has at least one $a_{ij} \neq 0$ in a column with $\rho_j = 1$, remove that row from the candidate set.

Step 7. Re-inclusion of Rows. If any rows remain in the candidate set, then find row i having the smallest ν_i. Remove row i from the candidate set and re-include it in the structure set. Increment ρ_j by one for every column j where $a_{ij} \neq 0$.

Step 8. Stopping Condition. If the candidate set is empty, stop; else go to step 6.

To modify the algorithm in order to compute a new gradient vector after the removal of each row in phase 1, step 5 is changed as follows:

Step 5*. Remove Selected Row. Remove row i from the structure, decrementing ρ_j by one for every column j such that $a_{ij} \neq 0$. Locate each row k that is in conflict with row i. Decrement ν_k by the number of conflicts between the two rows. Add row i to the candidate set and return to step 3.

These two basic methods have been implemented as integral modules of a large scale optimization system. Therefore, explicit conflict matrices are not built. (To have done so would have consumed too much computer time and space.) Instead, all the information is stored in the vectors β, ρ, and ν. Logical flags associated with each row indicate whether it is eligible, and whether it is in the candidate set or in the structure set.

The problem data is expressed internally in terms of only the unique non-zero elements. This input is stored in a doubly linked list having both a row and a column thread. Thus, along with any non-zero coefficient a_{ij}, the location of adjacent non-zero elements in both the row i and column j are also immediately available. This crucial feature permits efficient row access for various operations (e.g., to locate all rows that conflict with a given row at a particular column).

6. COMPUTATIONAL RESULTS

The heuristic methods have been tested on 15 problems that vary in size from 92 constraints to 4,648 constraints. A description of each of the problems is given in Figure 2. As can be seen, four of the problems are mixed integer and two are pure integer.

Problem	Number of rows	Number of columns	Integer Columns	Non-Zeros
VANN	92	1,324	1,324	2,648
NETTING	103	247	103	494
AIRLP	171	3,040	0	6,023
COAL	171	3,753	0	7,506
TRUCK	239	4,752	4,752	30,074
CUPS	415	619	145	1,341
FERT	606	9,024	0	40,484
PIES	663	2,923	0	13,288
PAD	695	2,934	0	13,459
ELEC	785	2,800	0	8,462
GAS	799	5,536	0	27,474
FOAM	1,017	4,020	42	17,187
LANG	1,236	1,425	0	22,028
JCAP	2,487	3,849	560	9,510
ODSAS	4,648	4,683	0	30,520

Figure 2

The results of these experiments are given in Appendix A. The first two columns give the rows and non-zero column elements, respectively, of the GUB structures found. The time given in column three is the time required to locate the GUB set once the set of eligible rows has been determined. The final columns give additional information relating to the two versions of the gradient methods examined and represents total time in phase 1 and the number of rows re-included in the GUB structure during phase 2.

As with the earlier work cited, the Senju and Toyoda methods were found to be consistently the faster. Gradient row-deletion which updates the gradient after each row is removed takes longer in phase 1 than its non-updating counterpart. However, it so selectively deletes the rows, that few if any rows are ever added back into the structure during phase 2. This suggests that it be implemented as strictly a one phase method.

All methods are robust in that they consistently find large GUB sets. The conflict approaches generally find a larger number of variables with non-zero coefficients in the GUB rows. However, they definitely become relatively inefficient when larger problems are analyzed, regardless of the relative size of the GUB structure in the problem.

There is some discrepancy between these results and those published earlier [2]. The wide variation between gradient row-deletion with, and without, gradient updating has not been observed in the current experiments. It is hypothesized that this is due partially to differences in implementation of the various approaches and partially to problem size and structure variations between these studies.

7. PROBLEM COMPLEXITY

The *complexity* of a problem is said to be polynomial if an algorithm exists for which the fundamental operations are limited by a polynomial function of intrinsic problem dimensions. Such an algorithm would be called a *polynomial time* or *good* algorithm. The class of all problems for which such algorithms exist is denoted (P). If an algorithm is not polynomial time, then it is defined to be an *exponential time* algorithm. The disadvantage of an exponential algorithm is the explosive growth of the maximum solution time as the dimensions of the problem increase [14].

A problem x is said to be *reducible* to a problem y if each *good* algorithm for solving y can be used to produce in polynomial time a good algorithm for solving x [12]. Note that this does not necessarily require that a good algorithm for x and y actually exist. This requires only that if one exists for y, then one also exists for x.

An *intractable* problem is one for which it is known that no polynomial time algorithm exists. In between this class of problem, and the class P, is a vast number of problems whose status is uncertain. Among these is a class of *nondeterministic polynomial-time* problems (NP) for which a polynomial time algorithm can be shown to exist that can *verify* a guessed solution, but for which the existence of a (deterministic) polynomial time algorithm to actually solve a problem has not yet been demonstrated.

If every problem of the class NP is reducible to the problem y, then y is said to be *NP-hard*. In addition, if y itself belongs to NP, then y is NP-*complete* [6,12].

The following problem is known as the *independent set decision problem* (ISD). It belongs to the set of NP-complete problems.

(ISD) Given a graph $G = (V,E)$ and an integer t, decide whether G contains an independent set of size t or more.

The GUB decision problem (GUBD) can be defined as follows:

(GUBD) Given an integer p and an $m \times n$ matrix K defined as $K_{ij} = 1$ if $a_{ij} \neq 0$, and $K_{ij} = 0$ otherwise, decide whether K contains a set of p or more rows $i_1, i_2, \ldots, i_q$ such that

$$(*) \qquad \sum_{e=1}^{q} k_{i_e j} \leq 1 \qquad \text{for every column; } q \geq p \text{ .}$$

Given an instance of the ISD problem, the incidence matrix N can be constructed. This matrix along with the integer t is an instance of the GUBD problem. The following theorem proves the correctness of this reduction:

<u>Theorem</u>: The incidence matrix N has t rows satisfying (*) if and only if there are t vertices in G that are independent.

Proof.

a) Assume there exists t rows of N that satisfy (*). They correspond to vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_t}$ in G. If any two of these vertices are adjacent, then

$$\sum_{e=1}^{t} n_{i_e j} = 2$$

where j is the column in N that corresponds to the edge connecting the two vertices. This is a violation of the assumption, hence the t vertices in G are not connected to one another.

b) Assume there exists t vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_t}$ in G that are independent. Since no two are adjacent, the corresponding rows in N satisfy (*) [19]. Q.E.D.

Since the ISD problem, a problem known to be NP-complete, is reducible to the GUBD problem, it follows that the GUBD problem itself is NP-complete. (It is clear that the reduction is polynomial time and it is also clear that GUBD is in NP.) The related problems of finding a maximum independent set and a maximum GUB set are not in NP, however, they are NP-hard. It is therefore unlikely that a polynomial-time algorithm will be found for these problems. Only exponential-time algorithms are presently available.

The above analysis of GUB algorithms has only addressed the *worst case* bound. No conclusions are made about the average performance of an algorithm. In other words, the possibility of the existence of an algorithm with good average performance, but having an exponential worst case bound, has not been ruled out.

8. UPPER BOUNDS FOR THE SIZE OF MAXIMUM GUB SET

The intrinsic difficulty of identifying a maximum GUB set has been shown to be essentially impossible for problems of the scale at hand. However, the efficient heuristic procedures have been shown to provide very large GUB sets, whose size appears to be relatively stable for each problem regardless of the particular method applied. This suggests that these large GUB sets may be, in fact, very nearly maximum, although there is no practical way to verify this directly.

Although the problem of determining the size of the maximum GUB set is also NP-hard, it is possible to develop an easily computable *upper bound* on the maximum GUB set size. This bound can then be used to objectively evaluate the quality of the GUB sets produced by heuristic algorithms.

It is clear that the number of rows of a GUB set can be no greater than the number of rows in the problem. Also any one row by itself can form a GUB set. But these bounds are of little practical use where considering the problem of identifying a maximum GUB set. Utilizing information that is already available in the heuristic procedure, it is possible to construct in polynomial time an upper bound on the size of the maximum GUB set. (It is also possible to construct a lower bound on the size of the maximum GUB set, but that topic is not pursued in this report.)

For the purpose of developing a better bound, the incidence matrix representiation (N) of the problem is used. Let s_i be the number of 1's in row i. Note that s_i is the number of edges incident to vertex i in G. Also note that $s_i = \beta_i - 1$. The number of columns in N represents the number of distinct conflicts that exist between the rows of the original problem. This number is denoted as c, and can be found by the following formula

$$c = \frac{\sum_{i=1}^{n} s_i}{2} .$$

If c is greater than 0, all the rows of N cannot simultaneously belong to a GUB set, which implies the cardinality of the GUB set is less than m. As c becomes larger, the following argument shows that the upper bound of the maximum GUB set decreases.

If c is positive, but strictly less than m, it is possible for all the conflicts to involve one row. Removal of that row would then leave $m-1$ rows that form a GUB set. Thus for c in the range from 1 to $m-1$, an upper bound on the size of the maximum GUB set is $m-1$. Since one row can conflict with at most $m-1$ other rows, once $c \geq m$, at least two rows have to be removed to form a GUB set. For $m \leq c \leq [(m-1) + (m-2)]$ it is possible to construct a incidence matrix such that all the conflicts are between a pair of rows and the remaining set of rows. Removal of the pair would result in a GUB set of $m-2$ rows. This constructive argument continues until $c = [(m)(m-1)]/2$, which occurs when each row conflicts with every other row. At that point, the max maximum GUB = min maximum GUB = one row.

In general, for any problem with an $m \times c$ incidence matrix, the largest maximum GUB set that can be obtained is:

$$u_1 = \lfloor .5 + \sqrt{.25 + (m)(m-1) - 2c}$$

where $\lfloor$ indicates truncation to an integer.

The above bound is *problem-independent* and a *sharp* bound in that matrices with a GUB set the size of the bounding value can be constructed.

With additional information about a specific problem a better bound can be constructed. Since s_i is the number of other rows that conflict with row i, removing row i from the set of rows reduces the number of conflicts, c, by s_i. Let y denote $\max s_i$. Since y is the largest row conflict count, c can be reduced by not more than y with the removal of each row. The minimum number of rows that would have to be removed to reduce the number of row conflicts to 0, is $\lceil c/y$. Therefore, given m, c and y, the bound can be improved to

$$u_2 = \begin{cases} m - \lceil \dfrac{c}{y} , & c \leq (m-y)y \\ \lfloor .5 + \sqrt{.25 + y(2m-y-1) - 2c} , & c > (m-y)y \; ; \end{cases}$$

where $\lceil$ indicates the nearest higher integer.

In order to determine y, the entire β vector must be examined.

A third, even better bound can be obtained with additional information on the *frequency* of the conflict counts from 1 to y. The procedure is the same as above, in that when a row is removed with y conflict count, c decreases by y. However, instead of continuing to decrease c by y, it is decreased by the next largest s_i. This procedure continues until, once again, c becomes zero. This bound is named u_3.

The bounds developed can be used to objectively evaluate the size of a GUB set found by heuristic methods. In two problems examined, VANN and AIRLP, the number of rows in the GUB set equal an upper bound on the maximum GUB set for the problem. Therefore, for those problems, the heuristic methods are verified to have located maximum GUB sets.

Manual specification of a GUB set from visual inspection can utilize these bounds as an excellent measure of the maximum additional rows to be found. This information is also an aid in deciding whether to subject the problem to additional automatic search for GUB.

9. EXTENSIONS

The upper bounds developed in this report vary from a problem-independent bound to tighter problem-dependent bounds. It is speculated that additional information can be easily extracted from the actual conflict structure of the problems that can be used to tighten the existing bounds even further. This is strongly suggested by manual analysis of problems with particularly loose bounds for which the conflict structure seems to have higher order pathology. In addition, lower bounds have been developed by similar methods.

Another area that warrants further study is the special structure of the incidence matrix representation of the original problem. It is noted that for an incidence matrix, N, the relative weights generated for each row are (except for a constant) identical for both the conflict and the gradient methods studied. This implies that for a matrix N, the row-deletion heuristics will identify the same GUB set.

As things now stand, GUB-finding demands far less cost than the benefits derived during model optimization. Better GUB-finding methods may result from simple extensions arising from relaxations of (S), use of conflict information of higher order, limited application of backtracking enumeration, or exploitation of conditioned bounds on the remaining candidate rows to allocate heuristic effort.

Finally, research is continuing on automatic location of network row structure (e.g., Musalem [16] and Wright [18]). As one illustration of an immediate generalization of the GUB results, a GUB set for a problem can be identified and then another GUB set of an eligible subset of remaining rows can be found. Thus, a *bi-partite network row factorization* can be achieved (e.g., transportation or assignment rows).

10. CONCLUSIONS

The computational benefits of a large GUB set for an LP problem are widely recognized. This report shows that the identification of a maximum GUB set is a difficult problem, essentially as hard as many other widely known difficult problems.

The use of heuristics seems inescapable. This report has examined two promising heuristics (with two versions of each) applied to a series of real life, large scale models. All versions are robust in their ability to find large GUB row sets. However the two versions that use the Senju and Toyoda method are consistently the fastest. These two methods are essentially equal in their efficiency and effectiveness. Since the version which recalculates the gradient after the removal of each row so selectively removes the rows during the first phase that few if any rows are re-included in the GUB set during the second phase, This suggests that the latter phase be omitted.

The representation of an infinite number of m-row matrices by a finite number of incidence matrices offers a powerful and concise way of examining the GUB problem. Under this representation, both basic heuristic methods investigated assign (within a constant) the same relative selection weights to each row.

Finally, the ability to define upper bounds on the maximum size of the GUB set gives a new powerful tool in this area. It enables one to evaluate the quality of GUB sets found even in very large problems, for which the algorithmic identification of a maximum GUB set is probably impossible in general. In some cases, verification of a heuristically achieved maximum GUB set is now possible. Further, the bounds developed may be further enhanced in future research, and may be applicable to related problems of equivalent complexity.

ACKNOWLEDGMENTS

The authors wish to thank Gordon Bradley and Shmuel Zaks for their insights on complexity, and also Glenn Graves and William Wright for their considerable assistance.

APPENDIX A

This appendix contains computational results for fifteen linear, mixed integer and integer models. All execution times reported are expressed in actual CPU seconds, accurate to the precision displayed for IBM 360/67 and FORTRAN H (Extended).

For clarity, the following terms are defined:

Eligible rows: The number of rows of the model initially eligible for inclusion in a set of GUB rows.

Conflict count: The number of columns of the incidence matrix for the problem.

Conflict density: The ratio of the conflict count to the maximum conflict count for that problem size [i.e., $m(m-1)/2$].

Time to find Elig: The time in CPU seconds to determine the set of eligible rows.

IMAX: The maximum of s_i.

U_1, U_2, U_3: Bounds defined in Section 8.

The methods are labelled:

CRA	Conflict Row-Addition
CRD	Conflict Row-Deletion
GRD*	Gradient Row-Deletion (with gradient update)
GRD	Gradient Row-Deletion

Problem :	VANN	Description :	Fleet Dispatch Model		
Rows :	92	Eligible rows :	69	IMAX :	0
Columns :	1324	Conflict count :	0	U1 :	69
Integer :	1324	Conflict density :	0	U2 :	69
Non-zero :	2648	Time to find Elig :	.141 sec	U3 :	69

Method	Rows in	Columns in	Time to find	Time in	Number added
	GUB set	GUB set	GUB set (sec.)	Phase 1	in Phase 2
CRA	69	1324	.237		
CRD	69	1324	.125		
GRD*	69	1324	.202	.198	0
GRD	69	1324	.202	.198	0

Problem :	NETTING	Description :	Currency Exchange Model		
Rows :	103	Eligible rows :	71	IMAX :	5
Columns :	247	Conflict count :	46	U1 :	70
Integer :	103	Conflict density :	1.85%	U2 :	59
Non-zero :	494	Time to find Elig :	.022 sec	U3 :	46

Method	Rows in	Columns in	Time to find	Time in	Number added
	GUB set	GUB set	GUB set (sec.)	Phase 1	in Phase 2
CRA	36	84	.169		
CRD	36	84	.164		
GRD *	36	77	.047	.042	0
GRD	36	72	.042	.037	0

Problem :	AIRLP	Description :	Fleet Dispatch Model		
Rows :	171	Eligible rows :	170	IMAX :	150
Columns :	3040	Conflict count :	2983	U1 :	151
Integer :	0	Conflict density :	20.77%	U2 :	150
Non-zero :	6023	Time to find Elig :	.076 sec	U3 :	150

Method	Rows in	Columns in	Time to find	Time in	Number added
	GUB set	GUB set	GUB set (sec.)	Phase 1	in Phase 2
CRA	150	3000	1.16		
CRD	150	3000	.761		
GRD *	150	3000	.645	.639	0
GRD	150	3000	.444	.439	0

Problem :	COAL	Description :	Energy Development Model		
Rows :	171	Eligible rows :	170	IMAX :	111
Columns :	3753	Conflict count :	3753	U1 :	146
Integer :	0	Conflict density :	26.13%	U2 :	136
Non-zero :	7506	Time to find Elig :	.106 sec	U3 :	121

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	111	3753	1.38		
CRD	111	3753	1.24		
GRD*	111	3753	.920	.912	0
GRD	100	2568	.641	.631	0

Problem :	TRUCK	Description :	Fleet Dispatch Model		
Rows :	239	Eligible rows :	221	IMAX :	171
Columns :	4752	Conflict count :	10438	U1 :	165
Integer :	4752	Conflict density :	42.94%	U2 :	159
Non-zero :	30074	Time to find Elig :	.116 sec	U3 :	144

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	32	1069	6.88		
CRD	30	1099	7.095		
GRD*	30	857	5.00	4.95	2
GRD	32	986	1.70	1.58	8

Problem :	CUPS	Description :	Production Scheduling Model		
Rows :	415	Eligible rows :	390	IMAX :	48
Columns :	619	Conflict count :	744	U1 :	388
Integer :	145	Conflict density :	.98%	U2 :	374
Non-zero :	1341	Time to find Elig :	.042 sec	U3 :	294

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	213	494	2.96		
CRD	214	442	3.15		
GRD*	214	466	.212	.194	0
GRD	200	394	.384	.132	24

Problem :	FERT	Description :	Production & Distribution Model		
Rows :	606	Eligible rows :	605	IMAX :	580
Columns :	9024	Conflict count :	16455	U1 :	577
Integer :	0	Conflict density :	9.01%	U2 :	576
Non-zero :	40484	Time to find Elig :	.257 sec	U3 :	567

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	559	9024	15.8		
CRD	559	9024	10.5		
GRD *	559	9024	6.73	6.71	0
GRD	559	9024	2.52	2.50	0

Problem :	PIES	Description :	Energy Production & Consumption Model		
Rows :	663	Eligible rows :	662	IMAX :	21
Columns :	2923	Conflict count :	4116	U1 :	655
Integer :	0	Conflict density :	1.88%	U2 :	466
Non-zero :	13288	Time to find Elig :	.866 sec	U3 :	422

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	180	1848	10.8		
CRD	169	1693	13.5		
GRD *	172	1811	2.82	2.77	1
GRD	177	1761	1.31	.788	28

Problem :	PAD	Description :	Energy Production & Consumption Model		
Rows :	695	Eligible rows :	694	IMAX :	23
Columns :	2934	Conflict count :	4416	U1 :	687
Integer :	0	Conflict density :	1.84%	U2 :	502
Non-zero :	13459	Time to find Elig :	.104 sec	U3 :	449

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	200	1864	13.1		
CRD	189	1771	16.6		
GRD *	188	1708	3.34	3.26	2
GRD	189	1275	1.35	.928	21

Problem :	ELEC	Description :	Energy Production & Consumption Model		
Rows :	785	Eligible rows :	784	IMAX :	22
Columns :	2800	Conflict count :	6167	U1 :	776
Integer :	0	Conflict density :	2.01%	U2 :	503
Non-zero :	8462	Time to find Elig :	.089 sec	U3 :	492

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	309	2461	11.4		
CRD	210	2791	16.1		
GRD*	309	2641	1.15	1.12	0
GRD	309	2605	.842	.579	14

Problem :	GAS	Description :	Production Scheduling Model		
Rows :	799	Eligible rows :	789	IMAX :	608
Columns :	5536	Conflict count :	22220	U1 :	760
Integer :	0	Conflict density :	7.15%	U2 :	752
Non-zero :	27474	Time to find Elig :	.151 sec	U3 :	652

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	583	5102	16.2		
CRD	639	5536	10.4		
GRD*	608	5309	3.79	3.77	0
GRD	639	5533	1.47	1.44	1

Problem :	FOAM	Description :	Production Scheduling Model		
Rows :	1017	Eligible rows :	1006	IMAX :	261
Columns :	4020	Conflict count :	8186	U1 :	997
Integer :	42	Conflict density :	1.62%	U2 :	974
Non-zero :	17187	Time to find Elig :	198 sec	U3 :	934

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	932	4020	23.4		
CRD	932	4020	9.47		
GRD*	917	3981	1.73	1.71	0
GRD	917	3981	.902	.879	0

Problem :	LANG	Description :	Equipment & Manpower Scheduling Model		
Rows :	1236	Eligible rows :	1235	IMAX :	184
Columns :	1425	Conflict count :	46424	U1 :	1196
Integer :	0	Conflict density :	6.09%	U2 :	982
Non-zero :	22028	Time to find Elig :	.072 sec	U3 :	973

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	382	1207	46.2		
CRD	338	908	54.2		
GRD*	342	923	14.9	14.8	2
GRD	342	922	12.4	1.13	234

Problem :	JCAP	Description :	Production Scheduling Model		
Rows :	2487	Eligible rows :	2446	IMAX :	488
Columns :	3849	Conflict count :	16578	U1 :	2439
Integer :	560	Conflict density :	.55%	U2 :	2412
Non-zero :	9510	Time to find Elig :	.265 sec	U3 :	1812

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	529	2072	104		
CRD	512	2186	153		
GRD*	529	2087	2.23	1.87	5
GRD	523	1393	3.98	1.10	59

Problem :	ODSAS	Description :	Manpower Planning Model		
Rows :	4648	Eligible rows :	4647	IMAX :	4194
Columns :	4683	Conflict count :	5220	U1 :	4645
Integer :	0	Conflict density :	.05%	U2 :	4645
Non-zero :	30520	Time to find Elig :	.263 sec	U3 :	4024

Method	Rows in GUB set	Columns in GUB set	Time to find GUB set (sec.)	Time in Phase 1	Number added in Phase 2
CRA	751	3116	369		
CRD	721	3846	651		
GRD*	749	4436	7.12	6.88	0
GRD	751	3020	3.01	2.57	2

REFERENCES

[1] Beale, E. M. L., "Advanced Algorithmic Features for General Mathematical Programming Systems," *Intzger and Nonlinear Programming*, ed. J. Abadie, North-Holland/American Elsevier, 119-137, 1970.

[2] Brearley, A. L., Mitra, G., and Williams, H. P., "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm," *Mathematical Programming* 8, 54-83, 1975.

[3] Brown, G. and Graves, G., "Design and Implementation of a Large Scale (Mixed Integer, Nonlinear) Optimization System," paper presented at ORSA/TIMS Las Vegas, Nov. 1975.

[4] Brown, G. and Thomen, D., "Automatic Factorization of Generalized Upper Bounds in Large Scale Optimization Problems," NPS55-80-003, Naval Postgraduate School (January 1980).

[5] Dantzig, G. B. and Van Slyke, R. M., "Generalized Upper Bounding Techniques," *Journal of Computer and System Sciences* 1, 213-226, 1967.

[6] Garey, M. R. and Johnson, D.S., *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA., 1979.

[7] Graves, G. W. and McBride, R. D., "The Factorization Approach to Large-Scale Linear Programming," *Mathematical Programming* 10, 91-110, 1976.

[8] Greenberg, H. J. and Rarick, D.C., "Determining GUB Sets via an Invert Agenda Algorithm," *Mathematical Programming* 7, 240-244, 1974.

[9] Hellerman, E. and Rarick, D., "Reinversion with the Preassigned Pivot Procedure," *Mathematical Programming* 1, 195-216, 1971.

[10] Hirshfeld, D. S., "Generalized Upper Bounding (GUB): Theory, Applications and Performance," paper presented at Share XXXV, Mathematical Programming Project, August 1970.

[11] Kaul, R. N., "An Extension of Generalized Upper Bounded Techniques for Linear Programming," (ORC65-27) Operations Research Center, University of California, Berkeley, 1965.

[12] Klee, V., "Combinatorial Optimization: What is the State of the ART?" paper presented at SIAM Applied Mathematics Conference, Monterey, CA, February 1978.

[13] Lasdon, L. S., *Optimization Theory for Large Systems*, The MacMillan Company, New York, N.Y. 1970.

[14] Lewis, H. R. and Papadimitriou, C. H., "The Efficiency of Algorithms," *Scientific American* 238, No. 1, 96-109, 1978.

[15] McBride, R. D., "Linear Programming with Linked Lists and Automatic Guberization," Working Paper No. 8175, University of Southern California, School of Business, July 1975.

[16] Musalem, J. S., "Converting Linear Models to Network Models," Ph.D. Dissertation, UCLA (December 1979).

[17] Senju, S. and Toyoda, Y., "An Approach to Linear Programming with 0-1 Variables," *Management Science* 15, B196-B207, 1968.

[18] Wright, W., "Embedded Network Identification in Large Scale Optimization," M.S. Thesis, Naval Postgraduate School (June 1980).

[19] Zaks, S. (Private Communication, September 1979).

AUTOMATIC IDENTIFICATION OF EMBEDDED STRUCTURE IN LARGE-SCALE OPTIMIZATION MODELS

Gerald G. Brown and William G. Wright

Naval Postgraduate School
Monterey, California

This paper discusses automatic detection and exploitation of embedded structure in Large-Scale Linear Programming (LP) models. We report experiments with real-life LP and mixed-integer (MIP) models in which various methods are developed and tested as integral modules of an optimization system of advanced design [6]. We seek to understand the modeling implications of these embedded structures as well as to exploit them during actual optimization. The latter goal places heavy emphasis on efficient, as well as effective, identification techniques for economic application to large models. Several (polynomially complex) heuristic algorithms are presented from our work. In addition, bounds are developed for the maximum row dimension of the various factorizations. These bounds are useful for objectively estimating the quality of heuristically derived structures.

I. INTRODUCTION

Automatic detection and exploitation of special structure in large-scale LP (or MIP) models has been the subject of a continuing research program conducted at the Naval Postgraduate School and UCLA over the past decade. This paper draws from various results in this effort, and refers (sparingly) to significant work by other researchers. The references contain complete descriptions of these results for the interested reader.

Our scope is intentionally limited to automated methods of sufficient efficiency to enable us to economically apply them to real-world optimization problems. Thus, we consider only those approaches showing greatest promise for immediate practical application. Although the interpretations of embedded model structure can lend profound insights in their own right, we are equally interested in detecting errors in data preparation and model generation--mathematically mundane issues of fundamental importance to the practitioner.

The sheer size of contemporary large-scale LP models presents significant computational difficulties, even for otherwise elementary factorizations. Implementation of effective structural analysis procedures is primarily a matter of exercising large-scale data structures efficiently. As we shall see, though, these practical considerations can give significant theoretical guidance in the specification of efficiently achievable classes of model transformations.

That detection of embedded special structure can be of practical importance in actual model solution is undisputed. It is widely known that explicit simplex operations can be materially improved in efficiency by incorporation of basis factorization methods (e.g. [6], [9], and references of [7]). The details of such modifications of the simplex procedure are not given here. However, the underlying themes of simplex factorization are the substitution of logic for floating point arithmetic, and separation of the apparent problem monolith into more manageable components.

This paper deals exclusively with row factorizations. The pervasive implied problem for row factorization is the identification of the *best* embedded structure from all those that may lie at hand in any particular model. Conventional wisdom differs as to the criterion for this discrimination among factorizations of the same class. However, it is generally accepted that the row dimensionality of the factorization serves as an excellent measure of effectiveness. In this sense, embedded special structures fall naturally into a taxonomy implied by the intrinsic complexity of the associated maximum row identification problems.

We proceed with a discussion of several types of embedded special structures detectable by efficient polynomially complex algorithms. These structures are considered in increasing order of maximum row identification complexity. We emphasize that *efficient* polynomial algorithms are operationally defined here as low-order polynomial in terms of intrinsic problem dimensions (e.g. number of rows, columns, and non-zero elements), and *not* in terms of the total volume of model information (e.g. total number of bits in all coefficients, ad nauseam).

2. SIMPLE REDUCTIONS

LP models often exhibit simply detected structural characteristics which permit a reduction in row dimensionality without loss of model information. Several such reductions are possible in evidently polynomial complexity. These include:

a) Void Rows

b) Void Columns

c) Singleton Rows (simple upper bounds)

d) Singleton Columns

e) Fixed Variables

f) Rows that Fix Variables

g) Null Variables

h) Non-extremal Variables

i) Redundant Rows.

Some of these reductions do not obviously decrease row dimension. However, the reductions may be applied repeatedly to the model, revealing at each iteration more rows which can be removed. Thus, the cyclic application of reductions continues until a minimal model results.

Experiments with some of these reductions have been reported by Brearley, Mitra and Williams [5]. More extensive work at large-scale has been done by Bradley, Brown and Graves [3] and by Krabek [10].

Detection of *all* redundant LP rows requires complete solution of equivalent LP problems, and is thus equivalent in complexity to LP. (We hesitate to say polynomial in the sense of Khachian's recent result.) Thus, we restrict redundant row detection to *orthogonal* redundancy, revealed by substitution of bounds for problem variables. An efficient detection algorithm results.

With real-life LP and MIP models, a remarkably large fraction of model rows can be removed by these simple techniques. For some cases, models have been nearly *solved* this way.

We note that integrality conditions can be superimposed on these simple reductions (e.g. tighten bounds on integer variables by truncation) to strengthen them. Nonlinear models also benefit from these reductions, and from others not addressed in this paper.

Table 1 contains the characteristics of several real-life linear and mixed integer models. Table 2 displays the results of simple reductions applied to these models [3]. Multiple *passes* are made for each model until no more reductions are possible. The times given are for execution on an IBM 360/67 using FORTRAN H (Extended) without code optimization.

TABLE 1

SAMPLE LP (MIP) MODELS

MODEL	ROWS	COLUMNS		NON-ZERO
		TOTAL	INTEGER	COEFFICIENTS
NETTING	90	177	114	375
AIRLP	171	3,040	0	6,023
COAL	171	3,753	0	7,506
TRUCK	220	4,752	4,752	30,074
CUPS	361	582	145	1,341
FERT	606	9,024	0	40,484
PIES	663	2,923	0	13,288
PAD	695	3,934	0	13,459
ELEC	785	2,800	0	8,462
GAS	799	5,536	0	27,474
PILOT	976	2,172	0	13,057
FOAM	1,000	4,020	42	13,083
LANG	1,236	1,425	0	22,028
JCAP	2,487	3,849	560	9,510
PAPER	3,529	6,543	0	32,644
ODSAS	4,648	4,683	0	30,520

TABLE 2

SIMPLE REDUCTIONS [3]

MODEL	FIXED COLUMNS	SINGLETON COLUMNS	SINGLETON ROWS	DOUBLETON EQUATIONS	REDUNDANT ROWS	PASSES	CPU SEC.
NETTING	8	1	29	7	17	4	0.81
AIRLP	20	0	0	0	0	2	1.78
COAL	0	0	0	0	0	2	2.12
TRUCK	2	0	0	0	1	2	5.57
CUPS	57	49	18	39	55	4	1.90
FERT	406	0	0	0	13	4	14.25
PIES	183	50	16	0	0	3	3.32
PAD	183	30	16	0	0	3	3.26
ELEC	494	56	120	3	14	4	8.64
GAS	501	60	31	0	30	4	10.08
PILOT	277	123	12	36	91	11	17.15
FOAM	2	0	36	0	0	2	3.30
LANG	105	220	68	9	55	20	61.45
JCAP	6	414	277	180	360	3	12.16
PAPER	145	190	90	359	45	5	20.61
ODSAS	0	40	0	3,609	40	3	31.00

3. GENERALIZED UPPER BOUNDS

Rows for which each column has at most one non-zero coefficient (restricted to those rows) collectively form a generalized upper bound (GUB) set. Usually, we additionally require that the coefficients in these rows be capable of being rendered to ±1 by simple row or column scaling.

The problem of identifying a GUB set of *maximum* row dimension is NP-hard [7], making optimal GUB factorization algorithms hopelessly inefficient for our purposes. Heuristics adapted from work by Graves and by Senju and Toyoda (see [13], and references of [5] and [7]) work very effectively and dependably at large-scale to find *maximal* GUB sets.

Unfortunately, the problem of determining just the *size* of the maximum GUB set is also NP-hard. However, Brown and Thomen [7] have developed bounds on the size of the maximum GUB set which are sharp and easily computed. These bounds have been used to show, in some cases, that maximum GUB sets have been achieved via heuristic methods. In any case, the bounds provide excellent objective measure of the quality of any GUB set, regardless of the means of its derivation. Frequently, manual GUB analysis will suffice for models with amenable structure.

The bounds are developed in terms of the number of distinct *conflicts* present in the model. Two rows are in conflict if they each have a non-zero element in a common column, making them mutually exclusive in a GUB set. If s_i is the number of rows in conflict with row i, then the total problem conflict count for a model with m rows is

$$c = \frac{1}{2} \sum_i s_i < \frac{1}{2} m(m-1) \quad .$$

A problem-independent bound on the size of the maximum GUB set is [7]

$$u_1 = \lfloor .5 + \sqrt{.25 + m(m-1) - 2c} \quad ,$$

where $\lfloor$ indicates truncation to an integer.

A tighter, problem-dependent bound is

$$u_2 = \begin{cases} m - \lceil \frac{c}{y} \, , & c \leq (m-y)y \\ \lfloor .5 + \sqrt{.25 + y(2m-y-1) - 2c} \, , & c > (m-y)y \, ; \end{cases}$$

where

$$y = \max_i s_i \quad .$$

Tighter upper bounds have been derived for the size of the maximum GUB set, as well as lower bounds.

Table 3 contains the results of automatic GUB factorization applied to the benchmark models [7]. Row eligibility is based on the capability to scale the row to contain only $0, \pm 1$ coefficients. *GUB quality* is the number of GUB rows found, expressed as a percentage of the best known upper bound on maximum GUB row dimension (actual GUB quality may be better than this conservative estimate). The results were obtained using FORTRAN H (Extended) with code optimization.

TABLE 3

GUB FACTORIZATION [7]

MODEL	ROWS-GUB ELIGIBLE	ROW CONFLICTS		GUB		
		COUNT	DENSITY	ROWS	QUALITY	SEC
NETTING	71	46	1.85%	36	78.26%	0.05
AIRLP	170	2,983	20.64%	150	100%	0.65
COAL	170	3,753	26.13%	111	91.74%	0.92
TRUCK	219	10,438	43.53%	29	20.28%	5.00
CUPS	336	744	1.32%	160	66.67%	0.21
FERT	605	16,455	9.01%	559	98.59%	6.73
PIES	662	4,116	1.88%	172	40.76%	2.82
PAD	694	4,416	1.84%	188	41.87%	3.34
ELEC	784	6,167	2.01%	309	62.80%	1.15
GAS	789	22,220	7.15%	608	93.25%	3.79
PILOT	975	12,110	2.55%	255	33.73%	2.75
FOAM	989	8,186	1.67%	917	98.18%	1.73
LANG	1,235	46,424	6.09%	342	35.15%	14.90
JCAP	2,446	16,578	0.55%	529	29.19%	2.23
PAPER	3,528	35,047	2.82%	1041	34.65%	5.77
ODSAS	4,647	5,220	0.05%	749	18.61%	7.12

4. IMPLICIT NETWORK ROWS

Implicit network rows are a set of rows for which each column has at most two non-zero coefficients (restricted to those rows) and for which columns with two non-zero coefficients (in those rows) can be converted by *simple* row and column scaling such that the non-zero coefficients have opposite sign. Such rows in LP are commonly called networks with gains.

Pure network rows (NET) can be converted by *simple* row and column scaling such that all non-zero coefficients (restricted to those rows) have value ±1, and such that columns with two non-zero coefficients (in those rows) have one +1 and one -1. Such rows in LP are called pure networks (e.g. [4]).

Simple row and column scaling is restricted such that application of each scale factor renders an entire row, or column, to the desired sign (and unit magnitude for pure NET).

The problem of identifying a NET factorization of *maximum* row dimension is NP-hard [14], making optimal NET identification algorithms practically useless. The problem of determining just the *size* of the maximum NET set is also NP-hard. Thus, heuristic identification methods are mandated.

An extension of GUB heuristics can be used to achieve NET factorizations. First, a GUB set is determined by methods mentioned in Section 3. Then, a second GUB set is found from an eligible subset of remaining rows. The second GUB set is conditioned such that its row members must possess non-zero coefficients of opposite sign in each column for which the prior GUB set has a non-zero coefficient.

This double-GUB (DGUB) factorization yields a *bipartite* NET factorization. Thus, DGUB heuristically seeks the maximum embedded transportation or assignment row factorization. Pure network equivalents derive from proper editing of eligible rows.

Generalizing on the theme of Senju and Toyoda [13], a more general method has been developed by Brown and Wright [8] for direct NET factorization of implicit network rows. Pure NET rows can be identified with the same procedure by simple screening of admissible candidate rows.

This heuristic is designed to perform a network factorization of a signed elementary matrix (0,±1 entries only). It is a deletion heuristic which is feasibility seeking. The measure of infeasibility at any point is a matrix penalty computed as the sum of individual row penalties. The algorithm is two-phased, one pass, and non-backtracking. The first phase yields a feasible set of rows, while the second phase attempts to improve the set by reincluding rows previously excluded. Each iteration in Phase I either deletes a row or reflects it (multiplies it by -1) and guarantees that the matrix penalty will be reduced. Thus, the number of iterations in Phase I is bounded by the initial value of the matrix penalty, which is polynomially bounded.

Let $A = [a_{ij}]$ be an $m \times n$ matrix with $a_{ij} = 0, \pm 1 \; \forall \; i,j$.

Problem: Find a matrix $N = [n_{ij}]$ with $(m-k)$ rows and n columns which is derived from A by

1. Deleting k rows of A where $k \geq 0$,
2. Multiplying zero or more rows of A by -1,

where N has the property that each column of N has at most one +1 element and at most one -1 element. We wish to find a "large" N in the sense of containing as many rows as possible, i.e. minimize k.

Terminology and Notation:

1. E is the set of row indices for rows eligible for inclusion in N and is called the eligible set.
2. C is the set of row indices for rows removed from E in Phase I (Deletion). Some rows in C may be readmitted to E in Phase II. C is called the candidate set.
3. The phrase "reflect row i' of A" means to multiply each element in row i' by -1, i.e. $a_{i'j} \leftarrow -a_{i'j} \; \forall \, j$.
4. Other notation will be defined in the algorithm itself.

ALGORITHM:

Phase I - *Deletion of Infeasible Rows*

Step 0: *Initialization.* Set $E = \{1,2,\ldots,m\}$, $C = \phi$.

For each column j of A compute the + penalty (K_j^+) and the - penalty (K_j^-) as follows:

$$K_j^+ = \Big(\sum_{i \in E : a_{ij} > 0} 1 \Big) - 1 \quad , \qquad K_j^- = \Big(\sum_{i \in E : a_{ij} < 0} 1 \Big) - 1 \; .$$

These penalties represent the number of excess +1 and -1 elements, respectively, in column j which prevent the rows whose indices remain in E from forming a valid N matrix. A penalty value of -1 for $K_j^+ (K_j^-)$ indicates that the column does not contain a +1(-1) element.

Step 1: *Define Row Penalties.* For every $i \in E$, compute a row penalty (p_i) as follows:

$$p_i = \sum_{j:a_{ij}>0} K_j^+ + \sum_{j:a_{ij}<0} K_j^- .$$

This is simply the sum of + penalties for all columns in which row i has a +1 plus the sum of - penalties for all columns in which row i has a -1.

Step 2: *Define Matrix Penalty.* Compute the penalty (h) for the matrix by summing the row penalties as follows:

$$h = \sum_{i \in E} p_i$$

If $h = 0$, then go to Step 7. Otherwise, go to Step 3.

Step 3: *Row Selection.* Find the row $i' \in E$ with the grteatest penalty, i.e.

Find $i' \in E$ such that $p_{i'} = \max_{i \in E} p_i$.

(If there is a tie, choose i' from among the tied values.) Compute the reflected row penalty $\bar{p}_{i'}$ for i' as follows:

$$\bar{p}_{i'} = \sum_{j:a_{i'j}>0} (K_j^- + 1) + \sum_{j:a_{i'j}<0} (K_j^+ + 1) \,.$$

This would be the row penalty for row i' if it were to be reflected.

Step 4: *Delete, or Reflect Row.*

Case i) $\bar{p}_{i'} \geq p_{i'}$. Let $E \leftarrow E - \{i'\}$, $C \leftarrow C \cup \{i'\}$. Go to Step 5.

Case ii) $\bar{p}_{i'} < p_{i'}$. Reflect row i'. Go to Step 6.

Step 5: *Reduce column penalties* as follows:

For all j such that $a_{i'j} > 0$, $K_j^+ \leftarrow K_j^+ - 1$

For all j such that $a_{i'j} < 0$, $K_j^- \leftarrow K_j^- - 1$

Go to Step 1.

Step 6: *Change column penalties* as follows:

Using the $a_{i'j}$ values <u>after</u> reflection of row i',

For all j such that $a_{i'j} > 0$, $K_j^+ \leftarrow K_j^+ + 1$ and $K_j^- \leftarrow K_j^- - 1$

For all j such that $a_{i'j} < 0$, $K_j^+ \leftarrow K_j^+ - 1$ and $K_j^- \leftarrow K_j^- + 1$

Go to Step 1.

Phase II - *Reinclusion of Rows from C*

Step 7. *Eliminate Conflicting Rows.* The rows with indices in E, some possibly reflected from the original A matrix, form a valid N matrix. However, some of the rows removed from E and placed in C may now be reincluded in E if they do not make $h > 0$. Remove from C (and discard) all row indices for rows which, if reincluded in E in present or reflected form, would make $h > 0$.

i.e. Remove i from C if

a) $\exists j_1$ such that $a_{ij_1} > 0$ and $K^+_{j_1} = 0$

or $a_{ij_1} < 0$ and $K^-_{j_1} = 0$

and

b) $\exists j_2$ such that $a_{ij_2} > 0$ and $K^-_{j_2} = 0$

or $a_{ij_2} < 0$ and $K^+_{j_2} = 0$

If $C = \phi$, STOP, otherwise go to Step 8.

Step 8. *Select Row for Reinclusion.* At this point a row from C may be reincluded in E. There are several possible schemes for selecting the row. After the row is reincluded, the column penalties are adjusted. Then go to Step 7.

No dominating rule has been discovered for breaking ties in maximum row penalty encountered in Step 3. The rule used for the computational results presented herein is to select the row with the minimum number of non-zero entries in an attempt to place a larger number of non-zero entries in the network set. Other possible rules are "first-come, first-served," maximum number of non-zero entries, type of constraint, or modeler preference.

Modifications can be made to Step 0 to allow for 1) Matrices including non - 0,±1 entries and/or 2) Pre-specified network rows. The modifications are:

1. $E = \{i \mid a_{ij} = 0, \pm 1 \text{ for all } j\}$

2. Let $P = \{i \mid \text{row } i \text{ is prespecified}\}$

   $E \leftarrow E - P$

   After computation of K_j^+ and K_j^-, find for all j

   if $\exists i \in P$ such that $a_{ij} = 1$ then $K_j^+ \leftarrow K_j^+ + 1$,

   if $\exists i \in P$ such that $a_{ij} = -1$ then $K_j^- \leftarrow K_j^- + 1$.

At termination of the algorithm, the rows in N are given by $E \cup P$.

One easily obtained upper bound on the maximum row dimension of the network factorization is:

$$u_1 = m - \underset{j}{\text{MAX}}(K_j^+ + K_j^-) \quad .$$

This bound is easily computed and evidently sharp. It can be used to objectively evaluate the quality of a heuristically derived network factorization. The bound may also be used to preemptively terminate factorization effort.

Another, generally tighter, bound has been developed which is based on the reflection and deletion potentials for each row in the eligible set. Using this information it is possible to obtain a lower bound on the number of rows which must be deleted to achieve a feasible network set. The upper bound is then:

$$u_2 = m - \text{lower bound on rows deleted.}$$

This bound is also evidently sharp and is the bound used to compute NET quality in the following table.

Table 4 displays the results of DGUB and NET factorizations of the benchmark models. Row eligibility is determined by the capability to scale each row, by row scaling alone, to contain only $0, \pm 1$ entries. The *NET quality* is the number of NET rows found, expressed as a percentage of the upper bound on maximum NET row dimension given above (actual NET quality may be considerably better than this estimate).

TABLE 4

NET FACTORIZATION [8]

MODEL	ROWS NET ELIGIBLE	DGUB ROWS	DGUB SEC	NET ROWS	NET QUALITY	NET SEC
NETTING	59	54	0.07	54	94.74%	0.08
AIRLP	150	150	0.41	150	100%	0.35
COAL	111	111	0.50	111	100%	0.43
TRUCK	219	47	8.40	46	33.58%	19.83
CUPS	300	251	0.29	295	99.33%	0.14
FERT	585	572	6.03	572	100%	6.15
PIES	142	128	0.56	128	96.97%	0.59
PAD	174	160	0.58	160	97.56%	0.59
ELEC	322	272	0.99	286	93.46%	2.07
GAS	752	682	5.00	668	94.08%	9.71
PILOT	109	109	0.92	109	100%	0.36
FOAM	966	951	1.89	951	99.58%	1.16
LANG	850	585	3.74	661	87.20%	14.82
JCAP	1,811	874	2.50	917	83.97%	44.07
PAPER	2,324	1,484	7.24	1,627	78.52%	94.16
ODSAS	410	317	3.39	286	77.51%	14.55

5. HIDDEN NETWORK ROWS

Hidden network rows[†] are a set of rows which satisfy NET row restrictions after linear transformation of the model. That is, realization of these (LNET) rows may require a general linear transformation of the original model.

The discrimination between *implicit* and *hidden* network rows is not (necessarily) in their use, but rather in their detection. The transformation group associated with implicit network rows involves *only* permutations and simple scaling of individual rows and columns. The hidden network rows require a completely general linear transformation and partial ordering. Thus, identification of hidden networks requires significant computation just to identify eligible rows, since any given row may conflict with subsets of its cohorts after transformation.

This problem has been solved for *complete* hidden network factorization, where all rows are shown to be LNET or the algorithm fails. Bixby and Cunningham [2] and Muslem [12] have given polynomially complex methods for complete LNET conversion. (The complete GUB problem is polynomial as well.)

Strategically, the complete hidden LNET factorization requires two steps:

DETECTION: necessary conditions for existence of a complete LNET factorization must be established, and

SCALING: a linear transformation to achieve the NET structure must be determined, if one exists.

[†] We have coopted the term *hidden* from Bixby [1], but his definition may not superficially appear to be equivalent.

Cunningham and Bixby attempt detection, followed by scaling. Musalem tries scaling, then detection. This is a crucial difference between methods, since problems which can not be completely NET factorized may fail in either step.

Briefly, Cunningham and Bixby *detect* by showing that the incidence matrix of the model rows can be converted to a graphic matroid. They employ a method by Tutte (see references of [2]). Given success, the graphic record of the detection is used to attempt to *scale* the model to NET, or to show that no such scaling exists.

Musalem *scales* the model to a ±1 matrix, and then uses a method by Iri (see references of [12]) to build a tree, edge by edge, which reveals the partial ordering coincident with complete hidden LNET factorization.

Both methods are polynomially complex. However, complete LNET factorization is relatively expensive by either method in that quite a large amount of real arithmetic and logic is required. Underlying data structures have not been suggested for either method. Both methods fail if complete LNET factorization is impossible, and neither leaves the investigator with much information useful in salvaging a partial LNET factorization. We conjecture that risk of preemptive failure narrowly favors the Musalem approach, since he defers the relatively involved detection step.

Locating a hidden LNET factorization of *maximal* row dimension has been suggested by Bixby [1] and by Musalem [12], but no concrete method is given and no computational testing is reported. Evidently, the *maximum* LNET problem is NP-hard, and its maximal relaxation remains unsolved in the practical sense of this report.

6. CONCLUSION

The techniques reported here have been used with great success on a wide variety of large LP (MIP) models. The context of this research is certainly atypical in that the models which we work with are often sent to us for analysis and solution precisely because they have already failed elsewhere. In these cases, our motives are to quickly diagnose suspected trouble before optimization, prescribe remedies, and perform the actual optimization reliably and efficiently.

This has undoubtedly biased our view of structural detection methods. Practical considerations arising from turnaround deadlines and the specific advantages of our own optimization system [6]† have colored our judgment. Many provocative suggestions for further research have not been pursued, either due to lack of opportunity, to poor intuition, or to sheer economics. Whether or not by equivalent prejudice, Krabek [10] reports some similar methods for simple reductions applied to large MIP's.

A great deal of insight has been gained from these experiments. The cost of factorization is truly insignificant relative to the information and (primarily) solution efficiency gained thereby. Revelations have ranged from outright rejection of absurd formulations to subtle inferences on the inter-personal

†The *X-System* (XS) differs in many ways from classical large-scale mathematical programming systems; it simultaneously supports simple and generalized upper bounds, general basis factorization, MIP, nonlinear, and decomposition features. In addition, the fundamental LP algorithm has been enhanced to intrinsically incorporate *elastic* range restrictions. XS is particularly suited for solution in limited time of large models with complicating features.

conflicts of model proponents. Very few models fail to reveal some totally unsuspected structural curiosity. Indeed, it is often some minor aberration that proves most revealing. Sometimes, the combined effects of several minor features collectively contribute to a discovery of significant model attributes.

Our general operational guideline has been to avoid heavy computational investment in factorization. Rather, highly efficient methods are used *repeatedly* on variations of each model. Manual and *intuitive* analysis of these results usually reveal much more than could be reasonably expected from any totally automated method applied to problems of exponential complexity. Interactive analysis of large-scale models is uncompromisingly challenging in a technical sense and equally rewarding.

Accordingly, we have not yet implemented maximal hidden network heuristics, or block-angular clustering methods. In the former case, we find intrinsic NET factorization to unerringly reveal more general network forms. Also, reformulation to a NET factorization commonly requires more than a linear transformation; variables and constraints must frequently be *augmented* to achieve the desired arc and node interpretation.

In the case of block-angular and attendant structures, we require a great deal more information than row and column index subsets and aggregate relations to develop an effective and economically sensible mathematical decomposition scheme; further, even for unfamiliar models such structure is usually apparent in those cases that invite decomposition.

Large factorizations are routinely found as intrinsic features in real-life models. However, we feel that it is an abominable practice to proselytize in favor of some particular model structure at the expense of model realism or common sense.

For instance, network models have recently received unprecedented attention in the literature. The implication has often been that since networks are usually found in models, networks should be used as the exclusive model. This is, of course, patent nonsense, smacking of a solution in search of a problem. An analyst should view factorizations as specializations of models, rather than forcing models to fit certain popular factorizations [4].

7. ACKNOWLEDGMENTS

We are deeply indebted to our colleagues Gordon Bradley and Glenn Graves, who have contributed fundamentally to this research. David Thomen is largely responsible for the GUB identification material.

8. REFERENCES

[1] Bixby, R., "Hidden and Embedded Structure in Linear Programs," paper presented at the Symposium in Computer-Assisted Analysis and Model Simplification, Boulder, 24 March 1980.

[2] Bixby, R. and Cunningham, W., "Converting Linear Programs to Network Problems," (to appear in Mathematics of Operations Research)(c. October, 1979).

[3] Bradley, G., Brown, G. and Graves, G., "Preprocessing Large Scale Optimization Models," paper presented at ORSA/TIMS, Atlanta (Nov. 1977).

[4] Bradley, G., Brown, G. and Graves, G., "Design and Implementation of Large Scale Primal Transshipment Algorithms," Management Science, Vol. 24, No. 1, p. 1 (1977).

[5] Brearly, A., Mitra, G. and Williams, H., "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm," Mathematical Programming, Vol. 8, p. 54 (1978).

[6] Brown, G. and Graves, G., "Design and Implementation of a Large-Scale (Mixed Integer) Optimization System," paper presented at ORSA/TIMS, Las Vegas, Nov. 1975.

[7] Brown, G. and Thomen, D., "Automatic Factorization of Generalized Upper Bounds in Large Scale Optimization Models," Technical Report NPS55-80-003, Naval Postgraduate School, Monterey (January 1980).

[8] Brown, G. and Wright, W., "Automatic Identification of Network Rows in Large-Scale Optimization Models," (in preparation).

[9] Graves, G. and McBride, R., "The Factorization Approach to Large-Scale Linear Programming," Math. Programming, Vol. 10, p. 91 (1976).

[10] Krabek, C., "Some Experiments in Mixed Integer Matrix Reduction," paper presented at XXIV International TIMS Meeting, Honolulu, June 18, 1979.

[11] McBride, R., "Factorization in Large-Scale Linear Programming," Ph.D. Dissertation, UCLA (1973).

[12] Musalem, S., "Converting Linear Models to Network Models," Ph.D. Dissertation, UCLA (Dec. 1979).

[13] Senju, S. and Toyoda, R., "An Approach to Linear Programming with 0-1 Variables," Management Science, Vol. 15, p. B196 (1968).

[14] Wright, W., "Automatic Identification of Network Rows in Large-Scale Optimization Models," M.S. Thesis, Naval Postgraduate School (May 1980).

HIERARCHICAL BLOCK-STRUCTURE AND FACTORIZATION METHODS

Vladimir A. Bulavskiy

Institute of Mathematics
Novosibirsk

In this paper some general concepts of hierarchical block-structure are presented. Previously considered structures are included as particular cases. The scheme of basis matrix factorization and a way of using this structure in nonlinear minimization are outlined.

1. INTRODUCTION

Two approaches are involved in developing computational methods to optimize large-scale linear systems. The first takes into account the sparsity of the data matrix; methods of this kind preserve sparsity through a preliminary rearrangng of the rows and columns at a suitable phase in the algorithm. The second approach exploits the special structure of the data matrix-- for example, hierarchical block structure; methods of this kind use the regular configuration of zeros in the data matrix for a special presentation of the inverse matrix. This paper describes some general concepts of hierarchical block structure and outlines a method for taking the structure into account.

The standard way of defining a nested structure is to choose an elementary block structure and then to allow several blocks (except the linking one) to have this structure recursively. The elementary structure used in this paper is more general than usual. It is based on Bulavskiy and Zryagina, 1977, 1978.

2. DEFINITION OF HIERARCHICAL BLOCK STRUCTURE

To introduce the general concept, we shall consider a few typical situations that are traditionally discussed. In Fig.1 three simple block-structured matrices are presented, with shaded areas indicating the allocation of nonzero values. All these situations can be described by a tree-like graph whose roots represent the entire matrix; other vertices correspond to the linked blocks and must be connected to the root by directed edges, which symbolize the block submission. However, each of the structures in Fig.1 have to be treated differently. To decompose the first two matrices, it is sufficient to remove the linking strip, which consists of either rows or columns. This operation does not alter the condition properties of the original matrix. In case (a) the rows of each diagonal block are linearly independent to at least the same degree as are those of the entire matrix. Thus we can, without loss of accuracy, divide each block independently into basic and nonbasic columns. The former group is the local basis of the corresponding block; the latter is included in the basis of the linking block.

We can treat case (b) in the same way, but the third matrix must be decomposed differently. Its linked blocks may be more ill-conditioned than is the whole matrix. To avoid loss of numerical accuracy, we divide the matrix in two steps. First, for example, we remove the horizontal linking strip as if case (a) had occurred with only one diagonal block. Dividing this block into local basic and nonbasic parts, we obtain the diagram on the left in Fig.2. The local basis is placed in the upper right-hand corner. As this local basis is a square nonsingular matrix of type (a), it can be divided in turn; the diagram on the right represents the resulting partition.

Thus we need consider only the two kinds of submission presented in cases (a) and (b) of Fig. 1. As both kinds may occur in one matrix, we must, to avoid confusion, identify and label the corresponding edges on the graph. It is convenient for us to label the edges of the first kind (on the left) with a minus sign and those of the second kind (on the right) with a plus sign. This conventionis illustrated in Fig. 3, where cases (a), (b) and (c) correspond to those in Fig. 1.

It seems reasonable to introduce a symmetrical structure that is a generalization of both principal structures. Such a structure and its graph are presented in Fig. 4. This structure is treated as elementary and each of the linked blocks is allowed to have this structure. Thus we come to the following general concept of *hierarchical block structure.*

Let $G(P,Q)$ be a graph with vertex set P and edge set Q. We assume that the graph is a tree with the root at the vertex 0 and that each of its edges is directed away from the root and denoted by the pendant vertex of the edge. Thus, $Q = P\setminus\{0\}$.

All edges are assumed to be labeled with a plus sign (for edges belonging to the set Q_+) or with a minus sign (for those from Q_-). The graph G is used as a skeleton of a structure. To define the structure, we must assign a block to each vertex. For this purpose we introduce the index sets

$$M_k, N_k, \bar{M}_k, \bar{N}_k, \quad k \in P \quad .$$

The meaning of these sets is clear from Fig. 4: M_k and N_k correspond to the entire block, while $\bar{M}_k$ and $\bar{N}_k$ describe its linking part. It is assumed that $\bar{M}_k \subset M_k, \bar{N}_k \subset N_k$ for $k \in P$ and $\bar{M}_k = M_k$, $\bar{N}_k = N_k$ for terminal blocks.

For our purposes, the following relations must hold. If vertices s and t are subordinated immediately to vertex k_r, then

1. the sets M_s and M_t, as well as N_s and N_t are disjoint
2. $M_s \subset M_k \setminus \bar{M}_k$, $N_s \subset \bar{N}_k$ if $s \in Q_-$
3. $M_s \subset M_k$, $N_s \subset N_k \setminus \bar{N}_k$ if $s \in Q_+$.

To complete the matrix determination, we must specify the blocks $A[\bar{M}_k, \bar{N}_k]$ for all k. The information introduced is not, of course, minimal. It is sufficient to have only the sets $\bar{M}_k$ and $\bar{N}_k$ for each $k \in P$, but the sets M_k, N_k, and the graph G demonstrate the hierarchical structure in explicit form.

4. BASIS FACTORIZATION

To describe the method of factorizing structured matrices, we consider some particular cases. If all the edges are labeled with a minus sign (that is, $Q_+ = \phi$), we have a purely *horizonal* structure. An example is presented in Fig. 5. We use two principal operations when decomposing a structured basis matrix:

1. select a maximal linearly independent set of columns for the matrix of full row rank
2. select a similar set of rows for a matrix of full column rank .

These operations are equivalent if we ignore the structure of the matrix, but in our case they are essentially different. Given the purely horizontal structure, we can implement the first procedure beginning with terminal blocks and advancing to the root. For example, in Fig. 5 we first select the local bases in four terminal blocks and for K = 3,4,5,6 obtain the following representation of these blocks:

$$A[\bar{M}_k,\bar{N}_k] = B_k \{ \underbrace{I}_{J_k} : \underbrace{R_k}_{S_k} \}$$

where the set J_k represents the basic columns in the block k, S_k represents nonbasic columns, and matrices B_k are the local bases. If we construct the matrices H_k as in Fig. 6 for K = 3,4,5,6 and multiply them by the entire matrix on its right-hand side, we exclude the nonbasic part of the terminal blocks. We can treat the transformed blocks 1 and 2 in the same way. As a result of these transformations, we obtain the decomposition in Fig. 7, where multipliers H_k must be ordered in accordance with block submission. Deeper hierarchies can clearly be treated in the same way. To eliminate the right-hand part of Fig. 7, we must multiply the right-hand side of this equality by the corresponding matrix $H_{0.}$

In the case under consideration the use of horizontal structure to a maximal degree does not affect the stability of the computations. This is not the case if the columns are linearly independent and we must select a row basis; this situation is presented in the diagram on the left in Fig. 8. We can begin with the terminal blocks again, but for computational stability we must choose some barrier δ and take care that the absolute value of the leading elements of the transformations is greater than δ. Thus in several blocks some rows will be free, as illustrated in the diagram on the right in Fig. 8, where the free rows are placed at the top.

In fact, we make the transformations as we made them previously, but the leading elements are chosen only from the lower part of the diagram. If this part is square, we obtain a local basis for this matrix. To eliminate the upper nonbasic part, we must now multiply the matrix on the right in Fig.8 by the appropriate matrix H_0, and this multiplication must also be effected on the left-hand side. If the lower part of the right matrix in Fig. 8 is not to be square, we must either decrease δ or note that the matrix to be decomposed is *ill-conditioned* (if δ is already sufficiently small).

When all the edges are labeled with a plus sign, we have a purely *vertical* structure. This case may be considered in the same way; the two situations that we previously encountered replace each other. Note that multipliers H_k in this case are on the left-hand side of the matrix $A[M_0,N_0]$:

$$(\prod_{k \in Q_+} H_k) \cdot A \qquad \text{is (lower) block-triangular.}$$

In the more general case presented in Fig. 9, we assume that the matrix to be decomposed is square and nonsingular. This structure is composed of two pure structures, one of which is horizontal and the other vertical. We may successively make use of both previously presented algorithms to give

$$(\prod_{k \in Q_+} H_k) \cdot A \cdot (\prod_{k \in Q_-} H_k) = B_T$$

where B_T is a (lower) block-triangular matrix whose diagonal submatrices are the local bases of the blocks. Note that if matrix A is not square but has more columns or more rows, the matrix B_T is trapezoidal.

To use the decomposition obtained we must have the inverses B_k^{-1} for all local bases. It is not our aim to discuss this matter. The inverses may be in a convenient form. If the inverse B_T^{-1} is available, the inverse for this *moustache-like* structure can be presented as

$$A^{-1} = (\prod_{k \in Q_-} H_k) \cdot B_T^{-1} \cdot (\prod_{k \in Q_+} H_k)$$

where the product is computed in the same order as the edge labelling of the graph in Fig.9.

The general case of hierarchical block structure can be reduced to these moustache-like structures. For this purpose, consider the example in Fig. 10, where the graph of a structured matrix is presented. If we ignore the structure of blocks 1,2,3,4, then we have the moustache-like factor structure in Fig. 11, and we can write the decomposition in (lower) block-triangular form as

$$H_1 \cdot H_2 \cdot A \cdot H_3 \cdot H_4 = [\bar{B}_3, \bar{B}_4, B_0, \bar{B}_1, \bar{B}_2] \quad .$$

The diagonal blocks are represented on the right-hand side of the equality.

The diagonal blocks $\bar{B}_1, \bar{B}_2, \bar{B}_3, \bar{B}_4$ can be reduced in turn to lower triangular matrices. For example, by multiplying $\bar{B}_1$ by the matrices H_7, H_6 on the left-hand side and by the matrix H_5 on the right-hand side, we arrive at the lower block-triangular matrix

$$H_7 \cdot H_6 \cdot \bar{B}_1 \cdot H_5 = [B_5, B_1, \bar{B}_7, B_6] \quad .$$

For the matrix $\bar{B}_7$, we must take yet another step. Thus we obtain a decomposition in which the order of multipliers is defined by the submission of blocks in each moustache-like structure and by the partial order in which these structures are nested. We shall not go into details in this discussion.

4. BASIS UPDATE

With regard to updating the decomposition, an algorithm exists for stable recomputation of the decomposition as one column is replaced by another, but the rules are complicated and we shall not consider them here. Similarly, it does not seem rational to apply these rules unless the structure has a low depth. It generally seems more reasonable for the modifications of the basis to be accumulated in either product form or in the form

$$(A + ST)^{-1} = \{I - A^{-1}S\cdot[I + TA^{-1}S]^{-1}\cdot T\}\cdot A^{-1} \ .$$

Here the $(p\times n)$ - matrix T consists of the unit rows indicating the basic columns to be changed, the columns of the $(n\times p)$ -matrix S are the corresponding corrections, and p is the number of modifications. The new column of $A^{-1}S$ is calculated by the simplex method. The necessary modifications of the $(p\times p)$ -matrix $[I + TA^{-1}S]^{-1}$ are clear from the diagrams:

if $\qquad A^{-1}S \to [A^{-1}S : q] \quad , \quad T \to \begin{bmatrix} T \\ \cdots \\ e \end{bmatrix}$

then $\qquad [I + TA^{-1}S] \to \left[\begin{array}{c:c} I + TA^{-1}S & T \\ \hdashline eA^{-1}S & 1 + eq \end{array}\right] \quad .$

There are three reasons for using this approach. First, a hierarchical structure allows partition of data, and each part of the information can be handled separately. In the above algorithm for handling the next block, we need the multipliers H_k of subordinate blocks only. Second, the use of a standard procedure for calculating the product (or any other) form of the inverse implies preliminary rearrangement of the rows and columns (in this case, it can be done for each block separately). Third, the hierarchical structure need be taken into account while updating the inverse only after every few iterations. Since the multipliers corresponding to different branches of the graph are commutative, we need implement the updating not for the entire matrix but rather only for those branches that have already accumulated a sufficiently large number of substitutions.

5. EXTENSIONS TO NONLINEAR OPTIMIZATION OVER LINEAR CONSTRAINTS

In conclusion, we may consider how to use the defined structure in nonlinear minimization subject to linear constraints. In many descent methods it is necessary to project some vector on the subspace defined by the system $A_z = 0$. To compute this projection, the matrix $(AA^T)^{-1}$ is needed. The rows of A are assumed to be linearly independent. Two cases may occur.

If A is an $m\times(m+d)$- matrix and α is small, then we may use the previously discussed algorithm and decompose the matrix in the form $A = B\cdot[I\ R]$, where B is square and nonsingular. Then the following equality holds:

$$(AA^T)^{-1} = B^{-T}[I + RR^T]^{-1}B^{-1} \quad .$$

The middle matrix can be rewritten

$$[I + RR^T]^{-1} = I - R[I + R^TR]^{-1}R^T \quad .$$

The order of the matrix $[I + R^TR]$ is α. As it is small, computing and storing the matrix is easy. The matrix B^{-1} may be decomposed according to the structure of matrix A, as stated above.

Note that this case occurs when the objective function differs from a linear one along directions in a low-dimensional subspace. If this is not the case, then α may be large and the above method becomes too expensive.

If our structure is purely horizontal, then either Householder or orthogonalization methods are convenient. Consider for example, the latter method. If in the course of orthogonalization we involve the rows beginning from the terminal blocks and moving to the root, then we obtain the decomposition $A = L \cdot Q$, where L is a lower triangular matrix, Q has orthogonal rows, and both matrices have the same structure as the matrix A. Then we can use the formula $AA^T = LL^T$.

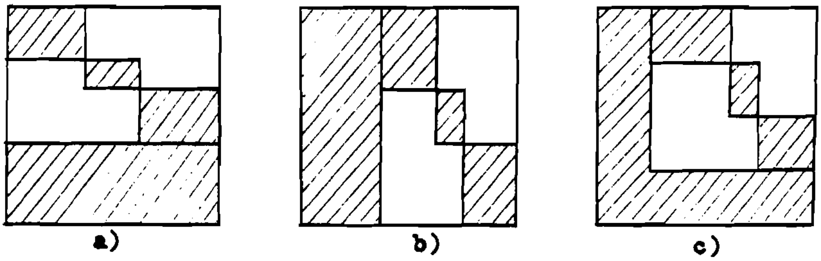


Figure 1

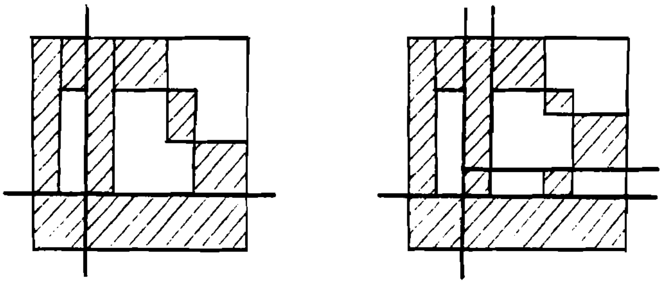


Figure 2

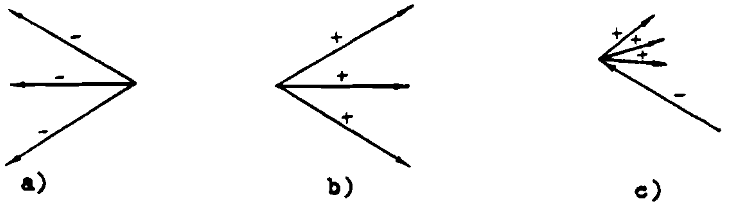


Figure 3

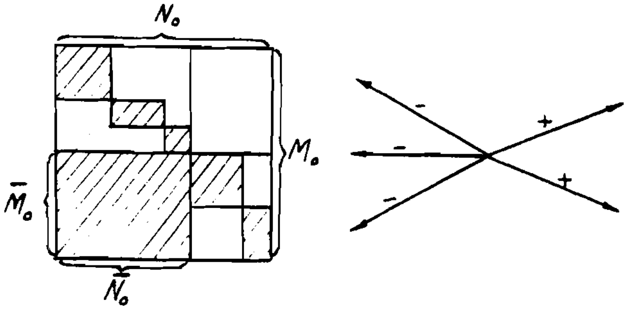


Figure 4

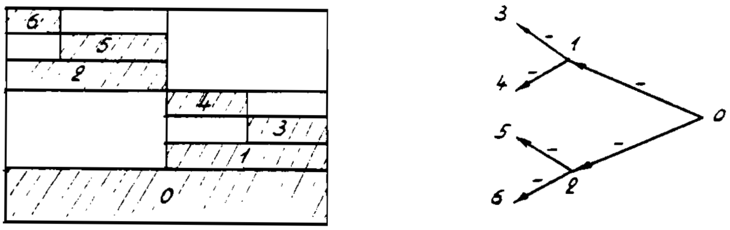


Figure 5

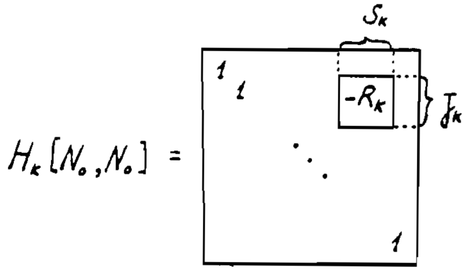


Figure 6

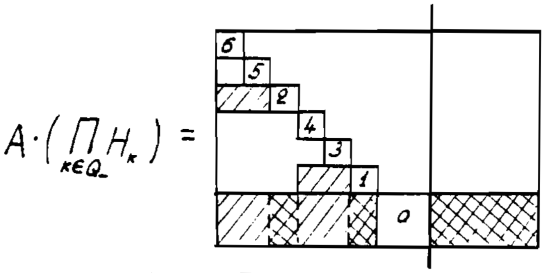


Figure 7

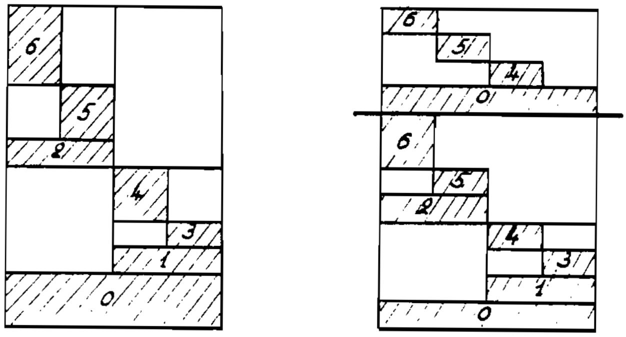


Figure 8

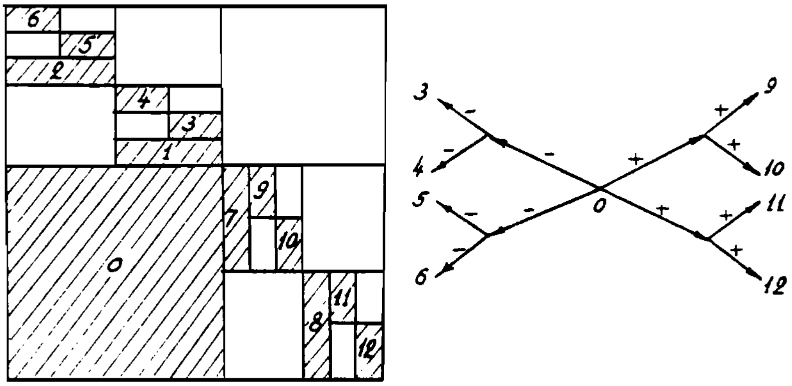


Figure 9

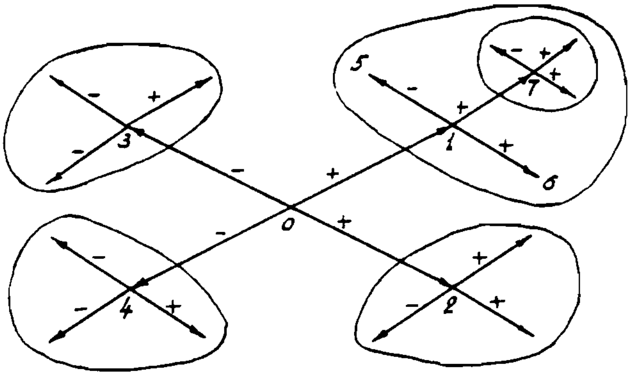


Figure 10

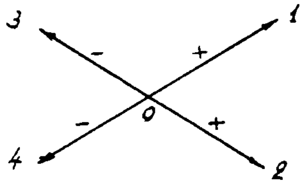


Figure 11

REFERENCES

Bulavskiy, V.A., and R.A. Zvyagina. 1977. A generalization of the block concept in linear programming. Dokl. Adad. Nauk SSSR 235(5):993-996 (in Russian). English translation in Soviet Math. Dokl. 18 (4):1061-1064.

Bulavskiy, V.A., and R.A. Zvyagina. 1978. Transformation of product form of matrix with hierarchical blocking. Optimizacija, Vyp. 22(39):53-68 (in Russian).

CONSTRUCTING LARGE LINEAR INPUT–OUTPUT SYSTEMS WITH RECURSIVELY GENERATED MATRICES

Joseph J.M. Evers and Tjibbe L. Knol

Department of Applied Mathematics
Twente University of Technology
Enschede, The Netherlands

A modelling technique is proposed that allows recursive construction of large-scale systems. The process–flow–transition structure introduced allows the integration of several polyhedral input/output (I/O)-processes. Such a structure can be transformed into a single I/O-process. A simple computer language is constructed which is oriented to this recursive definition of the input and output matrices of an I/O-process. All instructions required to generate these matrices as datafiles and their associated names as textfiles can be expressed in this language. The textfiles generated are part of the input of a reportwriter.

1. PROCESS-FLOW-TRANSITION STRUCTURES

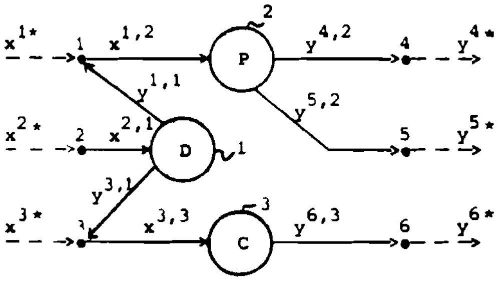
Particulary in the modelling of large economic or large production systems it is extremely important to maintain a stringent systematic in the form of a modular set up which is - at least from a logical view-point - invariant with respect to the complexity of the system. In particular, it appears that a module with a recursive nature offers excellent prospects for organizing data bases, numerical methods and reporting . results in a transparant manner. In addition it can be a natural starting point for computer-aided model design systems. Central theme of this study is a modelling method based on three elements: the polyhedral Input/Output process - being a special case of the more general concept (concave) Input/Output process, proposed elsewhere [1] - transition points, and commodity flows. After having introduced the concept I/O-process, an example will show how to integrate several I/O-processes with the help of "transition points" and "flows" into one single "process-flow-transition" structure. It will appear that this structure can be taken - after a self-evident transformation - as one single process with the same logical structure as our Input/Output process, implying that this modelling system possesses the desired recursive properties, indeed.

From an abstract physical view-point, an economic process can be characterized by a set of "feasible" Input/Output combinations, say: $S \subset \mathbb{R}^m \times \mathbb{R}^n$, where $\mathbb{R}^m$ represents the commodity space of the inputs, and where $\mathbb{R}^n$ stands for the commodity space of the outputs. Next, a preference ordering can be postulated by a utility function μ on S. In this context, an Input/Output process (or briefly I/O-process) is defined as a (bi-) function $\mu : S \subset \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^1$, satisfying the following hypotheses:

- $S \subset \mathbb{R}^m_+ \times \mathbb{R}^n$ (being a "minimal" hypothesis in order to support the distinction between inputs and outputs),
- for every $x, \bar{x} \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ so that $(x,y) \in S$, $\bar{x} \geq x$, it holds $(\bar{x},y) \in S$, $\mu(\bar{x};y) \geq \mu(x;y)$ (being a "free disposal" hypothesis concerning the inputs).

Within the context of a particular model formulating the I/O-process may bring out that the commodity spaces of the inputs and/or outputs are composed of several different commodity spaces. For instance $(\mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \times \ldots \times \mathbb{R}^{m_k})$ with respect to the inputs and $(\mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \ldots \times \mathbb{R}^{n_\ell})$ for the outputs. Then, instead of a single Input/Output pair (x,y), we have $((x^1,x^2,\ldots,x^k), (y^1,y^2,\ldots,y^\ell))$, with $x^j \in \mathbb{R}^{m_j}$, $y^j \in \mathbb{R}^{n_j}$. Of course, this does not affect the nature of our I/O-process.

Below we shall introduce the notion of a process-flow-transition stucture , with the help of a simple (perhaps somewhat cherché) model, represented by the "process-flow-transition" diagram:



In this example we have: three I/O-processes $P \subset \mathbb{R}^n_+ \times (\mathbb{R}^n_+ \times \mathbb{R}^n_+)$, $C \subset \mathbb{R}^n_+ \times \mathbb{R}^n_+$, $D \subset \mathbb{R}^n_+ \times (\mathbb{R}^n_+ \times \mathbb{R}^n_+)$, with a utility function μ on C only. We have six "transition points" numbered 1 to 6, we have eight internal commodity "flow vectors", three of them $x^{1,2}$, $x^{2,1}$, $x^{3,3}$ are input flows, the others $y^{4,2}$, $y^{5,2}$, $y^{6,3}$, $y^{3,1}$, $y^{1,1}$ are output flow vectors, and finally there are six "external flows" represented by the dotted arrows. Economically - after a suitable specification of the sets P, C, D - one may think about a configuration consisting of a production process (P), a consumption process (C), a distribution process (D), and two kinds of commodity stocks: the stocks for consumption and for productive purposes. All of these commodities may have a certain "life-time", in such a manner that the remaining part of the inputs $x^{1,2}$ (resp. $x^{3,3}$) after "passing" the production (resp, consumption) process is $\Lambda^p x^{1,2}$ (resp. $\Lambda^c x^{3,3}$), where Λ^p (resp. Λ^c) is a diagonal matrix with diagonal elements between 0 and 1. A part of the outputs of the production process $(y^{5,2})$, together with the "imports" to transition point numbered 2, can be added to the stocks for productive purposes or to the stock for consumptive purposes. This leads to the "distribution set" $D := \{(x^{2,1},(y^{1,1},y^{3,1})) \in \mathbb{R}^n_+ \times (\mathbb{R}^n_+ \times \mathbb{R}^n_+) \mid y^{1,1} + y^{3,1} \leq x^{2,1}\}$. In this context it is natural to put the utility function on P and D identical zero, and to take the utility function μ on C as the only internal basis for the valuation of flow realisations in the system.

Concerning the formal structure of the example we obviously have the constituting elements: commodity spaces, I/O-processes, transition points and flows. With each transition point only one commodity space is associated; thus, refering to this commodity space, we shall speak about the dimension of a transition point. Internal flows are located only between a transition point and an I/O-process; of course, the corresponding flow vector has the same dimension as the transition point. This assumption implies that flows can be indicated by associating with each I/O-process the connected transition points, both for the input side and for the output side. Further, the order has to be specified how the complete input and output flows of each I/O-process are composed of the separate flows; the formal set up will be defined with the help of set $\mathbb{C} := \{(\{i\}_{i=1}^{m}) \mid m = 1,2,\ldots,\}$. Thus, apart from flow facilities between the system as a whole and some "outside world", we define a process-flow-transition structure as a finite (or countable infinite) number of:

- transition points, indicated by a countable nonempty set M and a function $\xi : M \to \{1,2,\ldots\}$, refering to the dimensions,
- I/O-processes $\mu^j : S^j \subset \mathbb{R}^{\kappa_j} \times \mathbb{R}^{\omega_j} \to \mathbb{R}^1$, $j \in N$, N nonempty countable,
- input flows, associated with each I/O-process $j \in N$ by a function $\phi^j : M \to \{0,1,2,\ldots\}$, with $\phi^j(i) \neq 0$ for some $i \in M$, and with $(\{\phi^j(i)\}_{i \in M \mid \phi^j(i) \neq 0}) \in \mathbb{C}$,
- output flows, associated with each I/O-process $j \in N$ by a function $\psi^j : M \to \{0,1,2,\ldots\}$, with $\psi^j(i) \neq 0$ for some $i \in M$, and with $(\{\psi^j(i)\}_{i \in M \mid \psi^j(i) \neq 0}) \in \mathbb{C}$,

satisfying the following hypotheses:

- For each $\mu^j : S^j \to \mathbb{R}^1$, $j \in N$, the commodity spaces $\mathbb{R}^{\kappa_j}$, $\mathbb{R}^{\omega_j}$, possess the properties: (i) $\kappa_j = (\Sigma\ \xi(i)$, over $i \in M \mid \phi^j(i) \neq 0)$, $\omega_j = (\Sigma\ \xi(i)$, over $i \in M \mid \psi^j(i) \neq 0)$, (ii) $S^j \subset \mathbb{R}_+^{\kappa_j} \times \mathbb{R}_+^{\omega_j}$, (iii) for every $\bar{x}$, $x \in \mathbb{R}^{\kappa_j}$, $y \in \mathbb{R}^{\omega_j}$, so that $(x,y) \in S^j$, $\bar{x} \geqq x$, it holds: $(\bar{x},y) \in S^j$, $\mu^j(\bar{x};y) \geqq \mu^j(x;y)$.
- For each $i \in M$, there is a $j \in N$, so that $\phi^j(i) + \psi^j(i) > 0$ (i.e. each transition point is connected with at least one I/O-process).

In this context $\phi^j : M \to \{0,1,\ldots\}$ and $\psi^j : M \to \{0,1,\ldots\}$ will be called the input and output incidence functions of process j.

In the example one may define $M := \{1,2,\ldots,6\}$, $N := \{1,2,3\}$, $S^1 := D$, $\mu^1(x;y) := 0$ for all $(x,y) \in D$, $S^2 := P$, $\mu^2(x;y) := 0$ for all $(x,y) \in P$, $S^3 =: C$, $\mu^3(x;y) := \mu(x;y)$ for all $(x,y) \in C$,

$$\{\{\phi^j(i)\}\} := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \{\{\psi^j(i)\}\} := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In case there are flows between the system and some "outside world", the flows towards the system will be called import flows and the flows in the opposite direction, export flows. It is natural to associate these flows with the transition points, or formally with the elements of set M. Thus, the import and export flow structure will be indicated by the elements of sets $M^+ \subset M$, $M^- \subset M$, resp. (possible $M^+ := \emptyset$ or $M^- := \emptyset$), on the understanding that the following "import/export flow" hypotheses are satisfied:

- for every $i \in M^+$, there is a $j \in N$ so that $\phi^j(i) \neq 0$,
- for every $i \in M^-$, there is a $j \in N$ so that $\psi^j(i) \neq 0$;

in words: imports (exports resp.) are related to the inputs (outputs resp.) for at least one of the internal processes. Summarizing: in the presence of import and/or export facilities, the corresponding flows are considered as being a part of the process-flow-transition structure, provided the "import/export flow" hypotheses are satisfied.

Next we focus our attention to the problem how the magnitude of the flows between transition points and the I/O-processes is related to the composed input and output flows of the processes and to the import and export flows. Let us denote:

- the flow vector from transition point $i \in M$ towards I/O-process $j \in N$ by $x^{i,j} \in \mathbb{R}^{\xi(i)}$, with $x^{i,j} := 0$ if $\phi^j(i) = 0$,
- the flow vector from I/O-process $j \in N$ towards transition point $i \in M$ by $y^{i,j} \in \mathbb{R}^{\xi(i)}$, with $y^{i,j} := 0$ if $\psi^j(i) = 0$,
- the input vector of I/O-process $j \in N$ by $x^{*j} \in \mathbb{R}^{\kappa_j}$, where $\kappa_j := (\Sigma\, \xi(i)$, over $i \in M \mid \phi^j(i) \neq 0)$,
- the output vector of I/O-process $j \in N$ by $y^{*j} \in \mathbb{R}^{\omega_j}$, where $\omega_j := (\Sigma\, \xi(i)$, over $i \in M \mid \psi^j(i) \neq 0)$,
- the imports towards transition point $i \in M$ by $x^{i*} \in \mathbb{R}^{\xi(i)}$ with $x^{i*} := 0$ if $i \notin M^+$,
- the exports from transition point $i \in M$ by $y^{i*} \in \mathbb{R}^{\xi(i)}$ with $y^{i*} := 0$ if $i \notin M^-$.

Provided, the order how the input and output flow vectors x^{*j}, y^{*j} are composed of $\{x^{i,j}\}_{i \in M}$, $\{y^{i,j}\}_{i \in M}$, is specified by the incidence functions ϕ^j and ψ^j resp., we shall relate x^{*j} to $\{x^{i,j}\}_{i \in M}$ and y^{*j} to $\{y^{i,j}\}_{i \in M}$ with the help of functions $\mathbf{F}^i$, $i \in M$, assigning to all pairs $\{(x^{*j}, \phi^j)\}_{j \in N}$ and $\{(y^{*j}, \psi^j)\}_{j \in N}$, vectors $\mathbf{F}^i(x^{*j};\phi^j) \in \mathbb{R}^{\xi(i)}$ and $\mathbf{F}^i(y^{*j};\psi^j) \in \mathbb{R}^{\xi(i)}$ resp. in the following manner:

$$(1) \qquad \begin{cases} \mathbf{F}^i(x^{*j};\phi^j) := 0 \quad \text{if} \quad \phi^j(i) = 0, \text{ and in case } \phi^j(i) \neq 0: \\ \mathbf{F}^i(x^{*j};\phi^j) := (x^{*j}_{k+1}, x^{*j}_{k+2}, \ldots, x^{*j}_{k+\xi(i)}), \text{ where:} \\ \qquad k := 0 \quad \text{if} \quad \phi^j(i) = 1 \text{ or otherwise,} \\ \qquad k := (\Sigma\, \xi(\ell), \text{ over } \ell \in M \mid 0 < \phi^j(\ell) < \phi^j(i)), \end{cases}$$

and $F^i(y^{*j};\psi^j)$ being defined similarly. Given the formal structure as introduced before, F^i will be called the flow configuration function of transition point $i \in M$. One may verify that the functions F^i, $i \in M$ establish a one-one relation between the input and output flow vectors x^{*j}, y^{*j} and the process-transition flow vectors $x^{i,j}$, $y^{i,j}$ with $x^{i,j} := 0$ if $\phi^j(i) = 0$, and $y^{i,j} := 0$ if $\psi^j(i) = 0$. Thus, $\{(x^{i,j},y^{i,j})\}_{i \in M,\, j \in N}$ will be called an internal flow configuration if a sequence $\{x^{*j},y^{*j}\}_{j \in N}$, $(x^{*j},y^{*j}) \in S^j$, $j \in N$ exists such that $x^{i,j} = F^i(x^{*j};\phi^j)$, $y^{i,j} = F^i(y^{*j};\psi^j)$, $i \in M$, $j \in N$.

Next, an internal flow configuration $\{(x^{i,j},y^{i,j})\}_{i \in M,\, j \in N}$ will be called feasible with respect to the import and export flow vectors $\{x^{i*}\}_{i \in M}$, $\{y^{i*}\}_{i \in M}$ if, on each transition point $i \in M$ the - so called - commodity balance conditions:

$$y^{i*} + \Sigma_{j \in N}\, x^{i,j} = x^{i*} + \Sigma_{j \in N}\, y^{i,j},$$

are satisfied (provided the sums over $j \in N$ are well defined).

In some applications itmight be convenient to model an import or export flow as one of the internal flows. Within our formal set up, this can be done (somewhat tricky) by introducing the zero-dimensional real vector space $\mathbb{R}^0 := \{0\}$, 0 being the real number zero. Then an (artificial) I/O-process with its input part situated in $\mathbb{R}^0$ can be taken as a resource, whereas an I/O-process with its output part in $\mathbb{R}^0$ may be introduced as a final demand. Of course, the corresponding (artificial) transition points and related flows are associated with a "commodity" space $\mathbb{R}^0$.

Returning to our example: the diagram suggests that the proces-flow-transition structure itselves might be conceived as one single I/O-process on a "higher" abstraction level, just by taking the import and export flows as inputs and outputs and eliminating the corresponding feasible internal flows configurations by some optimality principle related to the internal utility function. We shall describe this transformation into an I/O-process, starting from the general structure as introduced before. However, in order to avoid complications we restrict ourselves to the case where the number of I/O-processes $|N|$ and transition points $|M|$ is finite. First of all we have to specify the order how the - what we shall call - external input vector and the external output vector are composed of the import and export flow vector resp.. In a similar manner as the internal flows constitute the internal input and output vectors, this can be done with the help of incidence functions; $\alpha: M \to \{0,1,2, \ldots\}$ for the external inputs, and $\beta: M \to \{0,1,2,\ldots\}$ for the external outputs. Having the import flows and export flows indicated by $M^+ \subset M$ and $M^- \subset M$ resp., these incidence functions have to satisfy the hypotheses $\{i \in M \mid \alpha(i) \neq 0\} = M^+$, $\{i \in M \mid \beta(i) \neq 0\} = M^-$, $(\{\alpha(i) \mid i \in M, \alpha(i) \neq 0\}) \in \mathbb{C}$, $(\{\beta(i) \mid i \in M, \beta(i) \neq 0\}) \in \mathbb{C}$. The corresponding dimensions are $r := (\Sigma\, \xi(i)$, over $i \in M^+)$ for the external inputs $\underline{x}$, and $s := (\Sigma\, \xi(i)$, over $i \in M$) for the external outputs $\underline{y}$. Now, utilizing our flow configuration functions $\mathbb{F}^i$, $i \in M$, the corresponding import and export flow vectors are $\mathbb{F}^i(\underline{x};\alpha)$, $\mathbb{F}^i(\underline{y};\beta)$, $i \in M$. Since input vectors are supposed to be nonnegative, the set of feasible external input/output combinations $\underline{S}$ can be defined:

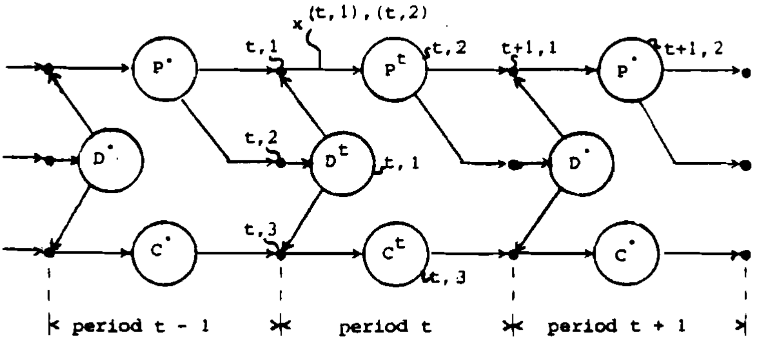
$$(2) \qquad \begin{cases} \underline{S} := \{(\underline{x},\underline{y}) \in \mathbb{R}^r_+ \times \mathbb{R}^s \mid \exists\, (x^{*j}, y^{*j}) \in S^j,\ j \in N:\ \forall i \in M: \\ \qquad \mathbb{F}^i(\underline{y};\beta) + \Sigma_{j \in N} \mathbb{F}^i(x^{*j};\phi^j) = \mathbb{F}^i(\underline{x};\alpha) + \Sigma_{j \in N} \mathbb{F}^i(y^{*j};\psi^j)\}; \end{cases}$$

in words: the set of combinations $(\underline{x},\underline{y}) \in \mathbb{R}^r_+ \times \mathbb{R}^s$ such that there exist corresponding feasible internal flow configurations. Next, the valuation of internal flow configurations can be effectuated on the basis of a weighted sum $\Sigma_{j \in N}\, \gamma^j \mu^j(x^{*j};y^{*j})$ over the separate utility functions μ^j, $\{\gamma^j\}_{j \in N}$ being a sequence of nonnegative weight factors. These considerations lead to an "external utility" functions $\underline{\mu}:\underline{S}\rightarrow \mathbb{R}^1 \cup \{+\infty\}$,defined by

$$(3)\qquad \begin{cases} \underline{\mu}(\underline{x};\underline{y}) := \\ \sup \Sigma_{j\in N} \gamma^j\mu^j(x^{*j};y^{*j}), \\ \text{over } (x^{*j},y^{*j}) \in S^j,\ j \in N, \\ \text{s.t. } \mathbb{F}^i(\underline{y};\beta) + \Sigma_{j\in N}\mathbb{F}^i(x^{*j};\phi^j) = \mathbb{F}^i(\underline{x};\alpha) + \Sigma_{j\in N}\mathbb{F}^i(y^{*j};\psi^j), \\ \qquad\qquad \forall i \in M. \end{cases}$$

As a consequence of the "import" hypothesis that for each $i \in M^+$ there is a $j \in N$ with $\phi^j(i) > 0$, we have that the external process satisfies the "free disposal" hypothesis, indeed; i.e. for each $\underline{x}$, $x \in \mathbb{R}^r$, $\underline{y} \in \mathbb{R}^s$ with $(\underline{x},\underline{y}) \in \underline{S}$, $x \geqq \underline{x}$ it holds $(x,\underline{y}) \in \underline{S}$, $\underline{\mu}(x;\underline{y}) \geqq \underline{\mu}(\underline{x};\underline{y})$. Thus, if $\underline{\mu}(\underline{x};\underline{y})$ is finite for all $(\underline{x},\underline{y}) \in \underline{S}$, then $\underline{\mu}:\underline{S}\rightarrow\mathbb{R}^1$ can be taken as an I/O-process.

With this fundamental transformation we have established the recursive nature of our process-flow-transition structure. Each I/O-process in such a structure might be generated by a (sub) process-flow-transition structure, and, the other way round, each process-flow-transition structure might be integrated as an I/O-process in a larger process-flow-transition structure. As an illustration of this recursivity, we consider a dynamic version of our example, as suggested by the diagram:



As period index we have introduced $t = 0,1,\ldots,h$, where $t := 0$ indicates the last past period, and where the positive integer h is the final period. The I/O-processes and transition points are indicated by the elements of the sets $\underline{N} := \{(t,1),(t,2),(t,3)\}_{t=1}^{h}$ and $\underline{M} := \{(t,1),(t,2),(t,3)\}_{t=1}^{h+1}$. The transition points are all n-dimensional. The incidence functions can be defined:

- $\phi^{(t,1)}(\theta,k) := 1$ if $(\theta,k) = (t,2)$,
- $\phi^{(t,2)}(\theta,k) := 1$ if $(\theta,k) = (t,1)$,
- $\phi^{(t,3)}(\theta,k) := 1$ if $(\theta,k) = (t,3)$, otherwise $:= 0$,

with respect to the input vectors, and for the output vectors:

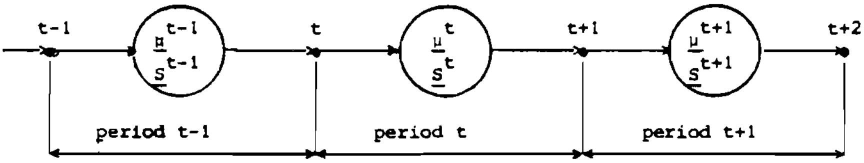
- $\psi^{(t,1)}(\theta,k) := 1$ if $(\theta,k) = (t,1)$, $:= 2$ if $(\theta,k) = (t,3)$,
- $\psi^{(t,2)}(\theta,k) := 1$ if $(\theta,k) = (t+1,1)$, $:= 2$ if $(\theta,k) := (t+1,2)$,
- $\psi^{(t,3)}(\theta,k) := 1$ if $(\theta,k) = (t+1,3)$, otherwise $:= 0$.

Next with the help of our flow configuration functions $\mathbb{F}^{i}$, all input and output flow vectors of the I/O-processes $\mu^{j}:S^{j} \to \mathbb{R}^{1}$, $j \in \underline{N}$ can be decomposed into process-transition flow vectors which have to satisfy the commodity balance conditions. Thus, the problem of finding an optimal trajectory, under exponential time discounting $\pi^{\uparrow t}$ with $\pi > 0$ ($\pi^{\uparrow t}$ stands for π power t) and a valuation of the terminal commodity stocks $q^{1},q^{2},q^{3} \in \mathbb{R}^{n}$, and given initial stocks $\underline{x}^{0} := (y^{(0,1)},y^{(0,2)},y^{(0,3)})$, can be written in the standard form:

$$
(4) \qquad \begin{cases} \sup \left(\left(\Sigma_{j \in \underline{N}}\, \gamma^{j}\mu^{j}(x^{*j};y^{*j})\right)\right) + \gamma^{h}\Sigma_{k=1}^{3} <q^{k},y^{*(h,k)}>), \\ \text{over } (x^{*j},y^{*j}) \in S^{j},\ j \in \underline{N}, \\ \text{s.t. } \Sigma_{j \in \underline{N}}\mathbb{F}^{i}(x^{*j};\phi^{j}) = \mathbb{F}^{i}(\underline{x}^{0};\alpha) + \Sigma_{j \in \underline{N}}\mathbb{F}^{i}(y^{*j};\psi^{j}),\ \forall i \in \underline{M}, \end{cases}
$$

where $\gamma^{(t,k)} := (\pi)^{\uparrow t}$, $t = 1,2,\ldots,h$, $k = 1,2,3$, where $<.,.>$ represents the inner product of two vectors, and where the incidence function α for the "imports" $\underline{x}^0$ is defined: $\alpha(\theta,k) := k$ if $\theta = 1$ and $k \in \{1,2,3\}$, otherwise $:= 0$. Note: in this example, the simplicity of the structure makes it possible to give an equally simple special formulation where the period index t is adopted explicitly.

Now, a more structural view-point can be obtain by taking the process-flow-structure for each separate period as one single I/O-process, and next linking these I/O-processes, dynamically. Thus, one may define for each period $t = 1,2,\ldots,h$ an (I/O)-process $\underline{\mu}^t:\underline{S}^t \subset \mathbb{R}^{3n}\times\mathbb{R}^{3n}\to\mathbb{R}^1 \cup \{+\infty\}$; this can be done in the same manner as the single period version of the model was transformed into an I/O-process. The resulting dynamic model can be characterized by the diagram:



The corresponding problem of finding an optimal trajectory can written as:

$$(5)\quad \begin{cases} \sup((\Sigma_{t=1}^{h} (\pi)^{\uparrow t}\ \underline{\mu}^t(\underline{x}^t;\underline{y}^t)) + (\pi)^{\uparrow h} <(q^1,q^2,q^3),\underline{y}^h>), \\ \text{over } (\underline{x}^t,\underline{y}^t) \in \underline{S}^t, \quad t = 1,2,\ldots,h, \\ \text{s.t. } \underline{x}^t = \underline{y}^{t-1}, \qquad t = 1,2,\ldots,h, \end{cases}$$

where $\underline{y}^0$ is the given initial state. Of course, one may fit this problem in our process-flow-transition structure. More generally the question arises, under what conditions a part of a process-flow-transition may be substituted by its formulation as I/O-process. As a matter of fact, one has to require only, that input and output flows can be distinguish, in such a manner that the hypotheses concerning the inputs are satisfied indeed; in that case one may conceive this as a <u>structural decomposition</u>, because the original input/output structure is preserved.

Beside this structural decomposition, one may apply <u>Lagrangean decomposition</u> techniques and the related shadow-price interpretation of Lagrange-multipliers. In order to introduce this approach briefly, let $\mu : S \subset \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^1$ be an I/O-process, let $u \in \mathbb{R}^m$ be a "price" vector for the inputs and let $v \in \mathbb{R}^n$ be a "price" vector for the outputs. Then the corresponding supremum of the "net-profit" ${}^*\mu(u;v)$ can be found by:

$$(6) \qquad {}^*\mu(u;v) := \sup(\mu(x;y) - \langle u,x\rangle + \langle v,y\rangle), \text{ over } (x,y) \in S.$$

Thus "net-profit" maximization leads to a function ${}^*\mu : {}^*S \subset \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^1$ - to be called the <u>dual I/O-process</u> - where the set ${}^*S$ (possible ${}^*S = \emptyset$) is defined:

$$(7) \qquad {}^*S := \{(u,v) \in \mathbb{R}^m \times \mathbb{R}^n \mid {}^*\mu(u;v) < +\infty\} .$$

In case ${}^*S \neq \emptyset$, it appears that ${}^*S \subset \mathbb{R}^m_+ \times \mathbb{R}^n$ (being an implication of the "free disposal" assumption on inputs), and that for each $u, \bar{u} \in \mathbb{R}^m$, $v \in \mathbb{R}^n$ with $(u,v) \in^* S$, $\bar{u} \geqq u : (\bar{u},v) \in {}^*S$, ${}^*\mu(\bar{u};v) \leqq {}^*\mu(u;v)$ (the latter being an implication of the hypothesis $S \subset \mathbb{R}^m_+ \times \mathbb{R}^n$). Obviously, in the opposite orientation, the dual I/O-process might be conceived (logically) as an I/O-process as well. In addition it is known that the epigraph of ${}^*\mu : {}^*S \to \mathbb{R}^1$ (i.e. the set $\{(u,v,\alpha) \in {}^*S \times \mathbb{R}^1 \mid {}^*\mu(u;v) \leqq \alpha\}$) is closed and convex. In case the hypograph of $\mu : S \to \mathbb{R}^1$ (i.e. the set $\{(x,y,\alpha) \in S \times \mathbb{R}^1 \mid \alpha \leqq \mu(x;y)\}$) is closed and convex, the function ${}^{**}\mu : {}^{**}S \subset \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^1$, defined:

$$(8) \qquad \begin{cases} {}^{**}\mu(x;y) := \inf\, ({}^{*}\mu(u;v) + \langle x,u\rangle - \langle y,v\rangle), \text{ over } (u,v) \in {}^{*}S, \\ {}^{**}S := \{(x,y) \in \mathbb{R}^m \times \mathbb{R}^m \mid {}^{**}\mu(x;y) > -\infty\}, \end{cases}$$

is the inverse transformation; i.e. ${}^{**}\mu : {}^{**}S \to \mathbb{R}^1$ is exactly the original function $\mu : S \to \mathbb{R}^1$ (cf. [1] or [2]).

Next let us consider the standard Lagrangean representation of maximization problem (3), with Lagrangean vectors $w^i \in \mathbb{R}^{\xi(i)}$, $i \in M$ on the transition points; to be written:

$$(9) \qquad \begin{cases} \sup(\Sigma_{j \in N}\, \gamma^j \mu^j(x^{*j};y^{*j}) - \\ \qquad - \Sigma_{i \in M} \langle w^i, (\mathbb{F}^i(\underline{y};\beta) + \Sigma_{j \in N} \mathbb{F}^i(x^{*j};\phi^j) - \\ \qquad\qquad - \mathbb{F}^i(x;\alpha) + \Sigma_{j \in N} \mathbb{F}^i(y^{*j};\psi^j))\rangle\}, \\ \text{over } (x^{*j},y^{*j}) \in S^j,\ j \in N. \end{cases}$$

Elaborating this expression one may verify that the supremum is finite if, and only if, there is a $\{(u^j,v^j)\}_{j \in N}$, $(u^j,v^j) \in {}^{*}S^j$ (each ${}^{*}\mu^j : {}^{*}S^j \to \mathbb{R}^1$ being the dual of $\mu^j : S^j \to \mathbb{R}^1$), such that, for all $i \in M$, $j \in N$: $\gamma^j \mathbb{F}^i(u^j;\phi^j) = w^i$, and $\gamma^j \mathbb{F}^i(v^j;\psi^j) = w^i$; in that case the value of the supremum is $\Sigma_{j \in N}\, \gamma^{j*}\mu^j(u^j;v^j) + \Sigma_{i \in M} \langle w^i, \mathbb{F}^i(\underline{x};\alpha) - \mathbb{F}^i(\underline{y};\beta)\rangle$. Consequently the corresponding "dual" problem takes the form:

$$(10) \qquad \begin{cases} \inf(\Sigma_{j \in N}\, \gamma^{j*}\mu^j(u^j;v^j) + \Sigma_{i \in M} \langle w^i, \mathbb{F}^i(\underline{x};\alpha) - \mathbb{F}^i(\underline{y};\beta)\rangle), \\ \text{over } (u^j,v^j) \in {}^{*}S^j,\ j \in N, \\ \qquad w^i \in \mathbb{R}^{\xi(i)},\ \ i \in M \\ \text{s.t. } \gamma^j \mathbb{F}^i(u^j;\phi^j) = w^i,\ \ \gamma^j \mathbb{F}^i(v^j;\psi^j) = w^i,\ \forall i \in M,\ j \in N. \end{cases}$$

Obviously, instead of the commodity balance restrictions appearing in the original - or primal - problem (3), the dual restrictions might be taken as (weighted) price equality condition on the transition points.

Now, provided the supremum in (3) is equal to the infimum (10) (which is generic in case the I/O-processes are concave), and provided $\{(\hat{u}^j,\hat{v}^j)\}_{j\in N}$ $\{\hat{w}^i\}_{i\in M}$ is a dual optimal solution, a necessary condition for optimality of $\{(\hat{x}^{*j},\hat{y}^{*j})\}_{j\in N}$ in (3) is, that each $(\hat{x}^{*j},\hat{y}^{*j})$ is optimal in the corresponding problem:

$$(11)\qquad \begin{cases} \sup(\mu^j(x^{*j},y^{*j}) - \langle\hat{u}^j,x^{*j}\rangle + \langle\hat{v}^j,\overset{*}{y}^j\rangle), \\ \text{over}\quad (x^{*j},y^{*j}) \in S^j; \end{cases}$$

in case of uniqueness this condition is sufficient, as well. Since in these optimality conditions the commodity balans restrictions are eliminated, the optimization is decomposed over the separate I/O-processes. Illustrative is the dual formulation of the abstract dynamic problem (5), which can be reduced to the form:

$$(12)\qquad \begin{cases} \inf(\pi\langle\underline{u}^1,\underline{y}^0\rangle + \Sigma_{t=1}^{h}\ (\pi)^{\dagger t}\,{}^{*}\underline{\mu}^t(\underline{u}^t;\underline{v}^t)), \\ \text{over}\ (\underline{u}^t,\underline{v}^t) \in {}^{*}\underline{S}^t, \quad t = 1,2,\dots,h, \\ \text{s.t.}\ \ \pi\underline{u}^{t+1} = \underline{v}^t, \quad t = 1,2,\dots,h-1, \\ \qquad \underline{v}^h = (q^1,q^2,q^3), \end{cases}$$

where q^1, q^2, q^3 are fixed given valuation vectors of the terminal state. Note: in this formulation of the dual problem the elimination of the dual variables $\{w^i\}_{i\in M}$ as introduced in (10), appears to be self-evident. Further, given a dual optimal solution $(\hat{\underline{u}}^t,\hat{\underline{y}}^t)$, $t = 1,2,\dots,h$, the decomposed problems (11), in fact are single period optimization problems.

We conclude this section with some summarizing remarks. First of all we found that the nature of the process-flow-transition structure is recursive or repetitive. Exploring this characteristic it is possible to describe such structures uniformly with the help of a simple recursive pointer system. The structure itselves gives self-evident starting-points for - what we have called - structural decomposition, at any desired abstraction level. Structural decomposition can be supported by standard Lagrangean decomposition techniques; the corresponding dual problem can be described with the help of the same recursive pointer system. Once the structure of the model is fixed in terms of such pointer system, it is possible to organize the data and the report writing along the same lines.

2. POLYHEDRAL PROCESS-FLOW-TRANSITION STRUCTURES

In this section we will specialize the domain of our I/O-process to a particular polyhedral set; to be precise, with a polyheder is meant any solution set in a finite dimensional real vector space of a finite system of linear inequalities and/or equalities. It will appear that every process-flow-transition structure where the processes are specialized in this manner, can be represented in a matrix form with again, a repetitive structure. Of course, if, in addition, the utility functions are linear or represented by a quadratic form, such matrices can be used directly in the standard optimization methods. Below the set of real m × n-matrices is denoted $\mathbb{R}^{m \times n}$; the set of real m × n-matrices with nonnegative elements is denoted $\mathbb{R}_{+}^{m \times n}$.

Formally, we define a polyhedral I/O-process as a (bi-)function $\mu : S \subset \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^1 \cup \{+\infty\}$, being representable by a quadruple consisting of a polyheder $P \subset \mathbb{R}^k_+$, a concave function $\nu : P \to \mathbb{R}^1$ its hypograph $\{(z,\alpha) \in P \times \mathbb{R}^1 \mid \alpha \leqq \nu(z)\}$ closed, a matrix $A \in \mathbb{R}^{m\times k}_+$, and a matrix $B \in \mathbb{R}^{n\times k}$ in the following manner:

$$(13) \quad \begin{cases} S := \{(x,y) \in \mathbb{R}^m \times \mathbb{R}^n \mid \exists z \in P : Az \leqq x,\ Bz = y\}, \\ \mu(x;y) := \sup \nu(z), \text{ over } z \in P, \text{ s.t. } Az \leqq x,\ Bz = y. \end{cases}$$

Observe that $P \subset \mathbb{R}^k_+$ and $A \in \mathbb{R}^{m\times k}_+$ implies: $S \subset \mathbb{R}^m_+ \times \mathbb{R}^n$; further, the inequality $Az \leqq x$ appearing in the definition implies that the "free disposal" hypothesis on inputs is satisfied. Clearly, in case μ is finite for all $(x,y) \in S$, the function $\mu : S \subset \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^1$ is an I/O-process, indeed. Obviously, for each $z \in P$, the quantities Az and Bz can be conceived as the "effective" inputs and the outputs resp. belonging to the process intensity vector z. Below, polyhedral I/O-processes will be denoted briefly, by the defining quadruples, the order of a polyheder, a function on that polyheder, the input matrix, and the output matrix. Thus, we will call a process-flow-transition structure polyhedral, if all processes are polyhedral I/O-processes specified by $(P^j \subset \mathbb{R}^{k(j)}, \nu^j : P^j \to \mathbb{R}^1, A^j \in \mathbb{R}^{\kappa(j)\times k(j)}_+, B^j \in \mathbb{R}^{\omega(j)\times k(j)})$, $j \in N$. Evidently, in this context it is possible to substitute input and output flow vectors x^{*j}, y^{*j} $(j \in N)$ by the expressions $A^j z^j$ and $B^j z^j$. Then, given the incidence functions $\phi^j : M \to \{0,1,\dots\}$ for the inputs, $\psi^j : M \to \{0,1,\dots\}$ for the outputs, $\alpha : M \to \{0,1,\dots\}$ for imports, and $\beta : M \to \{0,1,\dots\}$ for the exports (M being the index set for the transition points), the commodity balance conditions reduce to

$$(14)\quad \begin{cases} \mathbb{F}^i(\underline{y};\theta) + \Sigma_{j \in N}\, \mathbb{F}^i(A^j z^j) \leqq \mathbb{F}^i(\underline{x};\alpha) + \Sigma_{j \in N}\, \mathbb{F}^i(B^j z^j), \\ \qquad \text{for all } i \in M \text{ with } \phi^j(i) \neq 0 \text{ for some } j \in N, \\ \mathbb{F}^i(\underline{y};\beta) = \mathbb{F}^i(\underline{x};\alpha) + \Sigma_{j \in N} \mathbb{F}^i(B^j z^j), \\ \qquad \text{for all } i \in M \text{ with } \phi^j(i) = 0 \text{ for all } j \in N; \end{cases}$$

$\underline{x}$, $\underline{y}$ being the import and export flow vectors. Given these import and export flow vectors, $\{z^j\}_{j \in N}$, $z^j \in P^j$, will be called a feasible configuration of process intensity vectors if (14) is satisfied.

In connection with "commodity" space $\mathbb{R}^0$ in the definition of recourses and final demands as special I/O-processes, one may introduce the $0 \times k$-matrix as the $1 \times k$-matrix with all elements zero, and denote the set of $0 \times k$-matrices as $\mathbb{R}^{0\times k}$. Then <u>recourses</u> can be defined by taking $A \in \mathbb{R}^{0\times k}$ and <u>final demands</u> by taking $B \in \mathbb{R}^{0\times k}$.

As an illustration we specify our (single period) commodity distribution model of the first diagram as follows:

- the distribution process, indicated $j := 1$:
  $P^1 := \mathbb{R}_+^{2n}$, $\nu^1(z) := 0$ for all $z \in P^1$, $A^1 := (I^n, I^n)$, $B^1 := I^{2n}$
  (I^ℓ being the $\ell \times \ell$-identity matrix),
- the production process, indicated $j := 2$:
  $P^2 := \{(z', z'') \in \mathbb{R}_+^n \times \mathbb{R}_+^k \mid \bar{A}z'' \leqq r,\ \tilde{A}z'' \leqq z'\}$, given $\bar{A} \in \mathbb{R}_+^{\ell\times k}$, $r \in \mathbb{R}_+^\ell$,
  and given $\tilde{A} \in \mathbb{R}_+^{n\times k}$,
  $\nu^2(z) := 0$ for all $z \in P^2$, $A^2 := (I^n, 0)$ (0 being the $n \times k$ zero-matrix),
  $B^2 := \begin{pmatrix} \Lambda^P & 0 \\ 0 & \tilde{B} \end{pmatrix}$, given $\tilde{B} \in \mathbb{R}_+^{n\times k}$, and given the diagonal "duration matrix"
  Λ^P being introduced earlier,

- the consumption process, indicated $j := 3$:
  $P^3 := \mathbb{R}^n_+$, $v^3(z) := \langle p,z\rangle - \frac{1}{2}\langle z,Qz\rangle$, given $p \in \mathbb{R}^n_+$ and given $Q \in \mathbb{R}^{n \times n}$ symmetric positive semi-definite,
  $A^3 := I^n$, $B^3 := \Lambda^c$, given the diagonal "duration" matrix Λ^c,
- transition points with index set $M := \{1,2,3,4,5,6\}$,
- input and output incidence function $\phi^j : M \rightarrow \{0,1,\ldots\}$, $\psi^j : M \rightarrow \{0,1,\ldots\}$ as introduced earlier.

Instead of formulating the commodity balance conditions in terms of the flow configuration functions $\mathbb{F}^i$, one also may use sequences of matrices $(A^{i,j})_{i \in M, j \in N}$, $(B^{i,j})_{i \in M, j \in N}$ defined

$$A^{i,j} := (\mathbb{F}^i(a^j_{\cdot 1};\phi^j),\ \mathbb{F}^i(a^j_{\cdot 2};\phi^j),\ldots\ldots,\mathbb{F}^i(a^j_{\cdot k(j)};\phi^j)),$$
$$B^{i,j} := (\mathbb{F}^i(b^j_{\cdot 1};\psi^j),\ \mathbb{F}^i(b^j_{\cdot 2};\psi^j),\ldots\ldots,\mathbb{F}^i(b^j_{\cdot k(j)};\psi^j)),$$

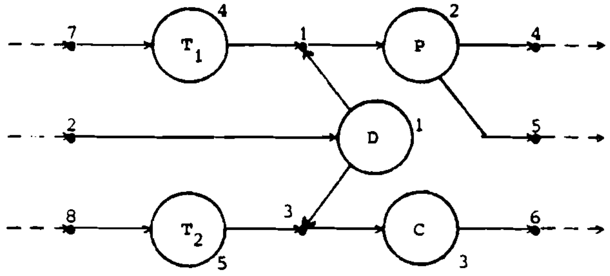
where $a^j_{\cdot \ell}$ ($b^j_{\cdot \ell}$ resp.) represents the ℓ-th column of matrix A^j (B^j resp.); observe that $A^{i,j} := 0$ ($B^{i,j} := 0$ resp.) if $\phi^j(i) = 0$ ($\psi^j(i) = 0$ resp.). Then, in the case that the processes and the transition points are ordered so that $N = \{1,2,\ldots,n\}$, $M = \{1,2,\ldots,m\}$, $n := |N|$, $m := |M|$, these matrices may be conceived as "block-elements" of composed input and output matrices $A^{**} := ((A^{i,j})^m_{i=1})^n_{j=1}$ and $B^{**} := ((B^{i,j})^m_{i=1})^n_{j=1}$, representing the complete input and output data in our example,these "super-matrices" take the form:

A^{**}	$j = 1$		$j = 2$		$j = 3$
$i = 1$	0	0	I^n	0	0
2	I^n	I^n	0	0	0
3	0	0	0	0	I^n
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0

B^{**}	$j = 1$		$j = 2$		$j = 3$
$i = 1$	I^n	0	0	0	0
2	0	0	0	0	0
3	0	I^n	0	0	0
4	0	0	Λ^p	0	0
5	0	0	0	$\tilde{B}$	0
6	0	0	0	0	Λ^c

Obviously the "pilars", numbered $j = 1,2,\ldots,n$ refer to the processes, whereas the "floors", numbered $i = 1,2,\ldots,m$ refer to the transition points.

Analogous to the transformation from a general process-flow-transition structure into an I/O-process, we also have such a transformation for the polyhedral case. However, since in the polyhedral case flows are expressed in terms of process intensity vectors, we have two differences. Firstly, we have to specify an ordering on the processes; this can be done with the help of an ordering function $\rho:\{1,2,\ldots,n\} \to N$, with $n := |N|$, $\rho(\{1,2,\ldots,n\}) = N$. Secondly, in order to preserve the nonnegativity convention concerning the inputs, external inputs (or imports) only may flow to transition points which are connected with processes only by input flows. Sometimes it will be necessary to extend the model in order to introduce suitable transition points. For instance, if, in our single period commodity distribution model, one likes to introduce external inputs at the transition points $i = 1,2,3$, one has to extend the process-flow-transition structure in the following manner:



where T_1 is represented by (P^4,ν^4,A^4,B^4), $P^4 := \mathbb{R}^n_+$, $\nu^4(z) := 0$ for all $z \in P^4$, $A^4 := I^n$, $B^4 := I^n$, and T_2 by (P^5,ν^5,A^5,B^5) being defined in the same manner.

Again restricting ourselves to a finite structure (i.e. $m := |M| < +\infty$, $n := |N| < +\infty$), and again assuming that the external input and output vectors are specified by flow incidence functions $\alpha : M \rightarrow \{0,1,2,\ldots\}$ and $\beta : M \rightarrow \{0,1,2,\ldots\}$ resp., our composed polyhedral I/O-process ($\underline{P} \subset \mathbb{R}^h$, $\underline{\nu} : \underline{P} \rightarrow \mathbb{R}^1$, $\underline{A} \in \mathbb{R}_+^{r \times h}$, $\underline{B} \in \mathbb{R}^{s \times h}$) is defined:

- $h := \Sigma_{j \in N} k(j)$, $r := (\Sigma\ \xi(i)$, over $i \in M$, s.t. $\alpha(i) \neq 0)$,

  $s := (\Sigma\ \xi(i)$, over $i \in M$, s.t. $\beta(i) \neq 0)$,
- $\underline{P} := \{(z^{\rho(1)}, z^{\rho(2)}, \ldots, z^{\rho(n)}) \in P^{\rho(1)} \times P^{\rho(2)} \times \ldots \times P^{\rho(n)} \mid$

  for all $i \in M$ with $\alpha(i) + \beta(i) = 0$:

  $\Sigma_{j \in N} A^{i,j} z^j \leqq \Sigma_{j \in N} B^{i,j} z^j$, if $\Sigma_{j \in N} \phi^j(i) \neq 0$,

  $\Sigma_{j \in N} B^{i,j} z^j = 0$, if $\Sigma_{j \in N} \phi^j(i) = 0\}$,

  $\rho : \{1,2,\ldots,n\} \rightarrow N$ being an ordering on the processes,
- $\underline{\nu}(z^{\rho(1)}, z^{\rho(2)}, \ldots, z^{\rho(n)}) := \Sigma_{j \in N} \gamma^j \nu^j(z^j)$,

  $\{\gamma^j\}_{j \in N}$ being a given sequence of nonnegative weight factors,
- $\underline{A} := ((\underline{A}^{\ell,j})_{\ell=1}^{m'})_{j=1}^{n}$, $m' := |\{\alpha(i)\}_{i \in M | \alpha(i) \neq 0}|$,

  $\underline{A}^{\alpha(i),j} := A^{i,\rho(j)}$, $i \in M$ so that $\alpha(i) \neq 0$, $j \in N$
- $\underline{B} := ((\underline{B}^{\ell,j})_{\ell=1}^{m''})_{j=1}^{n}$, $m'' := |\{\beta(i)\}_{i \in M | \beta(i) \neq 0}|$,

  $\underline{B}^{\beta(i),j} := B^{i,\rho(j)}$, $i \in M$ so that $\beta(i) \neq 0$, $j \in N$;

provided the following "import hypotheses" concerning the import incidence function are satisfied:

- for each $i \in M$ so that $\alpha(i) \neq 0$, there is at least one $j \in N$ with $\phi^j(i) \neq 0$,
- for each $i \in M$ so that $\alpha(i) \neq 0$, it hold: $\psi^j(i) = 0$ for all $j \in N$.

Returning to our example in the extended form: putting $\alpha(7) := 1$, $\alpha(2) := 2$, $\alpha(8) := 3$, $\alpha(i) := 0$ for $i \neq 7,2,8$, $\beta(4) := 1$, $\beta(5) := 2$, $\beta(6) := 3$, $\beta(i) := 0$ for $i \neq 4,5,6$, $\rho(j) := j$, $j = 1,2,\ldots,5$, the corresponding composed polyhedral I/O-process can be defined:

- $\underline{P} := \{(z^1,z^2,\dots,z^5) \in P^1 \times P^2 \times \dots \times P^5 |$
  $(I^n,0)z^2 \leqq (I^n,0)z^1 + z^4, \quad z^3 \leqq (0,I^n)z^1 + z^5\}$,
- $\underline{v}(z^1,z^2,\dots,z^5) := v^3(z^3)$,
-

$\underline{A}$	j = 1		j = 2		j = 3	j = 4	j = 5
i = 1	0	0	0	0	0	I^n	0
2	I^n	I^n	0	0	0	0	0
3	0	0	0	0	0	0	I^n

$\underline{B}$	j = 1		j = 2		j = 3	j = 4	j = 5
i = 1	0	0	Λ^p	0	0	0	0
2	0	0	0	$\tilde{B}$	0	0	0
3	C	0	0	0	Λ^c	0	0

Next, taking such a composed polyhedral I/O-process $(\underline{P}^t \subset \mathbb{R}^{h(t)}, \underline{v}^t : \underline{P}^t \to \mathbb{R}^1, \underline{A}^t \in \mathbb{R}_+^{3n\times h(t)}, \underline{B}^t \in \mathbb{R}^{3n\times h(t)})$ as an abstraction of a underlying process-flow-transition structure for a period $t \in \{0,1,\dots,h\}$, the polyhedral version of our dynamic model (5) can be written:

$$(16)\quad \begin{cases} \sup\ ((\Sigma_{t=1}^{h}\ (\pi)^{+t}\ \underline{v}^t(\underline{z}^t)) + (\pi)^{+h}\langle(q^1,q^2,q^3),\ \underline{B}^h\ \underline{z}^h\rangle), \\ \text{over}\ \ \underline{z}^t \in \underline{P}^t,\quad t = 1,2,\dots,h, \\ \text{s.t.}\ \ \underline{A}^t\underline{z}^t \leqq \underline{B}^{t-1}\ \underline{z}^{t-1},\quad t = 1,2,\dots,h, \end{cases}$$

where $\underline{z}^0$ is a given process intensity vector of the last past period $t = 0$. It should be clear that the recursive nature of composed input and output matrices, allow similar structural or Lagrangean decomposition structures as the general process-flow-transition structure.

3. MATRIXGENERATOR

The input and output matrices can be composed in a recursive manner. Solving problems with this structure by numerical methods means the availability of software to generate the matrices in the appropriate format. This software must come up to the next requirements:

a) Support of the modelling

- The set of instructions must allow the recursive composing of matrices, as proposed in the preceding sections.
- The manner to give instructions must be orientated to users with some mathematical background and some experience with this structure of modelling.
- Without handling the data, it must be possible to make a overview how the matrix is composed by the given instructions.
- Illegal instructions must be recognized and reported to the user. No illegal set of instructions may cause the matrixgenerator to collapse.

b) Datahandling requirements

- Without explicit mention it is not allowed to give instructions by which an already assigned value of a (sub)matrix is changed.
- To avoid not-defined situations, it is necessary to control the dimensions of the matrices during the elaboration of expressions and the assignment of values.
- A large number of data formats must be available and the implementation of another format or changing of a format must be easy. This enlarges the usability of the matrixgenerator.

c) Report writing

- The matrixgenerator must have facilities to associate names with rows or floors and columns or pilars. These names can be used for the reporting of the results.

d) Flexibility

- The software must be easy to understand, so one can make changes in the design of the matrixgenerator to one's own opinion.

e) Machine independency

- The size of the software must be small,to allow the matrixgenerator implementation on a microcomputer.
- The design of the software must allow that one builds an instruction set describing the recursive set-up of matrices on a small computer and runs that program with the data in on other computer.

In view of these requirements, we choose to design a language, with a simple syntax and a semantic directed at the recursive composing of matrices, in which all desired instructions can be represented. The grammar of this language is context-free and is expressed by so called syntax diagrams (see appendix B). The intention of these diagrams is twofold. First, it is of use to the user. With the help of the diagrams he can easily check whether an instruction is a well- or ill-formed sentence of the language.
Second , it is the starting-point for the implementation of the software. The diagrams are translated in a syntax parser. This parser is extended successively by error-recovery, code-generation and code-interpretation.

The stepwise refinement leads to a modular construction of the software [3]. A table-driven parser approach makes the language extensible, so it can be extended by further syntactic constructs. This requires declaration of the variables preceding the instruction-part, where they occur.
A value is assigned to a submatrix, this value can be the result of the evaluation of an expression. The various manipulations on matrices are defined as operators. Monadic operators are transposition of a matrix, inversion of a matrix, inversion of the entries; dyadic operators are multiplication, addition, subtraction and also the fusion of two matrices to one larger matrix.
Finally standard procedures organizes the input and output of data. These procedures can be adapted easily, when the format of the data is different from the formats already implemented.

Using recursion and dynamic data structuring techniques, it is possible to write short programs. Because of the availibility of Pascal-compilers for many computersystems the software of the matrixgenerator is written in the programming language Pascal.

3.1. Datatypes and operators

In our matrixgenerator language the standard implemented data types are: integer, real, matrix and file. The types integer and real are well known, for the other two types the following can be said:

The type matrix

An object of the type matrix is specified by the multiplicity of the representing matrix. Multiplicity is a generalization of the well-known dimension concept it indicates the number of matrices the super-matrix, introduced earlier, consists of. The number of rows and columns denotes the dimensions of a matrix, similary the number of floors and pilars denotes the multiplicity of a super-matrix, also called compound matrix.

The value of the type matrix is an element of the set of m × n matrices with entries belonging to $\mathbb{R}^1$. The following operators applied to operands of the type matrix yield a matrixvalue.

- The monadic operators; sign inversion (-), inversion (INVERT), transposition (TRANSPOSE) of a matrix.
- The dyadic operators; multiplication (*), addition (+), subtraction (-), augmentation by placing matrices side by side (COL) or by placing a matrix below (ROW) or diagonally below (DIA) an other matrix. The operators COL, DIA and ROW are defined as:

$$\text{A COL B} := (A,B)$$

$$\text{A DIA B} := \begin{pmatrix} A & O \\ O & B \end{pmatrix}$$

$$\text{A ROW B} := \begin{pmatrix} A \\ B \end{pmatrix} \ .$$

- The dyadic operators REPCOL, REPDIA and REPROW are the repetition variants of the operators COL, DIA and ROW. The left operand is of the type integer and the right of the type matrix. These operators yield a result of the type matrix and are defined as:

$$\text{K REPCOL A} := \underbrace{(A,A,\ldots\ldots,A)}_{\text{k times}}$$

$$\text{K REPDIA A} := \begin{pmatrix} A & & & \\ & A & & O \\ O & & \ddots & \\ & & & A \end{pmatrix}$$

$$\text{K REPROW A} := \left.\begin{pmatrix} A \\ A \\ \vdots \\ A \end{pmatrix}\right\} \text{k times}$$

- The scalar multiplication (*), is also implemented, in the case the left ●perand may be of the type integer or real. Let PI and A being a real resp. matrix variable, then is the expression PI * A a valid one.

The type file

A variable of the type file designates a sequence of data. The name of the variable and the name of the file on secondary storage of a computer are the same.

3.2. A matrixgenerator program

Every program expressed in the matrixgenerator language consists of a declaration part, where all objects are defined and a statement specifying the actions to be executed upon this objects.

A program consists of:

- program
  - declaration part
    - multiplicity definition
    - variable declaration
    - function declaration
  - statement part
    - assignment statement
    - change statement
    - repetitive statement
    - standard procedure

3.3. Declaration part

A declaration part consists of a multiplicity definition part, a variable declaration part and a function declaration part.

A multiplicity definition introduces an identifier as a synonym for the number of floors or pilars of a compound matrix. The use of multiplicity identifiers makes a program more readable. The user can also group the example dependent multiplicity of the matrices at the beginning of the program where it can be easily changed.

A variable declaration associates an identifier and a standard type with a new variable. In the declaration of a variable of the type matrix the multiplicity is denoted when the variable represents compound matrix. The number of floors and pilars is recorded next to the symbol matrix. An example of the variable declaration of a matrix A consists of 2 floors and 2 pilars

```
        VARIABLE

              A : MATRIX (2,2);
```

A function is a program part, which calculates a value of the type matrix. This value is used in the evaluation of an expression. The function declaration has the same form as a program,but is preceeded by a function-heading of the form:

```
        FUNCTION     identifier ;
```

3.4. Statement part

A statement can be an assignment, compound,change, repetitive statement or a standard procedure call. The assignment statement specifies that a newly computed value has to be assigned to a variable. The new value is obtained by evaluating an expression consisting of standard or variable operands, operators and function designators. The matrixgenerator language knows three standard objects of the type matrix:

MTRØ - Zero-matrix

MTR1 - E-matrix

The value of all entries is 1

IDEN - Identity matrix.

The dimensions of these standard objects must be recorded next to the appropriate symbol MTRØ, MTR1 or IDEN. In the case of an identity matrix it suffices to specify only one dimension.

The normal rules of operator precedence is observed in the evaluation of an expression. The monadic operators have the highest precedence, followed by the multiplying and repetitive operators and of lowest precedence, the adding operators.

In an assignment the variable and the expression must be of the same type. In case of a matrix value the dimensions of the variable and expressions must also correspond with each other. To every component of a compound matrix must be assigned a value. Only a change statement can change the value of a matrix variable. Assignment to variables of the type files is not possible.

The input and output of data is handled by the standard procedures READ and WRITE. A READ or WRITE procedure call associates a file on secondary storage of a computer with a matrix variable in the program. One of the parameters of these procedures designates the format of the transmitted data.

Other facilities are:

- Visualizing the values already assigned to submatrices of a compound matrix on that stage of the program.
- Associating names with floors or pilars of the compound matrix. These names are listed on a file and are part of the input of the report writer.
- Associating relational symbols (≤, ≥, =) with the rows of a compound matrix to meet the input data requirements of some L.P.-programs.

3.5. Software structuring

The construction of the matrixgenerator starts from the syntax diagrams. The diagrams are translated in an appropriate program structure. Such a program is able to analyse the syntax of an input sequence of symbols. The parser uses a scanner whose task it is to get the next symbol. The scanner also skips separates and recognizes reserved words, integer and real numbers, special symbols and identifiers. The parser collects the declared identifiers denoting the multiplicity of matrices variables and functions in a table. The occurence of an identifier within a statement then causes a search of this table to determine whether or not the identifier has been properly declared. Up to this point the parser can only determine whether or not the input sequence of symbols belongs to the matrixgenerator language.

As a first refinement (error-recovery) the parser is argumented with an appropriate error diagnostic system and after a syntax error the parsing process will be continued to find possibly further mistakes.
In a second refinement (code generation) the instructions (operators, assignments, standard procedures) are collected in an other table. For this purpose, it is necessary to list an expression in the postfix form sequence. An interpreter is added to generate a program in the programming language Pascal from the both tables with identifiers and instructions. The generated program can be executed, not necessarily by the same machine, with the help of specially written library programs.

REFERENCES

[1] J.J.M. Evers, "The Dynamics of Concave Input/Output Processes", in Convex Analysis and Mathematical Economics, (J. Kriens ed.), Lecture Notes in Economics and Mathematical Systems, 168, Springer Verlag (1979).

[2] R.T. Rockafellar, "Convex Analysis", Princeton University Press (1970).

[3] N. Wirth, "Algorithms + Data Structures = Programs", Prentice Hall, Englewood Cliffs (N.J.) (1976).

APPENDIX A: An example

A multiperiod process-flow-transition problem can be formulated as a dynamic I/O-process (16). Taking a planning horizon h := 4, the composed polyhedral I/O-process $(\underline{P}^t \subset \mathbb{R}^{6n+k(t)}, \nu^t : \underline{P}^t \to \mathbb{R}^1, \underline{A}^t \in \mathbb{R}_+^{3n\times(6n+k(t))}, \underline{B}^t \in \mathbb{R}^{3n\times(6n+\ell(t))})$ proposed in this paper, can be written as:

$$\sup((\Sigma_{t=1}^4 (\pi)^{+t}(<\underline{p}^t,\underline{z}^t> - \tfrac{1}{2}<\underline{z}^t,\underline{Q}^t\underline{z}^t>) + (\pi)^{+4} <(q_1,q_2,q_3),\underline{B}^4\underline{z}^4>),$$

$$\begin{aligned} \text{s.t.}\quad & \hat{\underline{A}}^t\underline{z}^t \leq \underline{r}^t, && t = 1,2,3,4,\\ & \underline{A}^t\underline{z}^t \leq \underline{B}^{t-1}\underline{z}^{t-1} && t = 1,2,3,4,\\ & \underline{z}^t \geq 0, \end{aligned}$$

where $\underline{z}^0$ is a given process intensity vector of the last period t = 0. The vector $\underline{p}^t$ and the matrix $\underline{Q}^t$ are

$\underline{p}^t$	1
1	0
2	0
3	p^t
4	0
5	0

$\underline{p}^t \in \mathbb{R}^{6n+k(t)}$

$\underline{Q}^t$	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	Q^t	0	0
4	0	0	0	0	0
5	0	0	0	0	0

$\underline{Q}^t \in \mathbb{R}_+^{(6n+k(t))\times(6n+k(t))}$

Because of the polyhedral structure the matrix $\hat{\underline{A}}^t$ and the vector $\underline{r}^t$ are defined as:

$\hat{\underline{A}}^t$	1		2		3	4	5
1	$-I^n$	0	I^n	0	0	$-I^n$	0
2	0	$-I^n$	0	0	I^n	0	$-I^n$
3	0 0	0 0	$-I^n$ 0	$\tilde{A}^t$ [illegible]	0 $\hat{0}$	0 0	0 0

$\hat{\underline{A}}^t \in \mathbb{R}^{(3n+\ell(t))\times(6n+k(t))}$,

$\underline{r}^t$	1
1	0
2	0
3	0 r^t

$\underline{r}^t := \mathbb{R}_+^{3n+\ell(t)}$

The matrices $\underline{A}^t$ and $\underline{B}^t$ are defined as:

$\underline{A}^t$	1		2		3	4	5
1	0	0	0	0	0	I^n	0
2	I^n	I^n	0	0	0	0	0
3	0	0	0	0	0	0	I^n

$\underline{B}^t$	1		2		3	4	5
1	0	0	Λ^p	0	0	0	0
2	0	0	0	$\tilde{B}$	0	0	0
3	0	0	0	0	Λ^c	0	0

The problem can be simplified to a concave quadratic programming problem.

$$\sup(\langle p^*,z^*\rangle - \frac{1}{2}\langle z^*,\underline{Q}^* z^*\rangle + \pi^{+4}\langle q^*,Tz^*\rangle)$$

$$\text{s.t. } (S+C)z^* \leq b^* + r^*$$

$$z^* \geq 0$$

For convencient we have supposed that the polyhedral $\underline{P}^t$ is time-independant and also the input and output matrices $\underline{A}^t$ and $\underline{B}^t$. To come to the short notation we have introduced:

z^*	1
1	$\underline{z}^1$
2	$\underline{z}^2$
3	$\underline{z}^3$
4	$\underline{z}^4$

,

p^*	1
1	$\pi\underline{p}$
2	$\pi^{+2}\underline{p}$
3	$\pi^{+3}\underline{p}$
4	$\pi^{+4}\underline{p}$

,

$\underline{Q}^*$	1	2	3	4
1	$\pi\underline{Q}$			
2		$\pi^{+2}\underline{Q}$		
3			$\pi^{+3}\underline{Q}$	
4				$\pi^{+4}\underline{Q}$

,

$$\begin{array}{c|c} q^* & 1 \\ \hline 1 & \begin{array}{c} \underline{q}^1 \\ \hline \underline{q}^2 \\ \hline \underline{q}^3 \end{array} \end{array} \quad , \quad \begin{array}{c|c|c|c|c} T & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 0 & \underline{B} \end{array} \quad , \quad \begin{array}{c|c|c|c|c} C & 1 & 2 & 3 & 4 \\ \hline 1 & \hat{\underline{A}} & & & \\ \hline 2 & & \hat{\underline{A}} & & \\ \hline 3 & & & \hat{\underline{A}} & \\ \hline 4 & & & & \hat{\underline{A}} \end{array}$$

The dimensions of $\hat{\underline{A}}$, $\underline{A}$ and $\underline{B}$ are not of the same size, this fact must be taken into account by the definition of matrix S.

$$\begin{array}{c|c|c|c|c} S & 1 & 2 & 3 & 4 \\ \hline 1 & \begin{array}{c} \underline{A} \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} \\ \hline 2 & \begin{array}{c} -\underline{B} \\ \hline 0 \end{array} & \begin{array}{c} \underline{A} \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} \\ \hline 3 & \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} -\underline{B} \\ \hline 0 \end{array} & \begin{array}{c} \underline{A} \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} \\ \hline 4 & \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} 0 \\ \hline 0 \end{array} & \begin{array}{c} -\underline{B} \\ \hline 0 \end{array} & \begin{array}{c} \underline{A} \\ \hline 0 \end{array} \end{array} \qquad \begin{array}{c|c} r^* & 1 \\ \hline 1 & \underline{r} \\ \hline 2 & \underline{r} \\ \hline 3 & \underline{r} \\ \hline 4 & \underline{r} \end{array}$$

Like matrix S, must vector b^* be defined in an appropriate form

$$\begin{array}{c|c} b^* & 1 \\ \hline 1 & \begin{array}{c} \underline{B}\,\underline{z}^0 \\ \hline 0 \end{array} \\ \hline 2 & 0 \\ \hline 3 & 0 \\ \hline 4 & 0 \end{array}$$

, with $\underline{z}^0$ the known identity vector in the last period t = 0.

$z^{*}, p^{*} \in \mathbb{R}^{24n+4k}, q^{*} \in \mathbb{R}^{3n}, \quad r^{*}, b^{*} \in \mathbb{R}^{12n+4\ell}$

$Q^{*} \in \mathbb{R}^{(24n+4k) \times (24n+4k)}, \; T \in \mathbb{R}^{3n \times (24n+4k)},$

$C, S \in \mathbb{R}^{(12n+4\ell) \times (24n+4k)}.$

The standard Lagrangean of the concave quadratic programming leads to the linear complementarity problem formulation

$$\begin{pmatrix} Q^{*} & (S+C)^{T} \\ -(S+C) & 0 \end{pmatrix} \begin{pmatrix} z^{*} \\ u \end{pmatrix} - \begin{pmatrix} v \\ y \end{pmatrix} = \begin{pmatrix} p^{*} + \pi^{+4} \, T^{T} q^{*} \\ -(b^{*} + r^{*}) \end{pmatrix}$$

$<(z^{*}, u), (v, y)> = 0$

$z^{*} \geq 0, \; u \geq 0, \; v \geq 0, \; y \geq 0.$

$u \in \mathbb{R}^{24n+4\ell}$, $v \in \mathbb{R}^{24n+4k}$ are the Lagrange multipliers and $y \in \mathbb{R}^{24n+4\ell}$ is the slack-vector of the constraints $(S+R)z^{*} \leq b^{*} + r^{*}$.

This problem can solved with the Lemke algorithm.

With the next program we want to generate the matrix M and vector d defined as:

M	1	2
1	Q^{*}	$(S+C)^{T}$
2	$-(S+C)$	0

d	1
1	$p^{*} + \pi^{+4} \, \underline{B}^{T} q^{*}$
2	$-(b^{*} + r^{*})$

$M \in \mathbb{R}^{(48n + 4(\ell+k)) \times (48n+4(\ell+k))}$, $d \in \mathbb{R}^{48n+4(\ell+k)}$ and $\underline{B}^{T} q^{*} = T^{T} q^{*}$.

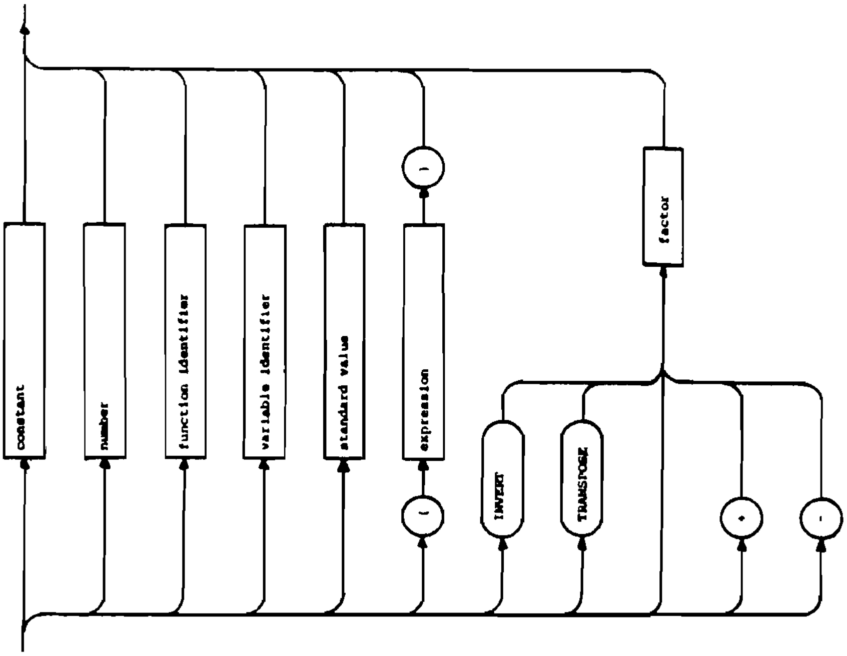
```
{program to generate the input of a computerprogram based on the Lemke
algorithm}
MULTIPLICITY
    ML = 3; MM = 5; MN = 4;
VARIABLE
    {dimension indicators for submatrices}
    K,L,N: INTEGER;
    {discount factor π and π↑2}
    PI, SQPI: REAL;
    {compound matrices}
    AT, A, B: MATRIX(ML,MM);
           S: MATRIX(MN,MN);
           M: MATRIX(2,2);
           D: MATRIX(2,1);
    {matrices and vectors}
    P, Q, R, PP, QQ, C, RR, RB: MATRIX;
    {auxiliary matrices and vectors}
    PT, QT, ASS, AST, PG, BS, RS, ZNUL, QD: MATRIX;
    {files on secondary storage with data}
    PFILE, QFILE, ASSFILE, ASTFILE, PGFILE, BSFILE, CGFILE, RFILE,
    ZNULFILE, QDFILE: FILE;
    I,J: INTEGER
BEGIN
    {the actual parameter FORMAT in the procedure call read and write
    designates an arbitrary format of the data}
    K := 6, L := 1; N := 3;
    {construction of vector p}
    READ(PFILE, FORMAT, PT); {read the data for vector p}
    P := MTRØ(2*N,1) ROW MTRØ(N+K,1) ROW PT ROW(2 REPROW MTRØ(N,1));
```

```
{construction of matrix Q̲}

READ(QFILE, FORMAT, QT);

Q := MTRØ(2*N,2*N) DIA MTRØ(N+K, N+K) DIA QT DIA(2 REPDIA MTRØ(N,N));

{composing matrix Â̲}

AT(1,1)  := -IDEN(N) COL MTRØ(N,N);

AT(1,2)  :=  IDEN(N) COL MTRØ(N,K);

AT(1,4)  := -IDEN(N);

AT(2,1)  :=  MTRØ(N,N) COL -IDEN(N);

AT(2,3)  :=  IDEN(N);

AT(2,5)  := -IDEN(N);

READ(ASSFILE, FORMAT, ASS); {read data for matrix Ã}

READ(ASTFILE, FORMAT, AST); {read data for matrix Ā}

AT(3,2)  := (-IDEN(N) COL ASS) ROW(MTRØ(N,N) COL AST);

{composing matrix A}

A(2,1)  := IDEN(N) COL IDEN(N);

A(2,2)  := MTRØ(N,N+K);

A(3,3)  := MTRØ(N,N);

A(1,4)  := IDEN(N);

A(3,5)  := IDEN(N);

{composing matrix B}

B(1,1)  := MTRØ(N,2*N);

READ(PGFILE, FORMAT, PG); {read data for matrix Λ^P}

B(2,1)  := PG COL MTRØ(N,K);

READ(BSFILE, FORMAT, BS); {read data for matrix BS}

B(2,2)  := MTRØ(N,N) COL BS;

READ(CGFILE, FORMAT, B(3,3)); {read data for matrix Λ^C}

FOR I := 4 TO 5 DO B(3,I)  := MTRØ(N,N);

{construction of vector r̲}

READ(RFILE, FORMAT, RS); {read data for vector r}

R := (3 REPROW MTRØ(N,1)) ROW RS;
```

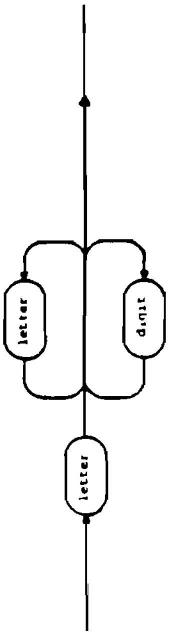
```
    PI := 0.9; {value of the discount factor}  SQPI := PI*PI;
    {construction of vector p*}
    PP := PI*P ROW SQPI*P ROW PI*SQPI*P ROW SQPI*SQPI*P;
    {construction of matrix Q*}
    QQ := PI*Q DIA SQPI*Q DIA PI*SQPI*Q DIA SQPI*SQPI*Q;
    {construction of matric C}
    C := AT DIA AT DIA AT DIA AT;
    {composing of matrix S}
    FOR I := 1 TO 4 DO S(I,I) := A ROW MTRØ(L,N);
    FOR I := 1 TO 3 DO S(I+1,I) := -B ROW MTRØ(L,N);
    {construction of vector r*}
    RR := R ROW R ROW R ROW R;
    {construction of vector b*}
    READ(ZNULFILE, FORMAT, ZNUL); {read data for vector z0}
    RB := B*ZNUL ROW MTRØ(L,N) ROW(3 REPROW MTRØ(N+L,1));
    {composing matrix M}
    M(1,1) := QQ;
    (M1,2) := TRANSPOSE(S+C);
    M(2,1) := -(S+C);
    {composing vector d}
    READ(QDFILE, FORMAT, QD); {read data for vector q*}
    D(1,1) := PP + SQPI*SQPI*TRANSPOSE(B)*QD;
    D(2,1) := -(RR + RB);
    {put the data in the file DATA}
    WRITE(DATA, FORMAT, M);
    WRITE(DATA, FORMAT, D);
END.
```

APPENDIX B: Syntax diagrams

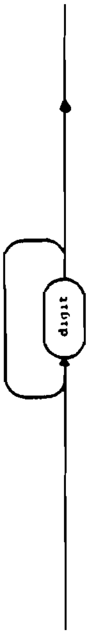
factor



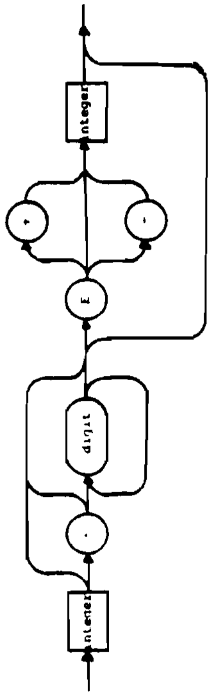
identifier



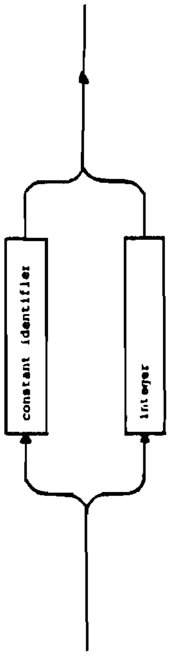
integer

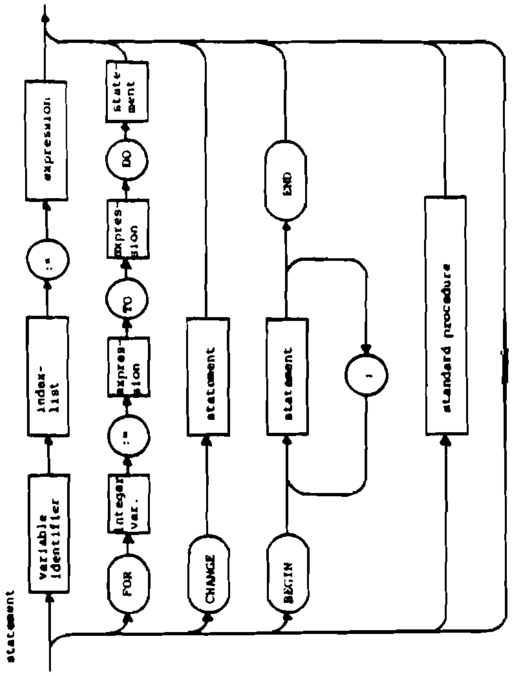


number

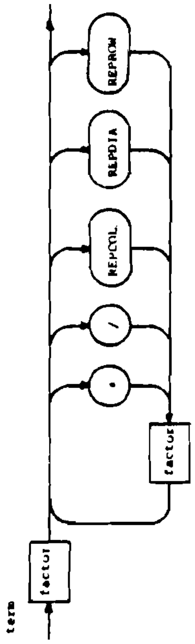


constant

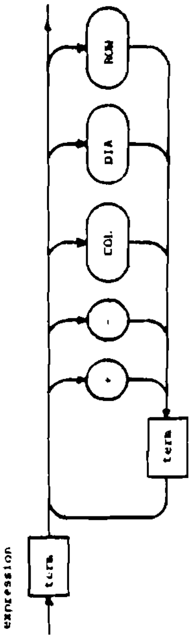




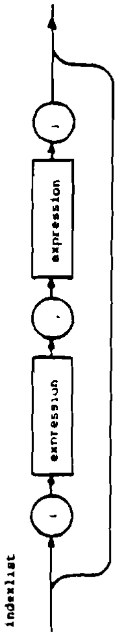
statement



term

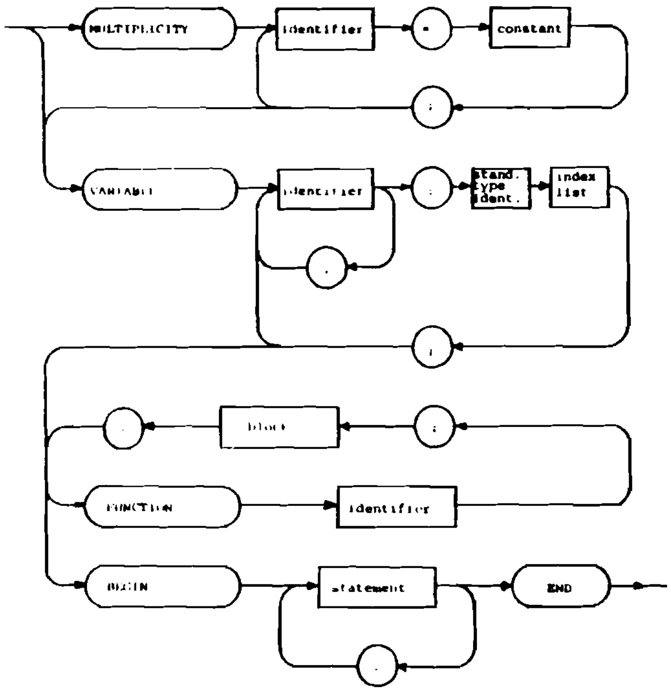


expression

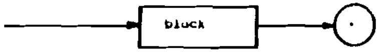


indexlist

block



program



APPENDIX C

Reserved words

BEGIN, CHANGE, COL, DIA, MULTIPLICITY, DO, END, FOR, FUNCTION, INVERT, REPCOL, REPDIA, REPROW, ROW, TO, TRANSPOSE, VARIABLE

Standard types

FILE, INTEGER, MATRIX, REAL

Standard objects

IDEN, MTRØ, MTR1

Standard procedures

NAME, READ, SIGNFLOOR, SIGNROW, VIEW, WRITE

COMPUTATIONAL EXPERIMENTS IN THE FORMULATION OF LARGE-SCALE LINEAR PROGRAMS

Gerhard Knolmayer

Institut für Betriebswirtschaftslehre
Universität Kiel

One of the decisions in the construction of a linear program is which formulation should be used. This paper explains why there is usually a very large number of equivalent formulations and reports on the computational behavior of these formulations. The usual textbook hypothesis — which claims that CPU-time increases with the cube of the number of constraints — is falsified by the reported experiments which suggest that advantage in reducing the number of rows may be overcompensated by an increase in the number of nonzeros.

COMPUTATIONAL EXPERIMENTS IN THE FORMULATION OF LARGE-SCALE LINEAR PROGRAMS

1. DECISIONS IN BUILDING A DECISION MODEL

Several (meta-) decisions have to be made in the construction of a decision model:

- Which section of reality should be modelled ?
- How accurate should one model this section of reality ?
- Which algorithm should be used ?
- Which people, computer, software should be employed ?
- Which formulation should be used for a given degree of accuracy of the model ?

The implementation of different answers on these questions will result in different benefits and costs of the decision model. The ultimate benefit of modelling is to gain insight into reality. In more detail one could distinguish between

Benefit from model accuracy
Benefit of the ease of understanding
the formulation
the solution
of the model.

On the other hand one can partition the costs of decision models into

Costs of model construction
Costs of collecting data
Costs of manipulating data
Costs of computation.

Many computational experiments have been performed in mathematical programming (MP). Most research has concentrated upon the comparison of algorithms and codes. Recently the need for research on a methodology of formulating MP-models has been expressed /18/.

Most computational experiments compare "costs", usually by giving CPU-time. Sometimes costs and benefits are compared, e.g. if the "quality" of solutions obtained is compared to the CPU-time needed for exact and heuristic algorithms. From this point of view one can distinguish the four areas of computational experiments shown in Fig. 1. These areas have been investigated to a very different extent. This paper concentrates on a cost-comparison of equivalent formulations by using a production code for linear programming (LP).

Type of comparison Object of comnarison	Cost-Comparison	Cost-Benefit-Comparison
Algorithms (Codes)		
Formulations	X	

Fig. 1: Types of experiments and topic of the paper (X)

2. The Need to Study Equivalent Formulations

We define equivalent formulations as models from which identical optimal activity levels can be derived (by using a report writer); the optimal values of the objective functions coincide. Several researchers have compared two equivalent formulations for linear or mixed-integer problems; I make references to the well-known studies of H.P.Williams /17;19/ on (mixed-) integer models and to the confrontation of linear product-mix-models with a "normal" resp. "aggregated" technological matrix /12;14,p.148-157;16,p.27-82/. Such comparisons suffer from the fact that often not only two but plenty of equivalent formulations exist. Especially if MP-models are generated from data bases containing information on every-day-operation the model builder has to decide which of the equivalent formulations should be generated. This decision determines the computational effort for matrix generation and for optimization.

In principle one can define basic relations from the data base as activities of the LP model and connect these activities by balance equations. But often the so emerging model will be unsolvable by production codes due to an enormous number of balance equations. A product-mix-model for a manufacturing firm with 400 final and 10000 intermediate products, with 30000 materials, 300 capacities and an average number of 5 operations for the manufactured products would need 82301 rows and 82400 structurals! Therefore it is desirable to generate a compact model by eliminating balance equations. Fig. 2 shows a small out of the very large number of equivalent LP-models that can be generated from a data base. In Fig. 2 the size of the model is measured by the number of rows. Data manipulation looks highly attractive from the usual textbook hypothesis that CPU-time grows with the cube of the number of rows /cf. e.g. 1, p.83;3,p.16;5,p.146;6,p.181;15,p.118;20,p.10/. Few authors claim that CPU-time is influenced by the density of the model, too /11,p.57;14,p.190/. By eliminating balance equations usually the number of rows is reduced and the density rises. Therefore rules of thumb are wanted which inform about presumable effects of matrix condensation. To support the decisions in model construction two types of experiments are necessary:

- Experiments of generating MP-models out of (non-specialized) data bases
- Experiments on the optimization behavior of equivalent formulations.

This paper reports on the second type of experiments.

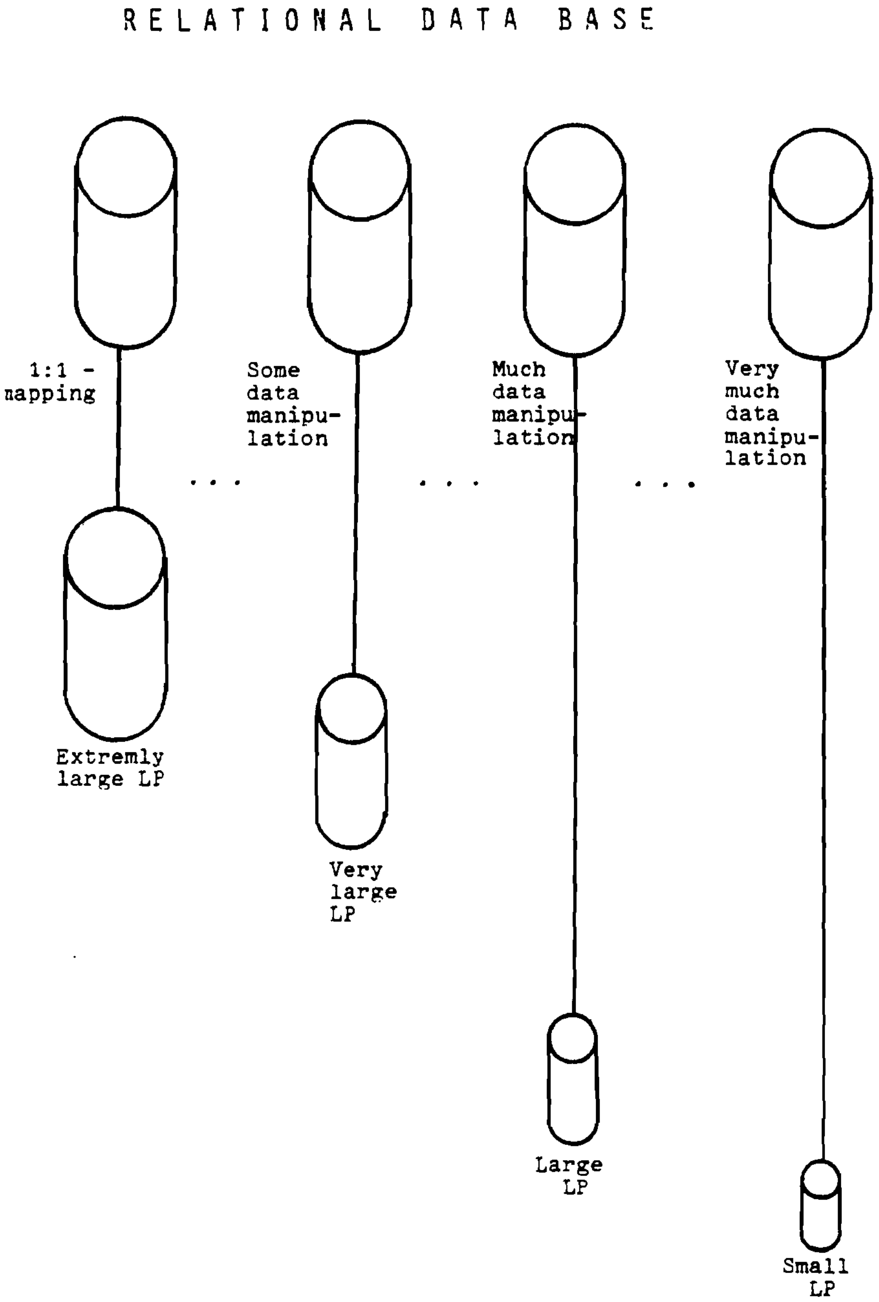


Fig.2: Different ways of matrix generation result in equivalent LP-models of different size

3. The Elimination of Balance Equations

3.1. Theory

Let

$$(1)\qquad \begin{aligned} c' \; x &\rightarrow \max ! \\ A^{M1} \; x &= b^{M1} > 0 \\ A^{M2} \; x &= 0 \\ x &\geq 0 \end{aligned}$$

be a feasible LP with slack and surplus, but without artificial variables. The indices of the constraints i form the set $M=M1 \cup M2$. Rows $i \in M2$ are called balance equations. Let $M2=M21 \cup M22$. We search for a transformed LP with new variables $\tilde{x}$

$$(2)\qquad \begin{aligned} c' \; T \, \tilde{x} &\rightarrow \max! \\ A^{M1} \; T \, \tilde{x} &= b^{M1} \\ A^{M2} \; T \, \tilde{x} &= 0 \\ T \, \tilde{x} &\geq 0 \end{aligned}$$

which is equivalent to (1) but computationally more appropriate. The latter requirement might be achieved if

$$A^{M22} \; T = 0 \; .$$

In this case $|M22|$ rows can be dropped as redundant.

If the original formulation (1) contains $p=|M2|$ balance equations there are at least 2^p equivalent formulations! To overcome the problems due to this enormous number of equivalent formulations we restrict the discussion to those formulations which arise by a sequential elimination of balance equations. The sequence can be determined heuristically by some plausible criterium. In the sequential procedure we have

$$T = \prod_{i=1}^{|M22|} T_i$$

where T_i is the transformation matrix for the i-th elimination of a balance equation.

It remains to determine matrices T_i in such a way that (1) and (2) are equivalent. After elimination of $f-1<p$ balance equations there exists a row $k \in M21$ so that

$$\bar{A}^k = A^k \prod_{i=1}^{f-1} T_i \ .$$

Let $\bar{POS}(k) = \{j \mid \bar{a}_{kj} > 0\}$ and $\bar{NEG}(k) = \{j \mid \bar{a}_{kj} < 0\}$. These sets are nonempty if there are no null variables. $\bar{x}_r$ $(r \in \bar{POS}(k))$ can be positive if and only if at least one $\bar{x}_j$ $(j \in \bar{NEG}(k))$ is positive. This "If-then"-relation allows $\bar{x}_r > 0$ and $\bar{x}_s > 0$ $(s \in \bar{NEG}(k))$ implicitly by a "coupled activity" $\bar{\bar{x}}_u > 0$. The coefficients of the coupled activity are computed as

$$\bar{\bar{a}}_{iu} = g\ (\ \bar{a}_{ir} \cdot |\bar{a}_{ks}| + \bar{a}_{is} \cdot |\bar{a}_{kr}|\ )$$

so that variable u has a zero in row k. g is an arbitrary positive factor; in the subsequent text we assume g=1.

All possible activity levels of the prior formulation can be expressed by $|\bar{POS}(k)| \cdot |\bar{NEG}(k)|$ coupled activities. After the transformation all variables $\bar{x}_j$ $(j \in \bar{POS}(k) \cup \bar{NEG}(k))$ can be deleted. In the matrix

$$T_f = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & \\ 0 & 0 & 0 & & 0 & |\bar{a}_{ks}| & \\ 0 & 1 & 0 & & 0 & 0 & \\ 0 & 0 & 0 & \cdots & 0 & |\bar{a}_{kr}| & \cdots \\ 0 & 0 & 1 & & 0 & 0 & \\ \vdots & & & & & & \\ 0 & 0 & 0 & & 1 & 0 & \end{bmatrix}$$

there are unity column vectors for the untouched activities and two non-zeros in those columns which represent coupled activities. We have $T_f \geq 0$ and therefore $T \geq 0$.Furthermore we get modified sets

$$\bar{\bar{M}}21 = \bar{M}21 - \{k\}$$

$$\bar{\bar{M}}22 = \bar{M}22 + \{k\} .$$

The transformation reduces the number of rows by one. The effect on the number of legitime variables depends on the number of positive and negative coefficients in row k:

$$\bar{\bar{n}} = \bar{n} + |\bar{P}OS(k)| . |\bar{N}EG(k)| - |\bar{P}OS(k)| - |\bar{N}EG(k)|$$

Table 1 shows how the number of legitimate (=non-artificial) variables changes with the sign of the non-zeros in the eliminated balance equation. The effects of condensation on model structure are illustrated in Table 2 and Fig. 3 for a refinery model given by Meyer-Steinmann /10,p.390-393/:
The points on the right hand of Fig. 3 characterize the original formulation; the effect of sequential data manipulation on problem structure is shown by going to the left. "Activity coupling" reduces the number of rows far more than e.g. the REDUCE-module of APEX-III.

From an economic point of view one can describe the condensation by the isoquant given in Fig. 4. It might happen that both formulations compared in literature are unsolvable on the system used while some equivalent formulations might be computationally well suited. The isoquant must be read from right to left.

		Number of		
$\|\bar{P}OS(k)\|$	$\|\bar{N}EG(k)\|$	new legitimate variables	legitimate variables deleted	Net Effect
3	1	3	4	-1
2	2	4	4	0
7	4	28	11	17

Table 1: Effects of Eliminating a Balance Equation k

Number of						
rows	structurals	variables	structural nonzeros	nonzeros	density	structural density
70	76	146	323	393	3.84540 %	6.07143 %
69	75	144	321	390	3.92512 %	6.20290 %
68	74	142	315	383	3.96645 %	6.25994 %
67	73	140	313	380	4.05117 %	6.39951 %
66	72	138	307	373	4.09530 %	6.46044 %
65	71	136	301	366	4.14027 %	6.52221 %
64	70	134	295	359	4.18610 %	6.58487 %
63	69	132	289	352	4.23280 %	6.64826 %
62	68	130	283	345	4.28040 %	6.71252 %
61	67	128	281	342	4.38012 %	6.87546 %
60	66	126	279	339	4.48413 %	7.04545 %
59	65	124	277	336	4.59267 %	7.22295 %
58	64	122	275	333	4.70605 %	7.40841 %
57	63	120	273	330	4.82456 %	7.60234 %
56	62	118	271	327	4.94855 %	7.80530 %
55	61	116	269	324	5.07837 %	8.01788 %
54	60	114	267	321	5.21442 %	8.24074 %
53	59	112	265	318	5.35714 %	8.47458 %
52	58	110	263	315	5.50699 %	8.72016 %
51	57	108	261	312	5.66449 %	8.97833 %
50	56	106	259	309	5.83019 %	9.25000 %
49	55	104	257	306	6.00471 %	9.53618 %
48	54	102	255	303	6.18873 %	9.83796 %
47	53	100	253	300	6.38298 %	10.15655 %
46	52	98	251	297	6.58829 %	10.49331 %
45	51	96	249	294	6.80556 %	10.84967 %
44	50	94	247	291	7.03578 %	11.22727 %
43	49	92	245	288	7.28008 %	11.62791 %
42	48	90	243	285	7.53968 %	12.05357 %
41	47	88	241	282	7.81596 %	12.50649 %
40	46	86	239	279	8.11047 %	12.98913 %
39	45	84	237	276	8.42491 %	13.50427 %
38	44	82	235	273	8.76123 %	14.05502 %
37	43	80	233	270	9.12162 %	14.64489 %
36	42	78	231	267	9.50855 %	15.27778 %
35	41	76	229	264	9.92481 %	15.95819 %
34	40	74	227	261	10.37361 %	16.69118 %
33	39	72	224	257	10.81650 %	17.40482 %
32	38	70	222	254	11.33979 %	18.25658 %
31	37	68	218	249	11.81214 %	19.00610 %
30	36	66	214	244	12.32323 %	19.81481 %
29	35	64	210	239	12.87716 %	20.68966 %
28	34	62	206	234	13.47926 %	21.63866 %
27	33	60	213	240	14.81481 %	23.90572 %
26	32	58	220	246	16.31300 %	26.44231 %
25	32	57	240	265	18.59649 %	30.00000 %
24	32	56	261	285	21.20536 %	33.98438 %
23	32	55	282	305	24.11067 %	38.31522 %
22	32	54	303	325	27.35690 %	43.03977 %
21	32	53	316	337	30.27853 %	47.02381 %
20	32	52	329	349	33.55769 %	51.40625 %
19	46	65	497	516	41.78138 %	56.86499 %
18	70	88	853	871	54.98737 %	67.69841 %
17	116	133	1375	1392	61.56568 %	69.72617 %
16	240	256	3075	3091	75.46387 %	80.07813 %

Table 2: Effects of condensation for the refinery model /10, p. 390 - 393/

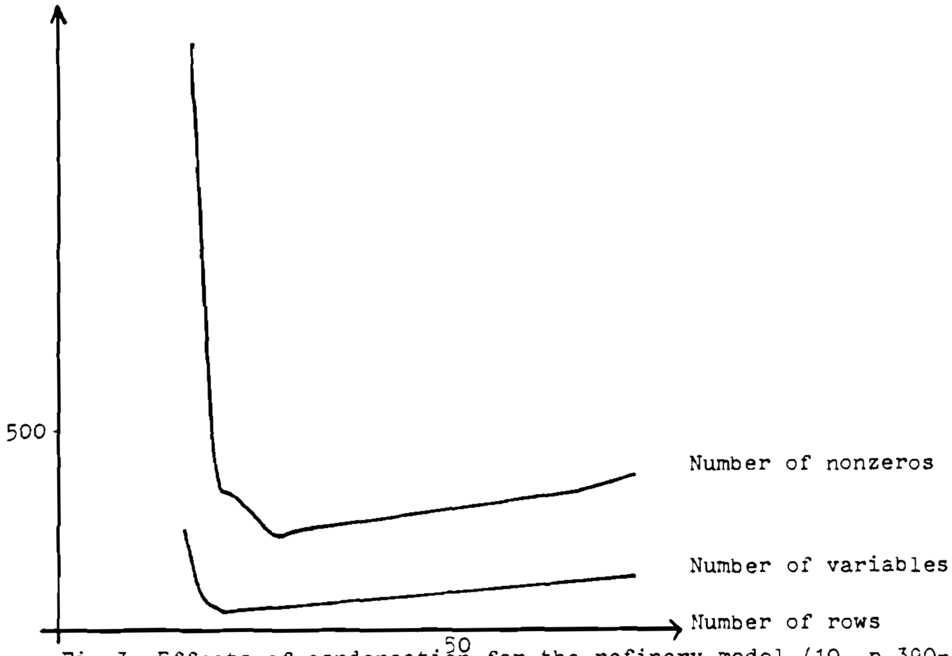


Fig.3: Effects of condensation for the refinery model /10, p.390-393/

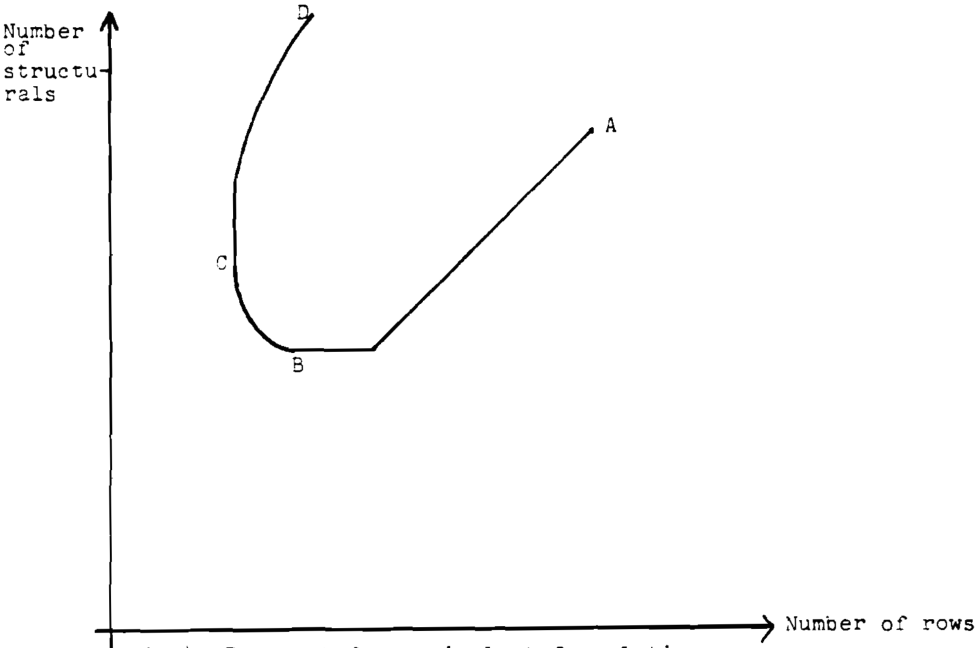


Fig.4: Isoquant for equivalent formulations

In economic theory only the part BC of the isoquant would be regarded as efficient. In this connection the part AB is efficient too because it takes resources to go from A to B. The part CD is explained by the reason that by eliminating a balance equation one or more bounds can become regular rows; this part of the isoquant is inefficient.

As soon as a formulation (2) is reached which is regarded computationally well suited the optimal levels of the activities $\tilde{x}^{\circledast}$ are determined. The optimal values of the original variables can be computed by

$$x^{\circledast} = T\,\tilde{x}^{\circledast} \; . \tag{3}$$

3.2. An Example

Consider a problem in which two final products x_1 and x_2 are produced by using a part, which can be either purchased (x_3) or produced (x_4):

$$\begin{array}{llllllllll}
\text{Max.} & 500\,x_1 & + & 1000\,x_2 & - & 200\,x_3 & - & 150\,x_4 & & \\
\text{s.t.} & 2\,x_1 & + & 1\,x_2 & & & + & 1\,x_4 & \leq & 1000 \\
 & & & 4\,x_2 & & & + & 2\,x_4 & \leq & 2000 \\
 & 1\,x_1 & + & 4\,x_2 & - & 1\,x_3 & - & 1\,x_4 & = & 0 \\
 & & & x_j \geq 0 & & \text{all } j & & & &
\end{array}$$

The first two constraints represent capacities, the third is the balance equation for the part. The definitions

$\tilde{x}_1$ ... quantity of final product 1 produced by using parts purchased

$\tilde{x}_2$ ... quantity of final product 1 produced by using parts produced by the firm

$\tilde{x}_3$ ... quantity of final product 2 produced by using parts purchased

$\tilde{x}_4$ ... quantity of final product 2 produced by using parts produced by the firm

allow the formulation

$$\begin{array}{lrrrrl}
\text{Max.} & 300\,\tilde{x}_1 + & 350\,\tilde{x}_2 + & 200\,\tilde{x}_3 + & 400\,\tilde{x}_4 & \\
\text{s.t.} & 2\,\tilde{x}_1 + & 3\,\tilde{x}_2 + & 1\,\tilde{x}_3 + & 5\,\tilde{x}_4 & \leq 1000 \\
 & & 2\,\tilde{x}_2 + & 4\,\tilde{x}_3 + & 12\,\tilde{x}_4 & \leq 2000 \\
 & & \tilde{x}_j = 0 & \text{all } j\ . & &
\end{array}$$

Formally such a reformulation can be obtained by multiplying the original coefficient matrix with the transformation matrix

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 4 & 0 \\ 0 & 1 & 0 & 4 \end{bmatrix}$$

We have $M2=\tilde{M}22=\{3\}$, $A^{\tilde{M}22}T=(1\ 4\ -1\ -1)$. $T=0$ and $\bar{\bar{n}}=4+2\cdot 2-2-2=4$. The optimal solution for the condensed LP is $\tilde{x}^{\otimes}=(250\ 0\ 500\ 0)'$. Optimal levels of the original variables can be determined by $x^{\otimes}=T\tilde{x}^{\otimes}=(250\ 500\ 2250\ 0)'$.

The different paths through the networks in Fig. 5 show that a general series transformation is employed for eliminating a balance equation.

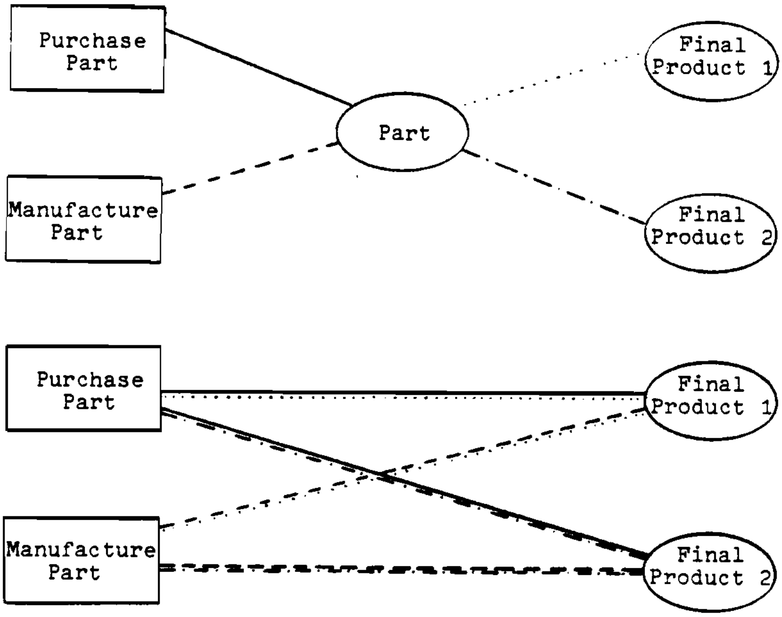


Fig.5: Elimination of a balance equation as general series transformation

4. The Design of the Computational Experiments

In Fig. 6 the general design of the experiments is sketched. The following tasks were necessary:

1. Problem Generation
   Computational experiments usually need a problem generatcr, especially if a statistical analysis is wanted. A problem

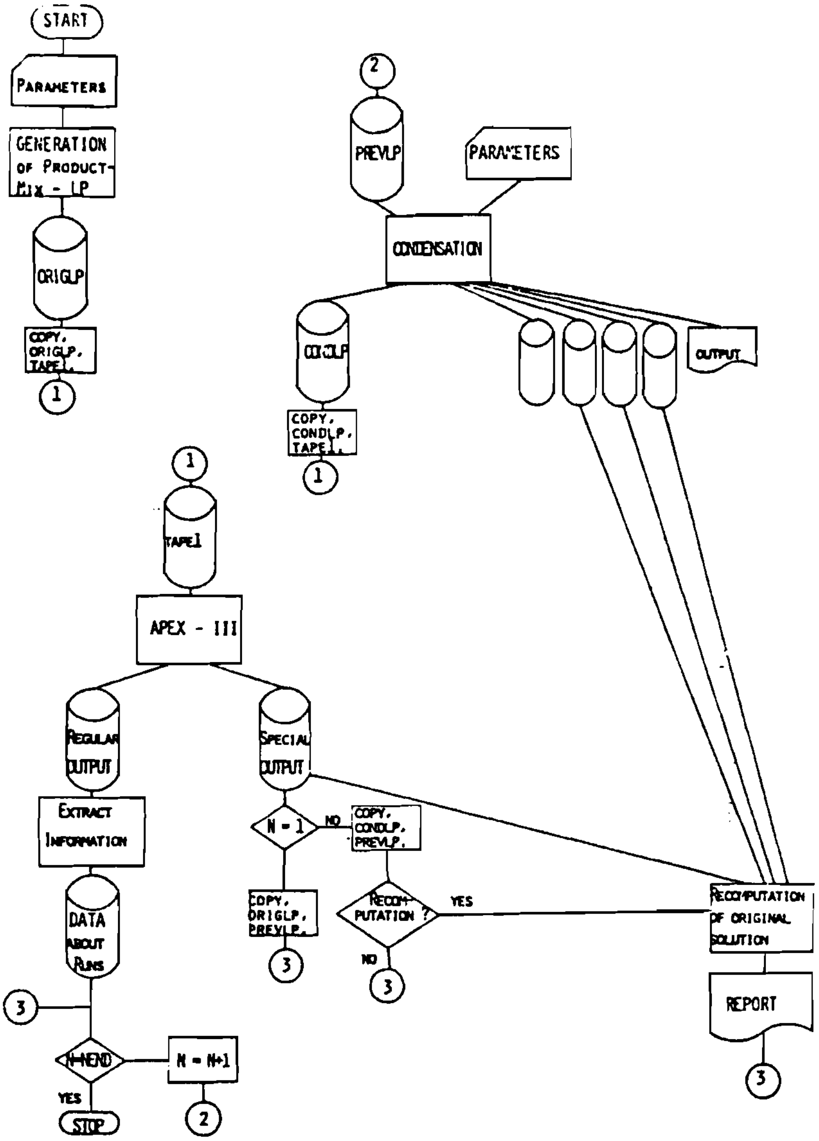


Fig. 6: Flow of information in computational experiments

generator PPPGEN was written in FORTRAN to create LP-models of product-mix-type. The user specifies the type of model to be created in much detail by setting 16 scalar and 3 vector parameters. One set of parameters generates different LP-problems with very similar but not identical structures by use of random numbers.

2. Preprocessing
   A FORTRAN-program performs the transformations discussed above. The user controls the order in which the balance equations are selected for elimination by 7 parameters. This selection is based on an estimation of the number of additional non-zeros an elimination might create.

3. Optimization
   Optimization was done by the in-core-system BASE-APEX-III using the standard parameters (except LOG=1) on a CYBER 74 under NOS/BE. The reported CPU-time was needed for optimization only. (The maximal deviation of CPU-time due to multiprogramming is only about 1 % on the system used.)

4. Postprocessing
   APEX-III produces an FORTRAN-accessible file which was used to determine the optimal levels of the activities in the original formulation. This postprocessing is based on (3) although the matrix T was not computed explicitly.

5. Recording Information about Optimization
   The regular OUTPUT-file of APEX-III contains information which is necessary to analyse the optimization behavior. This output-file was read by a program which recorded the structure of the model and the specifics of the solution process.

6. Regression Analysis
   The data collected in step 5 were examined by regression analysis. First the exponents of the variables in various

regression models were determined by SPSS' module for non-linear regression. The results were used to define transformed variables for a "linear" regression through the origin. Several hypotheses on the dependence of optimization time on model structure were compared by the coefficient of determination, R^2.

7. Control Experiments
   A test developed by Hoel /7/ was used to compare the best regression equation against the textbook-hypothesis.

A more detailed description of the experiments and the program lists are given in /9/.

5. Results of the Computational Experiments

Four problem classes and four problem sizes for each problem class have been examined. For each of the 16 cases 5 models were generated. Three problem classes were used to develop an appropriate explanation for the CPU-time observed; problem class 4 was used to control the results. Table 3 shows the approximate structure of the models in largest size. For problems of smaller size the figures in Table 3 have to be reduced by 25%, 50% and 75%.

All 80 formulations were condensed in five steps. In these steps a balance equation was eliminated if not more than a certain number of additional non-zeros were expected to arise. For problem class 3 and largest size Table 4 shows the effects of these condensations. The optimization was done by the procedures CRASH and PRIMAL of BASE-APEX-III. All formulations were optimized using constant field length RFL,100000_8 (=32768 decimal words of 60 bits each). Optimization time was reduced remarkably in the first phases of the condensation but in latter phases the condensation did not pay.

	Number of rows/columns in problem class			
	1	2	3	4
R O W S				
Objective function	1	1	1	1
Capacity constraints	25	25	25	25
Balance equations for final products	45	45	45	45
Balance equations for intermediate products	200	430	200	430
Balance equations for materials	250	20	250	20
	521	521	521	521
C O L U M N S				
Sales variables for final products	45	45	45	45
Sales variables for intermediate products	~20	~43	~20	~43
Purchase variables for intermediate products	~20	~43	~20	~43
Purchase variables for materials	250	20	250	20
Production variables for 45 final products	~135	~135	~59	~59
Production variables for intermediate products	~600	~1290	~260	~559
	~1070	~1576	~654	~769

Table 3: Structures of product-mix-models generated

Form. #	Max. add. nz	Average number of rows	columns	nonzeros	Aver. CPU-time	CPU - time estimated by textbook	"best" regression
1	-	521	651	3009	40.8	59.4	35.0
2	-5	351	481	2040	26.0	18.2	18.8
3	0	185	315	1619	8.4	2.7	7.8
4	30	144	286	1953	7.5	1.3	6.1
5	100	137	294	2288	10.8	1.1	6.0
6	1000	134	338	3206	10.4	1.0	6.5

Table 4: Effects of condensation in problem class 3

The data obtained from 317 LPs belonging to problem classes 1, 2 and 3 were analysed by regression models. Table 5 compares the quality of fit for several hypotheses and some other plausible equations.

	Regression equation	Explaining variables proposed by	R^2
(4)	$.00000042\ m^3$	/e.g. 1;3;5;6;15;20/	.701
	$.0627\ m$	/8/	.867
	$.000000015\ n^3$	/4;13/	.532
	$.0081\ m^{1.36}$		.889
	$.0147\ n^{1.05}$		.766
	$.0381\ nz^{.70}$		.697
(5)	$.0010\ m^{1.25}\ nz^{.33}$		.916
	$.0015\ m^{1.14}\ n^{.45}$		.913
	$.0293\ n^{2.53}\ nz^{-1.24}$		.816
	$.00085\ m^{2.35}\ [nz/(m.n)]^{.86}$		.912
(6)	$.00094\ m^{1.29}\ n^{-.17}\ nz^{.44}$		.916

Table 5: Comparison of regression models for explaining CPU-time in problem classes 1 to 3

The improvement of (6) over (5) is so small that

(7) $\text{PREDCPU} = a\ m^b\ nz^c$

is regarded as most suitable. For this model the approximate 95% confidence intervals for the exponents are computed in the non-linear regression by SPSS as

$$1.15 \leq b = 1.25 \leq 1.34$$
$$.26 \leq c = .33 \leq .39\ .$$

Although these intervals are tight they lead to rather wide confidence intervals for CPU-time.

The new assumption (5) was compared with the established hypothesis (4) via a test developed by Hoel /7/. This test leads to a linear regression of type

(8) $(OBSCPU - ESTHYP) = w \cdot (NEWHYP - ESTHYP)$

for additional data. The usefulness of NEWHYP is confirmed if w is significantly positive. Regression (8) gives a coefficient w=1.37 for 111 cases belonging to problem class 4; the 95% confidence interval is w±1.22. The t-value for regression (8) is 15.11. This value can be compared with the one-sided value for 95% and DF=110 which is 1.66. The scattergramm in Fig. 7 shows that in 93 of 111 cases the signs of the differences in (8) are identical. Therefore the new formula (5) predicts significantly better than the established hypothesis (4).

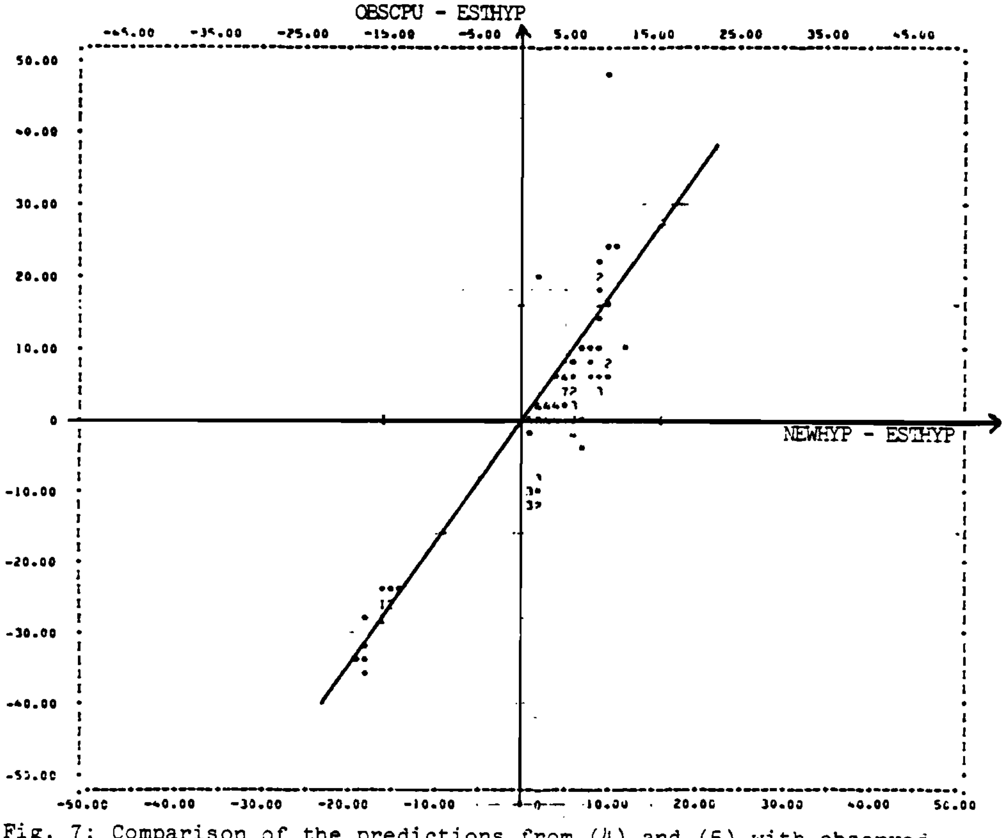


Fig. 7: Comparison of the predictions from (4) and (5) with observed CPU-time

6. SUMMARY

An identical LP-optimum usually can be obtained by many equivalent problem formulations. Data condensation is necessary if large models are generated from data bases containing information about every-day-operations. Most textbooks recommend to reduce the number of rows as much as possible. Our experiments show that the usual m^3-hypothesis is misleading and should be cancelled from textbooks. The experiments described above indicate that the number of non-zeros has remarkable influence on computational effort. The rule given by E.M.L.Beale /1,p.83/ that it is normally not worth saving a row by substituting a variable if this adds more than about half a dozen non-zeros remains useful in the light of our experimental results. The number of non-zeros may rise if the number of structurals is reduced; taking into account the effort for matrix generation one might propose an even easier rule of thumb:

"Eliminate balance equations only if

- the model is so large that the number of rows is a burden in the computational environment used
- the number of structurals is reduced by the elimination and the number of non-zreos rises only slightly."

For product-mix-models this rule suggests to use balance equations for products which have more than one way of preparation (e.g. make or buy; manufacturing variants) and more than one way of utilization (e.g. sell or process). Thus if options are available a "combined" formulation is recommended which differs from both formulations compared in literature.

If the resulting model is still to large the following actions could be taken into mind:

- May the problem be solved easier by codes with GUB-facilities and can such a code be made amenable ?

- Should one define in the first (i=1) LP only options which are expected to be optimal and generate for optimization run i+1 new variables for options which improve the solution of run i ? These candidates can be determined by the i-th dual solution.
- Is it possible to develop better algorithms for <u>dense</u> LP-problems ?

If all these questions have to be denied there is an effective "solution constraint" on the LP originally proposed. In this case one must take into account the potential benefits of differently accurate models and judge whether a less accurate model will allow enough insight into the real-world-problem that it pays to develop this less accurate model.

Main Symbols

DF	Degrees of freedom
ESTHYP	CPU-time predicted by the established hypothesis (4)
m	number of rows
M1	set of indices i with b_i 0
M2	set of indices i with $b_i=0$ (balance equations)
M21 $\subseteq$ M2	set of indices i for balance equations not eliminated
M22 $\subseteq$ M2	set of indices i for balance equations eliminated
n	number of structurals
nz	number of non-zeros
NEG(k)	set of indices j with $a_{kj}<0$
NEWHYP	CPU-time predicted by the new assumption (5)
OBSCPU	Observed CPU-time
POS(k)	set of indices j with $a_{kj}>0$
PREDCPU	Predicted CPU-time
R^2	Coefficient of determination
RFL	Requested Field Length
|SET|	number of elements in a SET
T, T_i	Transformation matrices

REFERENCES

/1/ Beale, E.M.L., Mathematical Programming in Practice,Pitman Publishing: London et al. 1968

/2/ Beale, E.M.L., The current algorithmic scope of mathematical programming systems, in: Mathematical Programming Study 4, ed. by M.L.Balinski and E. Hellerman, North-Holland: Amsterdam 1975, 1-11

/3/ Bradley, S.P., Hay, A.C., Magnanti, T.L., Applied Mathematical Programming, Addison-Wesley: Reading et al. 1977

/4/ Bramsemann, R., Controlling, Gabler-Verlag: Wiesbaden 1978

/5/ Driebeek, N.J., Applied Linear Programming, Addison-Wesley: Reading et al. 1969

/6/ Hillier, F.S., Lieberman, G.J., Operations Research, 2nd ed., Holden-Day: San Francisco et al. 1974

/7/ Hoel, P.G., On the choice of forecasting formulas, in: Journal of the American Statistical Association 1947, Vol. 42, 605-611

/8/ Holm, S., Klein, D., Size Reduction of Linear Programs with Special Structure, Skrifter fra Institut for Historie og Samfundsvidenskab, Odense Universitet: Odense 1975

/9/ Knolmayer, G., Programmierungsmodelle für die Produktionsprogrammplanung, Ein Beitrag zur Methodologie der Modellkonstruktion, Birkhäuser-Verlag: Basel-Boston-Stuttgart, to appear.

/10/ Meyer, M., Steinmann, H., Planungsmodelle für die Grundstoffindustrie, Physica-Verlag: Würzburg-Wien 1971

/11/ Mitra, G., Theory and Application of Mathematical Programming, Academic-Press: London-New York-San Francisco1976

/12/ Müller-Merbach, H., Switching Between Bill of Material Processing and the Simplex Method in Certain Linear Large-Scale Industrial Optimization Problems, in: Decomposition of Large-Scale Problems, ed.by D.M.Himmelblau, North-Holland/American Elsevier: Amsterdam-London-New York 1973, 189-199

/13/ Shah, A.A., Saber, J.C., A Transformation to Improve Computing in Linear Programs, Paper presented at the Joint National ORSA/TIMS Meeting at Las Vegas, November 1975

/14/ Smith, D., Linear Programming in Business, Polytech Publishers: Stockport 1973

/15/ Wagner, H.M., Principles of Operations Research, Prentice-Hall: Englewood Cliffs 1969

/16/ Wiggert, H., Programmoptimierung im Maschinenbau mit Hilfe der linearen Planungsrechnung, Beuth-Vertrieb: Berlin-Köln-Frankfurt am Main 1972

/17/ Williams, H.P., Experiments in the formulation of integer programming problems, in: Mathematical Programming Study 2,ed. by M.L.Balinski, North-Holland: Amsterdam 1974, 180-197

/18/ Williams, H.P., The Formulation of Mathematical Programming Models, in: Omega 1975, Vol. 3, 551-556

/19/ Williams, H.P., The reformulation of two mixed integer programming problems, in: Mathematical Programming 1978, Vol. 14, 325-331

/20/ Williams, H.P., Model Building in Mathematical Programming, Wiley-Interscience: Chichester et al. 1978

PROBLEMS OF SYMBOLOGY AND RECENT EXPERIENCE
(Or, Where Improvements Won't and May Come From)

William Orchard-Hays

Energy Information Administration
U.S. Department of Energy
Washington, D.C.

Significant improvements in use of standard large-scale LP, and related modeling which depends essentially on generated LP approximations or submodels, will *not* result from improvements in optimizing algorithms or even directly from improved computer implementations, only slightly from improved inversion and transformation schemes, and possibly somewhat from decomposition techniques applied at a high level. This will be true for at least several years.

While these statements are made.emphatically and must be taken with appropriate qualification, this paper discusses the background and results of its author's reflections along these lines, as presented to the Workshop.

INTRODUCTION

It had been my intention to begin with a somewhat brash statement and then proceed to defend it. The purpose of such an approach is, of course, to try to push aside conventional wisdom and habitual patterns in order to present a fresh viewpoint more clearly and forcefully. Had I been one of the first speakers, I would have done so and it would have been unfortunate. The differences in areas of interest, which have thus far been presented at this meeting under the heading of Large-Scale Linear Programming, show how diverse the subject actually is. One can only make broad statements, brash or otherwise, within very carefully defined limits.

All this is only a long way of saying that I have already learned, or at least been reminded of, a good deal at this Workshop. It may be helpful to others to summarize one set of observations. I believe the attendees are a representative cross-section of the field and it is clear that we represent at least four major areas of interest. (Of course, some of us wear different hats at different times.)

(1) The theoreticians. This is the largest interest group represented. I do not mean to intimate that the subjects and techniques discussed have no practical application. However, the orientation has a traditionally mathematical and academic flavor. Much of this work is fundamental to practical applications or improvements. Some falls by the wayside.

(2) The free-lance consultants. This area has been most clearly represented here by Marshall Fisher's presentation. My intended remarks would have been close to insulting to him. Conversely, his reported results would be incredible to me if I did not place them in proper context. Remarkable results can be achieved on some projects by those clever enough to perceive the proper approach. But our two areas of interest have a very small intersection.

(3) The algorithmic-system engineers. Several presentations represent this area and others, including myself, most frequently work in it. We may distinguish two sub-classes:

> (a) The in-house project director. This is most clearly represented by Dr. Aonuma's discussion of building a computational system using MPSX/370 as a base. Of course, theoretical and consulting-like work is involved but a specific, tailored system is the goal.

(b) The general system builder. Several presentations could be cited here but of particular interest is Ho and Loute's work on D-W decomposition using much the same approach as Aonuma and being extended by Loute to nested decomposition. This a very worthwhile software development work, even if only for experimental or comparative studies.

(4) The model implementers. Other speakers have addressed this subject, notably Knol, and we have yet to hear from Strazicky and Kalliö. It is also the area I wish to address. This category may sound presumptuous since most workers in the field would claim that they implement models. However, not all treat model implementation as a discipline in the sense intended.

THE DIFFICULTY OF IMPROVING SHEER COMPUTATIONAL PERFORMANCE

Having now set forth my view of the main emphases in the field, let me make my brash statement after all, trusting that you will apply it in the sense intended:

> Significant improvements in use of standard large-scale LP, and related modeling which depends essentially on generated LP approximations or submodels, will not result from improvements in optimizing algorithms or even directly from improved computer implementations, only slightly from improved inversion and transformation schemes, and possibly somewhat from decomposition techniques applied at a high level. This will be true for at least several years.

Let me point out some facts in defense of the above statement.

1. At EIA, they are regularly solving models of about 4500 constraints in 15-20,000 variables with well over 50,000 nonzero coefficients, from an advanced basis, more or less. The WHIZARD optimizer in MPS-III takes perhaps 5,000 interations for the first optimal which it does in less then five minutes. Do you think this can ever be significantly reduced?

2. The inversion procedure in WHIZARD takes less than .01 minutes for these bases which give no indication of instability. Do you really think you can beat that?

3. Many of us here have worked untold days, weeks and months trying to improve the simplex algorithm or make fundamental changes to it. The most we ever succeed in doing is coming close to standard commerical systems. On rare occasions when we seem to get better solution times, the real reason is found to be in special knowledge about the model.

4. Every few years someone proposes a different method. On closer investigation, these turn out to be flawed or, in at least one case I know, to approximate the efficiency of the simplex algorithm on a limited number of small test models.

But I can go further. What about matrix and report generation? The MEMM model at EIA is generated from results of about 14 upstream models and produces a large set of formal reports. In addition to files from upstream models, there is another large file of tables defining report layouts, a file of tables defining model structure, a file for initial matrix generation, another for a major revision, plus several other related inputs and outputs. Haverly's OMNI chunks through all this in four or five minutes. Do you think you can improve on that? (Note that the question is not whether MEMM could be improved.)

Decomposition has been around for over twenty years. Early bad experience with D-W algorithms led me to develop the block product form which solves the same structure with a partitioning technique. It is very close to what has been known as generalized-GUB. It was first implemented in 1967 in the LP/600 system which still exists in updated form in the current Honeywell MPS. I implemented it again in 1968 for the OPTIMA system. CDC threw the latter away, but Honeywell still claims the block product algorithm should be used for large problems with proper structure. I know of no one using it. The recent work by Etienne Loute at CORE is more promising but it is doubtful whether he beats the standard system. A really large problem with, say, ten periods with 1,000 constraints each and running on a 3033 instead of a 158 might give really impressive results. But as Jim Ho indicated, their system depends on standard MPSX/370 modules with their superstructure at a relatively high level.

George Dantzig said the other day that GUB had been highly successful. That is true only in a limited context. It was I that really implemented GUB in a commercial system and made it highly efficient; it was the original thrust of MPS-III. It did have some spectacular successes, maybe a dozen or so. (A couple it should have had were denied it due to vested interests.) These application essentially saturated the market. IBM put GUB in MPSX and dropped it from MPSX/370 because the number of users did not justify the cost. I have not encountered a real GUB problem in my own work for over five years.

Dennis Rarick built WHIZARD and he was a very clever programmer. Jim Welch now works for Ketron who took over MPS-III and he is also a very clever programmer. He has recently gone through

Rarick's code and thinks he may have made a 20-25 percent improvement. I am also a very good programmer but if Welch says he has done all he can, I would not challenge him, particularly since he now has John Tomlin to back him up on theory. I doubt if anyone here will claim they can do better.

WHERE WILL MODELING IMPROVEMENTS BE MADE?

One might conclude from the foregoing discussion that I am satisfied with current modeling practices and do not think further improvements are possible. This is not at all the case. The MEMM model referred to earlier is a terrible mess operationally in spite of the impressive executive times cited. This does not at all imply that results obtained are invalid but only that the effort expended to get them is inordinate. This should not be interpreted as a criticism of the EIA staff. They inherited models and parts from various sources and had to integrate them under conditions of extreme pressure. The point is that accepted modeling and computational practices do not permit such activities to proceed smoothly and expeditiously, and this has almost nothing to do with the basic efficiency of available optimizers and related data management systems for matrix and report generation.

There are actually two main problems involved which I will state but only discuss one. The other will have to await the outcome of work which I am just launching into.

The first difficulty has been alluded to by several speakers at this Workshop though not very succinctly or precisely. Perhaps the most meaningful words to point at the problem are "adaptability" and "flexibility", or rather their lack. The best software components are often not available separately but, even when they are, they are not very adaptable to new environments or requirements. IBM's modularizing of MPSX/370 is a step in the right direction but it does not go far enough. The large commercial MPSs, in spite of their impressive computing power and range of features, are actually not flexible. In fact, they seem to have followed the evolution of dinosaurs. I cannot go further into this subject in this discussion. It is a problem I will be addressing over the next several months. It has aspects which transcend merely the technical; for example, there are legal and proprietary impediments to a full resolution.

The second difficulty is that very few model implementers know how to use properly the tools that are available and, in some instances, strongly resist or else ignore capabilities that have been designed to help them.

I have called this the problem of symbology, which may be too small a word to convey the scope of its importance. The failure to properly symbolize things causes confusion and extra work at many points, all the way from LP model identifers (row and column "names") to data set names in a run stream at the operating system level. To make some approach to the subject, let me pose the question heading the next section.

WHAT IS A MODEL?

The term model is used in many senses, all the way from conceptualization to a particularized matrix. In reality, an LP model undergoes several stages of development and use. (The same is true of other types of mathematical models.) Like the word "file", it is impossible to get people to be precise with the use of "model". Unfortunately, "model" can be used in even more disparate senses so that different types of specialists on the same project have completely different views of what the model is.

It is possible to list the various stages of modeling and this will be done here briefly. Even so, different people will still have a different "feel" for what it is.

(a) Conceptualization: extracting from a real-world situation certain abstract relationships -- important to a desired investigation -- which are amenable to treatment by an available modeling technique. Many assumptions, simplifications and compromises are invariably necessary. Practical considerations must also be taken into account, such as availability of necessary data, software, analytic manpower, etc.

(b) Formulation: defining the variables, constraints, nature of the coefficients, limits, units (of measurement), scalings, etc. The assumptions, derivations and expected quality of results must also be stated as clearly as possible. This step involves detailed analytical work.

(c) Implementation: essentially data collection and analysis. This is often the most difficult part of the whole project and may involve a number of ancillary projects and even models.

(d) Computerization: converting to workable computer procedures the formulation and implementation and their implications. It almost invariably happen that new classes and sets of terminology are introduced in this stage, even to the point that the analysts of the preceding stages scarcely recognize their brainchild.

(e) Testing on live data: it is only at this stage that the model really begins to become "alive" and also where many defects appear. These usually lead to modifications of stages (b,c,d) until satisfactory results are obtained.

(f) Exercising the model for "real" cases. Note that experience and expertise from all the preceding stages must be brought to bear if best results are to be obtained.

In a narrower sense, a model is one particularization of the LP matrix for a case, perhaps including various alternate components or the ability to revise them for various steps in a (computer) run or coordinated set of runs. The designers and builders of application software for LP often use "model" in this sense, distinguishing this from an even more specialized form which is used for actual calculations. This terminology is also adopted by the users of such systems. Note that "users of the model" in the broad sense will include analysts who may not even be aware of such distinctions.

A CLARIFYING ANALOGY

Suppose one is going to erect a structure from a standardized architectural design. No two structures will be identical, of course, but each will have some specialization to accomodate differences in location, topography, climate, end-use, and so on. What are the major categories of materials, machines, etc., which must be taken into account? The following list is adequate for our purposes here. (We omit costs and investment schedule which need no analogy.)

1. Plans and specifications.
2. Bills of material and lists of equipment.
3. Erection schedule.
4. Specialized blueprints and instructions.
5. Preparatory and scaffolding materials.
6. Equipment for preparation and forms.
7. Structural materials.
8. Equipment for actual construction.
9. Removal of scaffolding and debris.
10. Finishing work, which depends on structural details.

Overseeing all this is a management and administrative function, actually several at various levels.

Now, it is our thesis that the use of a complex system of models to produce final results is analoguous to erecting a structure. Indeed, there are two stages: the creation of the modeling system itself, and its use for a specific case or run. For the first stage, each of the ten items above, plus management and administration can be analogized as follows.

1. Conceptualization of the modeling framework, identification of relationships to be taken into account, recognition of assumptions and limitations, formal statement of the modeling scheme (with review and professional opinion), estimates of the results obtainable ("architectural renderings"), formulation of symbology and representations, and overall flowcharts of the actual execution of model runs. All of this should exist in one or more volumes of formal documentation. Although these may seldom be referenced by experienced users of the modeling system during periods of intense activity, their contents, or course, are fundamental to the whole exercise.

2. Specifications of actual datasets which contain necessary data inputs for implementing the model, and the programs or applications systems which will process them.

3. Formulation of the run stream and other control programs necessary to carry out a run. Numerous time-dependencies and other subtleties must be taken into account.

4. Actual programming and checkout of preprocessor, generation and auxiliary programs specialized to the current class of cases.

5. There is invariably a considerable amount of utility software and auxiliary data required to carry out the overall scheme of execution. For example, temporary and scratch datasets are needed and, among other things, arrangements for their residency and life-span must be made.

6. The necessary utility programs or systems must be accurately identified are their availability assured.

7. The actual input data must be accessed at the precise time needed.

8. The necessary application systems or other software must be accessible and logically compatible with other components.

9. Temporary files must be purged. Also, most runs produce a large volume of uninteresting output which should be disposed of expeditiously.

10. Final reports must be prepared in presentable fashion, uncluttered by extraneous information. This nearly always require careful prearrangements starting at item 3 and constituting a major portion of item 4.

For the second stage -- making an actual run -- it is assumed, or course, that all the above has been done. Still, there is a not insignificant amount of planning and work for each run. (Some of the actual work may be scheduled differently, such as "prefabrication" of input variants.) We run through the ten items again for the second stage.

1. At least some thought must be given to whether the desired case is within the capabilities of the models.

2. The exact input datasets and their subsets must be specified.

3. The run stream must be specialized precisely.

4. Scenario parameters must be specified. (Here, this may come before 2).

5. Implications for temporary datasets must be taken into account.

6. Implications for control programs must be adjusted.

7. It is desirable to check beforehand that the specified input datasets really exist in a accessible state.

8. If special software is affected, the necessary module libraries must be arranged for.

9. and 10. Same as before but actually, not just planned.

It hardly seems necessary to comment on the management and administrative oversight which must go along with all this.

If the foregoing analogy is valid, a number of implications can be derived. These are the subject of the next section.

A FEW PRECEPTS

One always hesitates to be dogmatic and particularly with respect of how computing and analytical work should be done. Different people get good results with different styles and work habits. Nevertheless, when one is working within a complex system of activities, he cannot be judged on his personal results alone, but almost equally on how well they mesh with surrounding activities. Incompatibilities lead not only to extra interfacing work but also to inflexibility and the inability to trace easily the effect of changes.

One of the things largely missing in the computing field, and found in almost every other discipline, is standardization of symbology. Some de facto standardization exists, due mainly to such things as JCL which manufacturers can dictate in their basic software. (Even this is not always consistent.) In modeling work involving large arrays of values which must be identified in detail, the lack of standardization leads to incomprehensibility. This may not be the worst result; it hampers and even inhibits automated processing techniques. These latter are important to simplify summary and reporting steps, to make modifications effective at a high level of specification, and to assist analysts in detailed investigations.

Referring back to our analogy, suppose each architect used his own terminology, each draftman had his own symbols, and each supplier quoted materials in different weights, measures and packaging. The world would be a nightmare and every project an horrendous undertaking. Yet something similar occurs in the analytic use of computers.

It is not enough to organize each piece of a large project -- a form of suboptimization. There should be an overall consistency even if this imposes slightly awkward arrangements for particular parts. Everyone cannot be left free to devise his personal schemes, even if they are the best for his particular task. This principle reaches down to the lowest level of detail, perhaps particularly so. It is standardization at the lowest levels which permits flexible manipulation at high levels, not the other way around. At one time, each railroad defined its own track gauge and designed its own wheel flanges. It was much more important that these be standardized than, say, railroad management. Similar cases have occurred in computing, in a very broad context. At one time, every manufacturer claimed superiority for his recording scheme for magnetic tapes. IBM's method finally won out due to their dominance in the field. It may be that their method is not technically the best but it is much more important that today one can carry a tape all around the world and have it readable in almost any computer facility.

Whenever a required change at one level imposes changes at lower levels, difficulties are sure to ensue. (It is often cheaper to build a new structure than to remodel an old one.) Sometimes, lower level changes are unavoidable and/or desirable, but they should be made with great care, not invalidating or bypassing the original design. The ability which interactive computing techniques give us to diddle with almost any part of a system should not be used indiscriminately. In a word, some discipline should be maintained.

The above also implies that low level routines inappropriate to the task should not be used just because they are "standard system gear" and available. Sometimes whole application systems are used this way. One should not forget that substantial costs are involved in bringing all this machinery into place to do a trivial job. In general, it is advantageous to think of software and datasets as machinery and materials, and consider whether the ends justify the means. A good contractor would try to get incidental jobs done while machinery is in place for some larger purpose. Consider, for example, the size of the JCL and PCL decks required to activate an MPS, and the prescanning of the JCL and compilation of the PCL which is necessary.

A DETAILED EXAMPLE

Let me now turn from the general and analogous to the detailed and specific. During 1979 here at IIASA, I implemented a generator for a generalized regional agricultural model, or GRAM. The formal definition of GRAM, i.e., in mathematical-like notation, was the work of Professor M. Albegov and some associates. My task was to devise a computerized scheme to make GRAM a working reality. Another IIASA staff member, A. Por, also contributed heavily to this effort, particularly in devising, implementing and getting others to use a scheme for initital data specification and transformation. This was critical since we had realized from the outset that properly arranged and formatted data from a variety of government agencies in different countries would never be forthcoming without a relatively simple but flexible and easily processable format to which data originators could and would conform. This subject alone is worth 30-40 minutes of discussion which we will have to forego.

The GRAM generator was completed while I was at IIASA - although the reporting side needed further work - and was applied to a study of the Notec region of Poland. I was pleasantly surprised at the ability of our colleagues from Poland to provide ample data in appropriate format, the only hitch being the physical format of the transmittal tape which caused some intial trouble. A heavy contributor to the project was Janos Kacprzyk who deserves much credit for his dedicated and insightful efforts.

Since GRAM was being built from the ground up as an experimental system, I was able to put into effect without any conflicts some ideas that I had been trying to promote for some time past but with little success. These have to do specifically with LP naming conventions, i.e., symbology. Without claiming that my scheme is the best possible, let me present it to illustrate what such an approach can accomplish. Such ideas are not new but are seldom applied consistently. The best previous work I know of is that of Ken Palmer at Esso in London which dates back to at least 1970 and actually goes much further, for somewhat different purposes, than mine.

As we all know, LP rows and columns must have identifiers, and for practical reasons, these are limited to 8 characters. These are not properly regarded as names or even mnemonics but as encodings. The ability to use both letters and numbers is not for readibility but to extend the character sets and thus the number of usable combinations. In fact, a well-designed scheme does have considerable mnemonic quality but that is a side effect, not a goal.

The pieces, usually single but sometimes double characters, of which an identifier is composed are members of sets. These are mainly indexing sets. An LP model is essentially combinatorial in nature and this is reflected in the various combinations occurring in its identifers. But to be fully meaningful, the index sets must be assigned positions in the identifiers. Since a large model will involve more than eight index sets, considerable thought must be given to devising the most useful arrangement. This point is often handled in a very slipshod manner by model implementers which gets them into awkward messes later when they want to extract information or summarize over a set or sets.

Let us interrupt this line of thought a moment to consider what parts go into the construction of an LP model. First of all there are indexing sets or, in Haverly's terminology used in Magen and OMNI and perhaps elsewhere, classes. However, I prefer to reserve the word classes for a different purpose. Until one knows something about the categories of items to be considered, represented by index sets, there is very little one can say about a model.

Second, there are the main LP (primal structural) variables and these are usually divided into classes, say production, construction, inventory, sales, etc. Hence one needs a special index set called variable class designators to distinguish these. Further differentiation usually depends on regular index sets.

Third, there are the constraints in the variables and here four considerations come into play:

(a) The type of constraints;
(b) The constraint class, similar to variable class;
(c) The indexing which is to apply; and
(d) What data values are to be used.

Note that it is only here that numbers have been mentioned and we do not yet care what the values actually are.

Fourth, there are the data tables of which two kinds are principally involved: tables of structural coefficients, and tables of limit values (RHS, ranges, upper and lower bounds). These differ in their own indexing and naming requirements.

Fifth, and finally, are the specifications for the objective function or functions. Although depending on variable indexing and coefficient tables, these frequently do not fit neatly with the rest of the model. This was the case in the Notec model and it is the primary reason for certain special processing in generating MEMM. This is too large a subject to be further developed here but needed to be mentioned. It is a chief cause of awkwardness in generating standard MPS input files.

Returning now to index sets, one must be careful to distinguish between the name of a set, usually only one character itself, and the value of a member of the set. Thus one might have a set called I defined as I = {H, I, J, K, L}. (This is not a recursive definition.) To make this distinction, an upper case letter can be used to denote a set name and the lower case one of its members. Thus I = {i} where i is understood to run over symbolic, not numeric values even if the symbols are digits.

Since GRAM is a generalized system, further levels of abstraction are required. For example, variable classes must be specified. We do not know how many (cardinality of the variable class designator set) or their symbols (values in the set) but some such set much be specified. Since it is a master or primary set, values must be presented in two tables, one with the fixed name H:VAR.TYPE which gives their meaning against their class designator and the other with the fixed name M:VIDSTRUC which specifies their indexing structure.

The prefixes to the above table names reflect the fact that GRAM was implemented in DATAMAT, the data management extension to the SESAME interactive MPS. DATAMAT utilizes three forms of tables:

numeric tables, names prefixed G:
symbolic tables, names prefixed M:
text-string tables, names prefixed H:

They all utilize the same form of symbolic stubs and heads except H:table heads which are conventionalized for formatting purposes. Since other heads and stubs themselves constitute sets, void tables are sometimes sufficient for defining a set. In the above case, the stubs of the two tables are identical but the bodies serve different purposes and hence utilize different table forms. (Some systems, for example OMNI, permit all three forms to be combined in one table. Whether or not this constitutes a simplification is a moot point.)

For regular indexing sets, even their number and names are unknown a priori. When it is necessary to refer to the names of some indexing set whose name is unknown, an underlined upper case can be used, for example, A is the name of some set which might be A = {a}. In fact, all regular indexing sets are named in the stub of a table called H:INDICES, similar to H:VAR.TYPE. The bodies of these H:tables are used only for auto-documentation.

For each member of the stub of H:INDICES, another H:table must be specified (by that name) whose stub specifies the members of the set and whose body gives their meaning. For example, one line of H:INDICES is

```
I = 'ALL CROPS CONSIDERED'
```

which shows what index set I refers to. There is then another table H:I which has entries

```
W = 'WHEAT'
R = 'RYE'
B = 'BARLEY'
    etc.
```

which shows what members of set I refer to.

Unfortunately, every group of models has some special conditions to be taken into account. These are often expressed with "FOR" and "EXCEPT" phrases. Another technique was also used in GRAM for subsets of crops which had to be treated differently. Another table called H:CROPS has 2-character stubs, for example,

```
IG = 'GRAINS'
II = 'INDUSTRIAL CROPS'
    etc.
```

Corresponding to these stubs are void M:tables listing the subsets in their heads, for example:

```
M:IG = W, R, G
.....
M:II = B, O, L
.....
```

(the rows of dots terminate a table definition.) Such specifications are easy to devise but require that special processing code be installed in the generator. Further design effort is needed in this area and the foregoing is more an example of what not to do than of clean, general capability. Nevertheless, a few special gimmicks always seem necessary in particular cases and it is perhaps more important that the generator be easily modifiable.

Let us now see how LP column identifiers are put together. These are members of variable classes (or, for purists, are surrogates for them). The first variable class in GRAM is named X and represents growing of primary crops. The first position in an LP column identifier always represents the variable class. The following entry appears in table M:VIDSTRUC

X = XI.PRSA

The dot indicates that the third position is not used but is held by the dot. Any unused positions through the seventh are so held. The eighth was not used but left available, for example, if time periods were introduced.

The five letters after X are names of index sets and show that X-class identifiers are constructed by running over all combinations of members of these sets, subject to the provision that a nonzero coefficient appear in some constraint for each combination. An assembly language subroutine is used to run over all combinations, like a mixed radix counter. (An analogy is a digital clock showing days, hours, minutes and seconds.)

Another entry in M:VIDSTRUC is

U = UJKPRT.

No conflicts arise since sets I and J are never used in the same variables, and similarly for S and T. The placement of the P and R sets was dictated by the fact that they appear in all variable classes and most constraints classes. Only the fourth and fifth positions left enough positions on each side for orderly assignments. This kind of pre-analysis <u>must</u> be made before specifying identifier structure.

A different sort of conflict did arise with Y-variables which have the same structure as X-variables but which involve a subset of the I set in the same constraints as X-variables. This was resolved by defining a Y <u>set</u> and installing special processing code. This was the most awkward situation in GRAM and resulted from a formulation compromise with respect to secondary crops which are not adequately handled by LP. Again, some special gimmick always seems to arise.

Specification of constraint (i.e., LP row) identifiers is different from that of column identifiers and inherently more difficult. One may note the following differences immediately:

(1) There are generally more index combinations for columns but fewer classes. As a result, column identifiers are more straightforward though the number generated or examined may be very large.

(2) Index sets disappear from those constraints in which they are summed over. The fewer index sets appearing, the fewer constraints but the total number of coefficients may be about the same due to implied aggregations. For example, a constraint over a total region may include all the coefficients of contraints over its subregions.

(3) Constraint classes and index sets may not be sufficient to insure nonambiguity, due to multiple or special constraints over the same items. Conversely, index sets may appear which do not occur for columns.

(4) It is useful to assign one position (the first) to designate LP constraint type, independently of constraint class vis-a-vis the model. (This may help alleviate the problem of ambiguity but does not guarantee to eliminate it.)

(5) Constraint indexing must match limit table indexing and be consistent with coefficient table indexing. Universal set members "any" and "none" may also be necessary.

All these situations occurred in or were imposed on GRAM and it must be admitted that further work is needed in generalizing constraint specification. Nevertheless, the scheme used proved quite workable and its main points will be indicated here.

The first position of a constraint identifier is assigned to LP type. These are listed in a table H:CON.TYPE which is general to LP formulation (or intended to be) and not specific to a class or models. These types include simple upper bounds and GUB sets although, in fact, these features were not used as such. The main types were as follows:

A = 'AVAILABILITY, NO MIN REQUIREMENT'
B = 'BOUNDED ABOVE AND BELOW (RANGED)'
C = 'INEQUALITY CONDITN, NO CONSTANT'
D = 'DEMAND, NO UPPER LIMIT'
E = 'EQUALITY, GENERAL'
F = 'FUNCTIONAL FORM'
K = 'GUB INEQUALITY, (NO ACTUAL GUB)'
L = 'BALANCE EQUALITY, NO CONSTANT'

The second position specifies constraint class and is specific to the set of models. These classes are listed in H:CONCLASS and included, for example:

```
B = 'LABOR'
C = 'CAPITAL'
D = 'WATER'
   ...
L = 'LAND'
   ...
W = 'WAGES'
$ = 'COST OR PROFIT'
```

The last was used for functional definitions for which a special technique was used which will not be further discussed.

The actual constraint identifier structures appear only in the stub of a master table called M:CON which is, in fact, an abbreviated definition or "picture" of the entire model. Table M:CON is the primiary driver for the generator and ties together all the various parts, showing their relationships.

One identifier appearing in the M:CON stub is

```
BFFPR.
```

which indicates a set of double inequalities on fertilizer, indexed over set F (fertilizer types, not appearing in column identifiers), P (type of economy) and R (subregion), where P is restricted to values '2' and '3' (cooperative and private). The variable classes involved are X, Y and U which include index sets I, P, R, A, J, K and T (also the Y subset of I). Hence summation is over I (and Y), S, A, J, K and T for each FPR combination. This was the largest set of constraints in the model.

The head of M:CON and the row for BFFPR. are shown below.

```
M:CON = MIN, RHS, SUM, MSUM, ISET, PSET, NO.
                   - - -
BFFPR. = GFPRN, GFPR, AXAY, FU, ALL, '23', '23'
```

This is read follows. Lower limits (MIN) for these constraints are given in table G:GFPRN. Upper limits (RHS) are in table G:GFPR. Variable classes X and Y both appear, with coefficients from G:A taken positively (SUM). Variable class U appears with coefficients from G:F taken negatively (MSUM). All members of set I are used but only '2 and '3' from set P. This is constraint set number 23. (The I and P sets are specialized in several constraints and hence have columns in M:CON.)

Another set of constraints has the identifer BFF... and hence only one constraint for each fertilizer type. Summation is over P and R sets in addition to those previously noted. All the same coefficients occur but the MIN and RHS tables are smaller and different.

Since the GRAM generator is written in DATAMAT, it is easily modified. However, the DATAMAT processor is not very efficient for this kind of application on models as dense as that for the Notec region. (It had approximately 50,000 nonzero values.)
Essentially, DATAMAT was used like a compiler for which it was not designed. Nevertheless, once generated, the model was easily worked with and the general approach seems eminently suitable for automated model generation.

The important point we wish to make, however, is that careful design and control of symbology is a sine qua non for well-managed modeling systems. Much the same approaches could be used with other systems such as DATAFORM (very close to DATAMAT in language) or OMNI which has direct definition of sets (classes). Both the latter utilize compilers and are designed for batch processing whereas SESAME is an interactive system and DATAMAT is largely interpretive. (A special version of DATAMAT was implemented for GRAM which uses a kind of "half-compiler" and this greatly improved throughput.) The important task seems to be to convince LP modelers to use existing tools effectively and to demand further improvements in the future, rather than forever falling back on ad hoc FORTRAN programs and sloppy nomenclature.

APPLICATIONS AND MULTICRITERIA OPTIMIZATION

A DECOMPOSITION–COORDINATION APPROACH TO LARGE-SCALE LINEAR PROGRAMMING MODELS: AN ASPECT OF APPLICATIONS OF AONUMA'S DECOMPOSITION METHOD

Tatsuo Aonuma

Kobe University of Commerce

This paper examines a practical aspect of our decomposition method for dual angular linear programs[5] in applying it to a planning system as a means of coordination. We often recognize in real planning systems that a "feasible solution" to a large-scale planning problem is obtained by solving a sequence of subproblems and composing their solutions rather than formulating it as a single large-scale model to be optimized. In the decomposed models, the planning staff can control the solutions through the modification of the succeeding models so as to be more desirable with respect to their criterion. We consider how to improve the present planning system leaving its basic planning method unchanged. In particular, we tacitly assume that we shall be interested in considering a manner of computerization of the planning system from a practical viewpoint. We emphasize the need for a practical coordination method in order to improve the plans obtained separately from the submodels toward an overall objective. It is supposed that most of the models for such coordination will actually be formulated in dual angular form involving the submodels as part of their entire structure. Based on computational experience with our experimental codes, MULPS/FORTRAN[7] and MULPS/APL[6], we shall present one direction of application of our algorithm to a coordination method.

1. Introduction

We shall examine a practical aspect of our decomposition method for dual angular linear programs[5] in applying it to a planning system as a means of coordination. We oftan recognize in real planning systems that a "feasible solution" to a large-scale planning problem is obtained by solving a sequence of subproblems and composing their solutions rather than formulating it as a single large-scale model to be optimized. In such cases, there seems to be persuasive and legitimate reasons why such a planning manner has been adopted in the real systems. The reasons are closely related to the existence of intangible factors under real planning circumstances and the subjectivity of the planning staff's criterion to evaluate the plans. It may be too difficult to formulate those in a form of a single model. In the decomposed models, the planning staff can control the solutions through the modification of the succeeding models so as to be more desirable with respect to their criterion.

We shall consider how to improve the present planning system leaving its basic manner of planning unchanged. In particular, we tacitly assume that we shall be interested in considering a manner of computerization of the planning system from a practical viewpoint. For that purpose, we shall emphasize the need for a practical coordination method in order to improve the plans obtained separately from the submodels toward an overall objective, even if it is a minor alteration. It is supposed that most of the models for such coordination will be actually formulated as a dual angular type of model involving those submodels in its entire structure.

Based on our computational experience in using our experimental codes, MULPS/FORTRAN[7] and MULPS/APL[6], we shall present one direction of application of our algorithm to a coordination method, in which the algorithm may not be necessarily used for performing a global optimization of large-scale linear programs. Rather, we intend to replace the "experience-based" coordination in an actual planning system by a practical decomposition-coordination method when we computerize the planning system.

In Section 2 a decomposition-coordination method will be described as one of the future directions of large-scale linear programming research. In Section 3 the features of our algorithm and the relations with other algorithms will be mentioned, and the conclusions from the computational experience will be given. In the last section we shall introduce an example of a hierarchical decomposition approach to real production scheduling in a Japanese oil company, which will be regarded as the first step toward an application of our decomposition-coordination method.

2. The Need for a Decomposition-Coordination Method

It is well-known that hierarchical methods have been adopted in large-scale industrial planning systems for practical planning [14]. By introducing a hierarchical structure into a planning system, we can formulate a vague overall planning problem as a number of more concrete subproblems on various kinds of hierarchical levels ; hereafter we will refer to the terminology of hierarchical system theory of Mesarovic et al. [14].

For example, the planning system in a Japanese oil company,which is reputed to be one of the most intensive users of linear programming, has the following hierarchical structure basically :

Level 1 (Highest) : Long-term planning ; facility and investment planning, curdes selection and contracts, etc.

Level 2 : Short-term planning for six months or one year ; profit and loss planning, supply-demand planning,etc.

Level 3 : Production planning for every one month in the term.

Level 4 (Lowest) : Operational planning and scheduling for one month.

All planning activities except on Level 4 are performed in the head office, but those on Level 4 are performed at the refinery-sites.

We may regard this hierarchical planning system as a practical type of decomposition method to solve a large vague planning problem so that a large abstract model is decomposed into a number of concrete submodels and each is sequentially solved in a given hierarchical order. We should notice that, as a simple practical way to obtain an approximate solution to a large-scale planning problem on a hierarchical level, a similar approach is also used, on which we shall focus our attention hereafter. In particular, this simple way has been much more favourable in planning on lower levels than on higher levels, because there are many time-dependent activities having different priorities and uncertainty at the operational fields and these make the model too big. This type of hierarchical method to solve large-scale problems is called the *hierarchical decomposition method* [1],[2],[3],[12],[13].

Among the submodels created by the hierarchical decomposition method, there are two kinds of linkages, one between models on the different levels and another between those on the same level. In the hierarchical decomposition, those linkages are regarded as the boundary conditions of the subproblems, and each problem is solved in a given order for assumed values to those boundaries. An actual example of hierarchical decomposition will be described in Section 4.

One of the weak points in the hierarchical decomposition method is that it lacks an explicit manner to coordinate the solutions to the submodels toward an overall objective. Such coordination may mostly be performed informally by means of information exchange among the planners or between units in the organization. A set of the solutions to the submodels by this approach may be no more than a "feasible solution" to the entire planning problem. If a practical coordination method could be realized in which the assumed values given for the linkages could be adjusted so as to improve the overall performance, we will be able to design an efficient computerized planning system which could set up a better plan with only a smaller amount of effort than in the present system. We shall call the hierarchical decomposition approach having an explicit coordination process the *decomposition-coordination method*.

Much effort has been made so far for developing mathematical programming models of practical use, in such industries of Japan as oil and steel, on higher levels of planning. On the other hand,the models and the solution methods for planning and scheduling problems on the lower(operational) levels have been left aside except for the optimal control problems of physical processes. Rather, the skill of experienced staff has been regarded as being important. However, this tendency will be changing in the near future to a more computerized logical manner than presently because of the appearence of high-level small computers and the remarkable progress in management decision support systems. The personal computers will become popular and will be used at the operational fields for generating data for models and analyzing the results, linking it with the main computer very soon.

It seems that the following two points on large-scale mathematical programming will be important in the near future for computerization of the planning systems :

1) *On the higher levels of planning systems,* a coordination method and its related models useful for linking together the various kinds of planning models on two different hierarchical levels and for coordinating them toward an overall objective.

2) *On the lowest level,* a practical approach for solving large-scale problems on the same hierarchical level, in particular, scheduling problems : for example, most production scheduling models are too large to be directly solved by a computer installed at the operational fields. To install a large computer only for its purpose will not be economical. Therefore, the hierarchical decomposition method is often adopted for solving them. Any practical coordination method will be also very useful even for solving a large-scale operational problem so that the method may make it possible to improve the present solutions to some extent.

Two Types of Algorithms Required.

We shall now emphasize that two types of algorithms should be developed for large-scale linear programms for the purpose of realizing the above mentioned points.

1) *Algorithm as an Efficient Software :* This area is in line with the traditional direction of developing new algorithms, which are to solve the large-scale problems directly and efficiently. This may be regarded as finding another "simplex method" for large-scale problems.

2) *Algorithm as a Coordination Method :* A certain type of algorithm useful for computerizing a real planning process for obtaining an improved solution will be needed. Firstly, this algorithm should have a mechanism adapted for expressing at least two typical operations of decomposition and coordination which often exist in real planning processes for large-scale problems. It will be of a two-level scheme which means both solving subproblems in sequence and coordinating them for an improved solution. Secondly, this algorithm should have a certain type of algorithmic structure so as to be able to be easily realized as a user's own mathematical programming system in a simple way; it will be desirable for this purpose that the system can be easily written by the users themselves as

a "Mathematical Programming Systems Complex" built up by using advanced commercial codes.

There is some coordination in the real hierarchical decomposition approach. It may be performed on the basis of the planning staff's experience, which we shall call the *experience-based coordination*. It has been shown in papers on organizational theory, e.g.,[8],[9], that the number of information exchanges between the units in an organization for coordination is only a few, as it is limited mainly by cost and time. We shall need a coordination mechanism, in the algorithm, which is close to a real coordination manner.

In the case of computerization of a real planning system, we shall need to write our own computer program involving a matrix generation from maintained data bases, modification of input data and various submodels, the facilities for checking the overall acceptability of the results, a repeated optimization of subproblems, and the composition of the solutions to the various subproblems. We shall call it the *Mathematical Programming Systems Complex* (MPS Complex). For example, the Extended Control Language (ECL) of the MPSX/370 has the facilities appropriate for this purpose, by which we can write a MPS Complex in such a manner that the powerful MPSX can be used as a tool for the optimization of the various problems in the MPS Complex. The development of this type of algorithm seems to bring us a wider class of advanced MPS applications in the future.

3. Our Decomposition Algorithm as a Coordination Method

An original form of our decomposition algorithm [3] was developed as a coordination method for solving two-stage linear programs in the nested two-level approach to multi-period planning, called the PAIROP system[2], and then the idea was further extended to the present algorithm[5] for solving dual angular linear programs. Therefore, the algorithm has the features convenient for using it as a coordination method. In addition, we have observed from our computational experience [4],[5],[7] that the computational behaviour is also desirable to a decomposition-coordination approach. The features of our algorithm are summarized as follows :

The features of the algorithm.

1) The algorithm is of a resource-directive decomposition and can make use of "good" initial values for linking variables easily, such as those obtained by

the planner from experience. The other advantageous properties inherent in the resource-directive decomposition such as claimed in Burton et al.[8] are also found.

2) A number of linear programs are solved throughout the entire algorithm. An efficient computer program can be easily written by users as long as an advanced linear programming subroutine is available.

3) The optimizing strategy in the present algorithm is easily modified for the purpose of taking account of a planner's implicit utility function defined on the objective functions of the subproblems, since the structure of the subproblems is preserved throughout the optimization[3].

4) The number of coordination cycles required for optimality is relatively small and does not necessarily increase along with an increase in the number of subproblems. Rather, it stays at a relatively small number. We may roughly estimate it at the number of subproblems or even less, though it usually depends on the initial values for the linking variables.

5) As the solution attained at the first or earlier cycles of coordination is close to the optimum point, it may be effective to stop computing after a few cycles before the exact optimality is obtained, as it is in the experience-based coordination.

The relations with other algorithms.

Lately, we have noticed that our algorithm is basically along the same line as Gass' algorithm[11], and, therefore, is expressed in terms of Winkler's GBBF simplex method[17] specified by a special solution strategy. However, the basic idea underlying Winkler's algorithm can be regarded as a simplex method based on a basis factorization method, i.e., his algorithm belongs to the first type of algorithm mentioned in Section 2.

The main differences between Gass' algorithm and ours are summarized as follows:

1) In order to find an improved basis for a non-optimal subproblem, we employ a direction-finding problem, which is defined as a small linear program. We have already reported in [3], although it is in the case of two-stage linear programs, that the CPU time and the number of coordination cycles required for optimality in our method are less than those when employing Gass' type of selection rule to find a new basis. In our method more than one basic variables at a time are exchanged by solving the direction-finding problem. On the other hand, only one basic variable in the non-optimal subproblem is exchanged in Gass'

algorithm. This difference makes the number of coordination cycles smaller in our method.

2) In our algorithm the non-optimal subproblem is solved in a complete form only if the direction-finding problem can not be defined or it can not bring us a new basis. Gass' algorithm always solves it in a complete form.

3) In the coordination problem of our algorithm, free variables are defined at every cycle in order to improve the present values of the linking variables. The values of the linking variables are adjusted around the present values through the free variables. On the other hand, the original non-negative linking variables are used for this purpose in Gass' algorithm, because there is no concept of coordination in his algorithm. This implies that the coordination problem in Gass' algorithm is defined from the original linking matrices. In our algorithm it is defined from the linking matrix updated with respect to the present basis matrices of the subproblems.

The conclusions from the computational experience.

According to our computational experience in using two experimental codes, MULPS/FORTRAN for a medium-size computer[7] and MULPS/APL for a minicomputer[6], we can conclude the following:

1) The size of the coordination problems often restricts the use of our algorithm. The number of rows in the coordination problem is equal to that of the linking variables,and the number of columns is larger than the total number of rows in the entire problem. Both of these numbers are very large in real problems. Therefore, we must develop an efficient scheme to deal with as large a coordination problem as possible.

2) The coordination problem itself has quite often a special structure in case of real problems. We shall propose that commercial MPS packages to be developed in the future should have only a standard LP subroutine but also various ones for structured linear programs. Such an MPS could make it possible to bring us more advanced MPS applications than the present ECL does, in the case of the computerization of planning systems. For example, the coordination problem for dynamic models has a staircase structure. If we could employ an efficient algorithm[16] to make use of it, it would be possible to solve much larger problems more efficiently than we do now. Such an attempt is under way.

3) For the optimization of subproblems in sequence, at the first stage of the MULPS, we had better make full use of the optimal basis already obtained in the past sequence of optimization of the subproblems as a starting basis for the

subsequent subproblems to be optimized. We have found that the computing time is reduced considerably, if we use the optimal basis already obtained to the other subproblems. This is because the subproblems in most actual problems are very similar to each other in structure. To take our 3-stage dynamic planning model from a real oil refinery, the time needed for computing the second and third subproblems are nearly as half as for the first subproblem, when the optimal basis to the first is used as the starting basis for the other two.

4) The most time-consuming job in the MULPS computation is that of setting up and solving the coordination problem[6]. This time greatly depends upon the skill of file management as well as the performance of the linear programming subroutine adopted. We should consider developing efficient methods for file management appropriate for designing a MPS Complex.

4. An Example of a Hierarchical Decomposition Approach

Let us introduce an example of the hierarchical decomposition approach in production-scheduling based on the same idea as in [2], which is now working very successfully in a Japanese oil company [15]. The planning and scheduling problem is on the lowest(operational) level in the hierarchical planning system mentioned in Section 2. The refinery makes a production schedule for the next month at the end of every month on the basis of a production plan given by the head office. The production-scheduling problem may be formulated as three linear programs which respectively correspond to a production schedule for every 10 days and those may be linked by linking variables representing inventory-levels at the end of a period of 10 days. The length of a period has been decided according to the operational experience.

Each submodel has the size of about 200 x 600, where there are nearly forty different kinds of semi-products and thirty kinds of products to be blended. The hierarchicaldecomposition approach to this problem is described as follows: First, a crude-charge-schedule is obtained by the planning staff. Then, on the basis of the schedule, those subproblems are sequentially optimized in such a hierarchical manner as shown in Fig.1. After the three subproblems are solved, then the "experience-based coordination" starts. The planning staff checks the overall acceptability of the solutions, e.g., their feasibility and performance, on the basis of their operational experience. The crude-charge-schedule and the three blending problems are modified so that an improved schedule and blending plan may be obtained, and then the modified problems are reoptimized. Satisfactory solutions are mostly obtained after the first coordination, i.e., the modification of problems is usually performed only once.

The reasons why the problem is solved by the hierarchical decomposition method may be summarized as follows :

1) The blending operations obtained by this approach are more acceptable than those obtained when solved simultaneously, since the more accurate figures of properties of the blending stocks than the assumed ones before obtaining them can be recalculated before solving the problem for the next period, which may considerably vary from their values assumed at the beginning, depending upon the operations adopted for the previous periods.

2) The suboptimization is more time-saving for the purpose of obtaining a practical schedule than when the entire 3-stage model is solved at the same time by the computer at the operational field. The modification of the problems and the reoptimization of them will be more time-consuming and complicated for the entire model than for the decomposed model. In particular, the remarkable merit of the decomposition method is to be able to use a good starting basis for the second and third problems, which is derived from the optimal basis of the previous problem. For example, the computing time for the succeeding problems used to be less than a half of that for the first problem by this rule.

3) It is necessary to avoid a strong effect on the schedule for the first half of the month from the operations for the latter half, because usually there is more uncertainty in the data for the latter half than for the first half. It seems that the planning staff uses its own implicit utility function on the three objective functions of the subproblems for the purpose of considering the uncertainty in the planning process. The experienced staff's subjective judgement is regarded as very important in real situations.

The Decomposition-Coordination Approach.

An attempt to apply the decomposition-coordination approach to the present system is now being made with the intention of computerizing the experience-based coordination as much as possible. The system will be written as a MPS Complex based on our decomposition method by using the Extended Control Language of MPSX/370. In particular, the following features of the system will be emphasized :

1) The subproblems are sequentially generated and optimized from the first period to the third period. The system has the facilities to determine the initial values of the linking variables and to generate a part of a matrix of the succeeding problem from the solution to the previous problem.

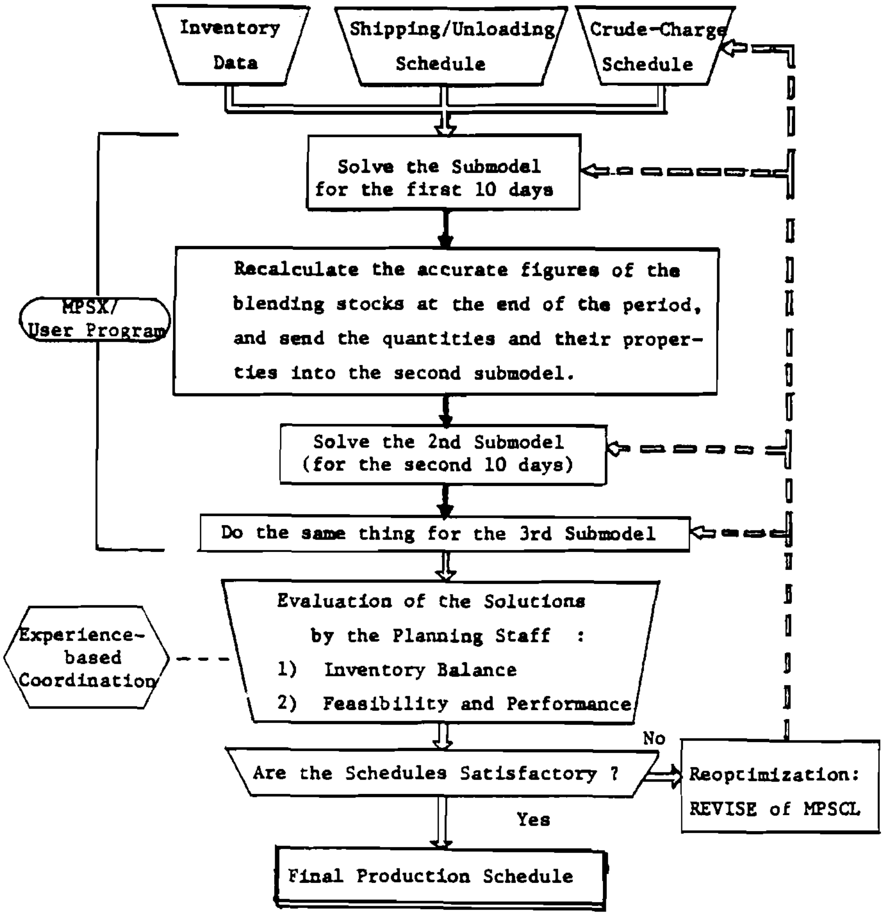
2) The system has the facilities to generate the coordination problem which is partly based on the planning staff's judgement.

3) The optimization of the coordination problem is performed in such an interactive manner that the planning staff can measure, by itself, its implicit utility function on the three objective functions of the subproblems [3].

Acknowledgment

The author wishes to thank Mr. Toshio Nishi, The Mitsubishi Oil Co., for his suggestions and the discussion with him concerning the application of the algorithm to the real production and scheduling problems, and also the staff of the Section of Operations Research of the company for its cooperation to realize his idea of the planning method.

Fig. 1 Hierarchical Decomposition Approach in Production Scheduling


Inventory Data
Shipping/Unloading Schedule
Crude-Charge Schedule
Solve the Submodel for the first 10 days
Recalculate the accurate figures of the blending stocks at the end of the period, and send the quantities and their properties into the second submodel.
MPSX/ User Program
Solve the 2nd Submodel (for the second 10 days)
Do the same thing for the 3rd Submodel
Experience-based Coordination
Evaluation of the Solutions by the Planning Staff :
1) Inventory Balance
2) Feasibility and Performance
No
Are the Schedules Satisfactory ?
Reoptimization: REVISE of MPSCL
Yes
Final Production Schedule

References

[1] Aonuma, T., "Hierarchical Decomposition in Dynamic Linear Models", WP-31, Kobe University of Commerce (Kobe, June 1976).

[2] ___"___ , "A Nested Two-level Approach to Multi-period Linear Programs" (in Japanese), *Abstract at the Annual Meeting of the Operations Research Society of Japan in 1976*, pp.75-76. Also, *Jour. of Kobe Univ. of Commerce*, 29,6,345-358(1978).

[3] ___"___ , "A Two-level Algorithm for Two-stage Linear Programms", *Jour. of Opns. Res. Soc. of Japan*, 21, 2, 171-187(1978).

[4] ___"___ , "Computational Experience in Using a Decomposition Algorithm(MULPS) for Multi-period Dynamic Linear Programs", WP-44, Kobe Univ. of Commerce (Kobe, July 1978).

[5] ___"___, "A Resource-directive Decomposition Algorithm for Weakly Coupled Dynamic Linear Programs", presented at the XXIV International Meeting of TIMS, June 18-22, 1979, Hawaii.

[6] ___"___, "MULPS Linear Programming System/IBM 5110 APL - A Decomposition Method for Dual Angular Linear Programs by a Minicomputer - ", WP-52, Kobe Univ. of Commerce (Kobe, April 1980).

[7] Aonuma, T., M.Morita, and H.Nishimoto, *Users' Manual of MULPS/HITAC 8250* (in Japanese), Research Report No.13, Kobe Univ. of Commerce (Kobe, December 1978).

[8] Burton,B.M., W.W.Damon, and D.W.Loughridge, "The Economics of Decomposition : Resource Allocation vs Transfer Pricing", *Decision Sciences* 5,3, 297-310(1974).

[9] Christen,J., and B.Obel, "Simulation of Decentralized Planning in Two Danish Organization Using Linear Programming Decomposition", *Mgmt.Sci.*, 24,15,1658-1667(1978).

[10] Freeland,J.R.,and J.H.Moore, "Implications of Resource Directive Allocation Models for Organizational Design", *Mgmt.Sci.*,23, 10,1050-1059(1977).

[11] Gass, S.I., "The Dualplex Method for Large-scale Linear Programs", ORC66-15, Operations Research Center, Univ. of California (Berkeley, June 1966).

[12] Goreux, L.M., and A.S. Manne (eds.), *Multi-level Planning : Case Studies in Mexico,* North-Holland, Amsterdam, 1973.

[13] Kidland, F., "Hierarchical Decomposition in Linear Economic Models", *Mgmt. Sci.,* 21, 1029-1039(1975).

[14] Mesarovic,M.D.,D.Macko, and Y.Takahara, *Theory of Hierarchical Multilevel Systems,* Academic Press, New York, 1970.

[15] Nishi, T., "Monthly Production Scheduling Program in Kawasaki Refinery", private communication.

[16] Perold, A.F., and G.B.Dantzig, "A Basis Factorization Method for Block Triangular Linear Programs", Technical Report SOL78-7, Systems Optimization Laboratory, Stanford University (Stanford, April 1978).

[17] Winkler, C., "Basis Factorization for Block-Angular Linear Programs : Unified Theory of Partitioning and Decomposition Using the Simplex Method", RR-74-22, IIASA (Schloss Laxenburg, November 1974).

[18] Slate, L., and K.Spielberg, "The Extended Control Language of MPSX/370 and Possible Applications", IBM System Jour., 17, 64-81(1978).

OPTIMAL DAILY SCHEDULING OF ELECTRICITY PRODUCTION IN HUNGARY*

I. Deák,† J. Hoffer,†† J. Mayer,†† A. Németh,†† B. Potecz,†† A. Prékopa,† and B. Strazicky†

†Computer and Automation Institute, Hungarian Academy of Sciences, Budapest
††Hungarian Electric Energy Industry

Given:

- a piecewise constant function approximating the country's total demand on electrical energy in 27 time-periods spanning 25 hours forward,
- the set of applicable production technologies for each power station in the country, and
- the actual network of power lines,

we must determine the method of production to be applied and its capacity level in each power station in each period of the day, so that the cost of production should be minimal, the demand should be met in each period, and the capacity and network constraints should be fulfilled.

The model of this problem is a large, but structured, mixed 0—1 programming problem, with at most 5 coupling constraints and a very special connection between the 0—1 variables. It is solved on a CDC 3300 computer. The method of solution is heuristic, involving Benders' decomposition method for subproblems.

*The problem treated in this paper has been modelled under the directorship of Prof. A. Prékopa in the O.R. Department of the Computer and Automation Institute of the H.A.S. jointly with specialists of the Hungarian Electric Energy Industry.

1. INTRODUCTION

At the Operations Research Department of the Computer and Automation Institute of the Hungarian Academy of Sciences there has been for several years a work in progress together with the experts of the Hungarian Electricity Boards Trust to apply operations research in the electricity power industry. In the course of this work the model and computer program system to be described in this paper /which can be considered as a case study/ has been completed. Starting from the verbal statement of the problem we have arrived, through a large number of steps at the solution of the real problem with real data. These steps are: clarification of every detail of the physical problem, adequate mathematical modelling of the problem, building up the data system required for the mathematical model, preparation of a program system, using the permanent data base, suitable for producing the numerical data of the actual problem to be solved. In the course of the modelling, a kind of problem formulation, describing the reality well enough had to be found, enabling at the same time the problem to be handled computationally. The completed model leads to a large-scale mixed variable linear programming problem where the integer variables are of 0-1 type. A method had to be worked out on the CDC 33oo computer that gives a nearly optimal solution to the problem in an acceptable time. The computer program system was required to present the results in the form prescribed by the user.

Caracteristic for the entire work has been the constant co-operation among the experts of the two institutes resulting in a permanent corrective activity in the subsequent stages.

2. FORMULATION OF PHYSICAL PROBLEM

2.1. The overall electric power demand of the country as considered for each day separately as a function of the time is illustrated on Fig.1. where the shape of the curve is characteristic. The time corresponding to the initial point of the curve is the so-called evening peak load time. This is followed by a time interval with decreasing load, thereafter by some hours when the value of the demand differs from the minimum value to a little extent only, thereafter a stage with increasing load - and the whole is repeated once more. The shape of the curve is in every case of this type, but the lenght of the intervals as well as the demand values change daily.

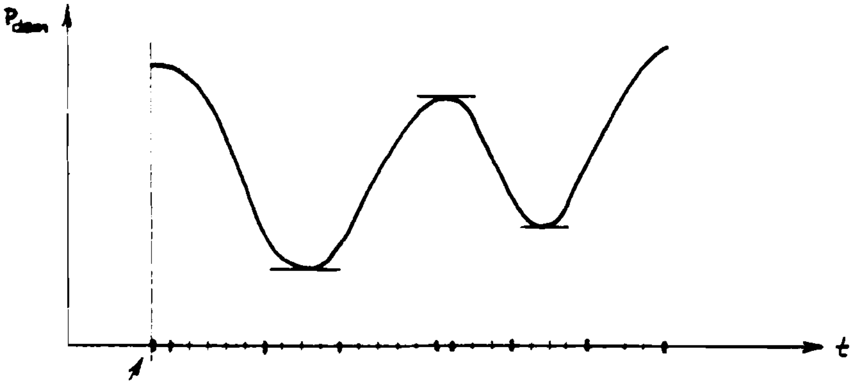


Fig.1.
A typical daily electric power demand function

The electric power demand of each day can be forecasted in advance with an accuracy of 1-2 % on the basis of the data available on the day before. We investigate always the 25 hours period following the evening peak, this is subdiveded into 23 one hour and 4 half-hour periods in which periods the demand can be assumed constant. The demand contains the estimated values of the power plant's own consumption and of the network losses.

2.2. The electric power demand is staisfied by the electric power generated in the country's power plants and from the neighbouring countries imported power. In our country there are about 2o such power plants that are considered in the model. The electric power imported from abroad in international co-operation is considered as one power plant with constant production.

In the power plants the power is generated by the combined operation of various aggregates in different modes of operation. Each mode of operation involves the combined work of certain aggregates. The applicable modes of operation and the physical quantities characterizing them are given for each power plant.

The given mode of operation of a power plant can run within given power limits and the production cost, as a function of the power level, is a function illustrated on Fig.2. This can fairly well be approximated by a piecewise linear function (Fig.3.) where for the slopes the relations

$$c_1 < c_2 \quad \dots < c_k$$

always hold.

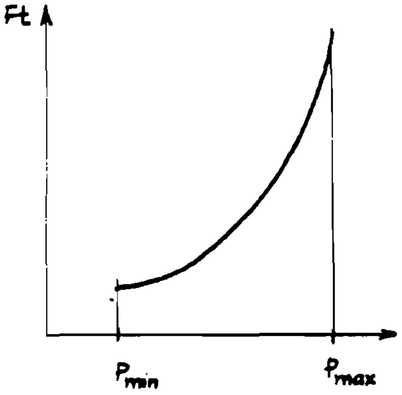


Fig.2.

Production cost function

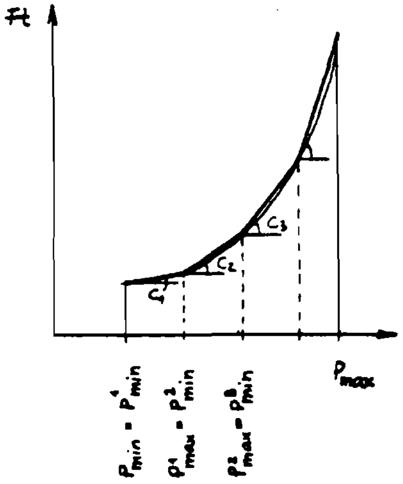


Fig.3.

Piecewise linear approximation of the production cost function

The change-over among modes of operation - start or shut off at least one of the generators - causes the turn of a mode of operation. Thus the change-over is not allowed among all possible modes of operation of a power plant, viz. not among those working with entirely different devices. An accidental failure or maintenance of the equipment can result in the daily change of the modes of operation in the power plant. Fig.4. shows an example of the modes of operation, and in Fig.5. we can see the function of still stand cost.

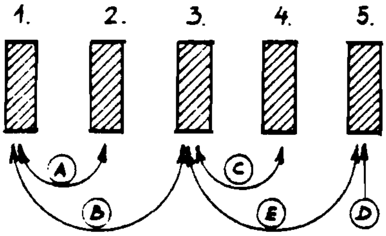


Fig.4.
An example for the definition of the modes of operation

1-2-3-4-5 denote generators,
A-B-C-D-E are possible modes of operations, where the arrows indicate the generators that work in the given mode of operation.
A direct change for example between the modes C and D is not allowed, but from C to E /it is a start of generator 5/ and from E to D a direct change is possible /shut off of generator 3./.

The electric network of the country is a set of nodes and branches. Its nodes are either power plants or points in which the power demands occur, and its branches power transmission lines and transformers with given physical characteristics. Some of the network's nodes can be connected to power stations and from almost all the consumer's demands are supplied. Also the electrical network can /and does/ daily change on account of maintenance, failure etc. Change means here that certain branches or nodes do not belong to the system on a given day, or the value of their physical characteristics differ from those in case of normal operation.

2.3. With this knowledge our task is to determine for each period of the following 25 hour duration the modes of operation to be applied in the different power plants and their production levels so that the power demand should be satisfied in each period, the physical restrictions on the actual network hold, moreover the so-called fuel constraints be satisfied with a minimum power production cost. The fuel constraints require that in some power plants the value of the daily overall production - directly connected with fuel consumption - should differ from a given value only to the extent of a given very small percentage. The reason of this restriction can be that we cannot consume more than the existing amount of fuel or that certain amount of fuel is expected to arrive on the next day and the storage capacity is limited.

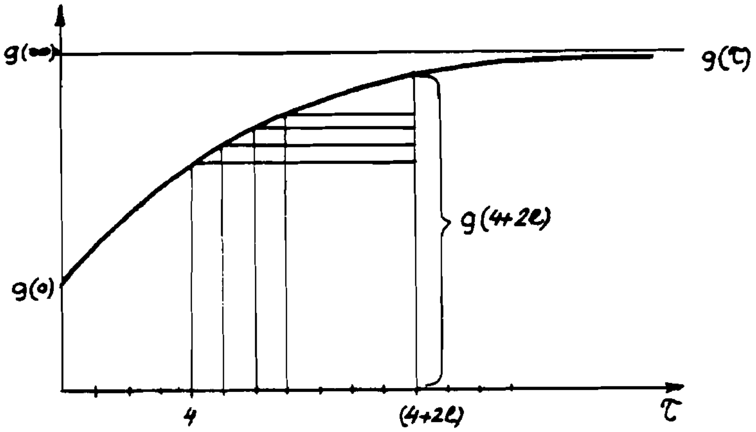


Fig.5.

Stillstand cost function

The power production cost contains the actual production cost, the change-over cost resulting from the switching of modes of operation resp. standstill and restart of the machines, as well as the cost of loss of power in the network.

3. ASSUMPTIONS

Because of the sophisticated nature of the whole power system to be optimized we had to make some assumptions /simplifications/ in order to obtain a model that can be handled.

3.1. By knowing the shape of the demand function we agree that in the first periods when the value of the demand does not increase we allow only such a change of the mode of operation which can be realized by shutting off a generator or generator groups. These periods together are called stop or shut off phases. No change in the mode of operation is allowed in the altogether 4 periods around the period with minimum demand /phase of stagnation/; only the production level of the given mode of operation can be changed. In periods of increasing demand only such change of mode of operation is allowed where at least one of the generators is turned on /start periods/. The investigated phases are therefore: stop, stagnation, start and once more stop, stagnation and start phases.

In connection with this we agree that at every plant we assigne subscripts /integers/ to every mode of operation starting from 1 and going up to the number of possible modes of operation at the given plant. We do it in such a way that whenever the transition from mode $j \to k$ $(j < k)$ is possible then from mode j to mode k we arrive by shutting off at least one generator. Note that a transition $j \to k$ is not always possible.

3.2. As a result of physical considerations we have agreed to prescribe the requirements limiting the physical state of the electric network only in the three periods with extreme demands /the first period, the first period of the first stagnation phase and the last period of the first start phase; these will be referred to as voltage check periods/. That is, we assume that if in these periods the physical restrictions of the network are satisfied, then in periods of "intermediate" demand with the application of "intermediate" modes of operation /cf. assumption 3.1./ the physical restrictions are also satisfied.

3.3. In order to determine the cost of power production the following simplification will be made.

a./ The cost functions of the particular modes of operation will be approximated by piecewise linear functions.

b./ Symmetric restarting will be assumed for the calculation of the still stand cost arising from the change of modes of operation. This means that if we shut off a generator at ℓ periods before the first period of the stagnation phase, then the restart takes place at ℓ periods after the last period of the stagnation phase, that is the still stand lasts $4+2\ell$ periods. The difference between the actual still stand cost and the approximate value of it will be neglected. The total cost in the $4+2\ell$ periods is subdivided into $4+2\ell$ parts and are assigned to these periods.

c./ The cost arising from the network loss will be calculated from the difference between the loss value taken already into account in the demand function and the calculated value of the actual loss depending on the network.

4. MATHEMATICAL MODEL

4.1. The variables of the model. Denote by E the number of power plants and let m(i) be the number of the modes of operation applicable in the ith power plant $i=1,2,\dots,E$. Hereinafter superscript t will always refer to the period, $t=1,2,\dots,27$.

4.1.1. Mode of operation variable. Let x^t_{ij} be 0-1 variable defined as follows, where $i=1,2,\dots E$, $j=1,2,\dots,m(i)-1$:

$$x^t_{ij} = \begin{cases} 0 & \text{if in power plant } i\text{, in the period } t \text{ the } j\text{th mode of operation or one with a subscript less than } j \text{ works,} \\ 1 & \text{if in power plant } i \text{ in period } t \text{ a mode of operation with a subscript greater than } j \text{ works} \end{cases}$$

In the sequel we shall use the notations x^t_{io} and $x^t_{im(i)}$ too and define them so that $x^t_{io} = 1$ and $x^t_{im\ i} = 0$. Note that

1. $x^t_{ij-1} - x^t_{ij} = 1$ if and only if in power plant i in period t just the jth mode of

operation works $(j=1,2,\dots,m(i))$, else $x^{t}_{i,j-1} - x^{t}_{ij} = 0$.

2. According to the above definition the variables belonging to the modes of operation of a fixed power plant can take in one period only the values $(\dots1,1,\overset{\frown}{1,0},0\dots)$ where the 0 standing in the 1,0 value exchange is in the jth place if just the jth mode of operation works. Among different periods the right-hand shift of the value exchange 1,0 corresponds to a mode of operation exchange reached by a shut off while the left-hand shift of the same corresponds to a start.

3. In the periods belonging to the stagnation phase we have $x^{t_0}_{ij} = x^{t_0+1}_{ij} = x^{t_0+2}_{ij} = x^{t_0+3}_{ij}$,

where t_0 is the first period of the stagnation phase, therefore it is sufficient to have only $x^{t_0}_{ij}$ among the variables of the model.

We well use, however, the symbols $x^{t_0+1}_{ij},\dots,x^{t_0+3}_{ij}$ formally in some relations where simplicit,of the expressions requires them.

4.1.2. <u>Production-level variable.</u> Denote $r(i,j)$ the number of the approximating lines in the approximation of the cost function belonging to the jth mode of operation of power plant i, and P_{ijmin} and P_{ijmax} the minimum and maximum production level of the mode of operation respectively. Denote p^{k}_{ijmin}, p^{k}_{ijmax} the power levels belonging to the terminal points of the kth approximating line of the cost function, where $p^{k}_{ijmin} = p^{k}_{ijmax}$, $k = 1,\dots,r(i,j) -1$, and $P^{1}_{ijmin} = P_{ijmin}$, $P^{r(i,j)}_{ijmax} = P_{ijmax}$ hold. Denote P^{t}_{ij} the operation level in period t of the jth mode of operation of power plant i. In order to determine it let us introduce the variables

$$P^{tk}_{ij}\ ,\ i = 1,2,\dots,E,\quad j=1,2,\dots m(i)\ ,\quad k=1,2,\dots,r(i,j)$$

so that

4.1.2.1. $\quad P^{tk}_{ij} \geq 0\,,$

4.1.2.2. $\quad P^{tk}_{ij} \leq P^{k}_{ijmax} - P^{k}_{ijmin}$

4.1.2.3. $\quad P^{tk}_{ij} > 0,$ only if $P^{t\ell}_{ij} = P^{\ell}_{ijmax} - P^{\ell}_{ijmin}$

for all $\ell < k$, and $x^{t}_{ij-1} - x^{t}_{ij} = 1.$

i.e. if plant i works on the jth mode of operation in period t.

By using these variables the above mentioned level is given by the following sum:

$$4.1.2.4.\quad P_{ij}^{t} = \left(x_{ij-1}^{t} - x_{ij}^{t}\right) P_{ijmin} + \sum_{k=1}^{r(i,j)} P_{ij}^{tk} .$$

The production of power plant i in the period t is equal to

$$4.1.2.5.\quad P_{i}^{t} = \sum_{j=1}^{m(i)} P_{ij}^{t} = \sum_{j=1}^{m(i)} \Big\{ (x_{ij-1}^{t} - x_{ij}^{t}) \cdot P_{ijmin} + \sum_{k=1}^{r(i,j)} P_{ij}^{tk} \Big\} .$$

The daily production equals

$$4.1.2.6.\quad P_{i} = \sum_{t=1}^{27} a_{t} . P_{i}^{t} = \sum_{t=1}^{27} a_{t} \cdot \Big\{ \Big\{ \sum_{j=1}^{m(i)} \Big((x_{ij-1}^{t} - x_{ij}^{t}) P_{ijmin} + \sum_{k=1}^{r(i,j)} P_{ij}^{tk} \Big) \Big\} ,$$

where $a_t = 0,5$ or $1,0$ depending on the duration of period t.

4.1.3. Voltage variable. Denote s the number of the nodes of the network with adjustable voltage and v_i^1, v_i^2, v_i^3, $i=1,2,\dots,s$ the voltage levels of these nodes in the three periods with extreme demands /voltage check periods/.

4.2. Constraints of the model.

4.2.1. Supply conditions. Denote P_{dem}^t the value of the power demand in period t. We require that the power demand be satisfied in each period, i.e.

$$\sum_{i=1}^{E} P_i^t = \sum_{i=1}^{E} \left\{ \sum_{j=1}^{m(i)} \left(\left(x_{ij-1}^t - x_{ij}^t \right) P_{ijmin} + \sum_{k=1}^{r(i,j)} P_{ij}^{tk} \right) \right\} = P_{dem}^t ,$$

$t=1,2,\dots,27.$

4.2.2. Bounds on the power levels.

$$0 \leq P_{ij}^{tk} \leq P_{ijmax}^{k} - P_{ijmin}^{k} , \quad i=1,2,\dots,E; \; j=1,2,\dots,m(i), \; k=1,2,\dots r(i,j) ; \; t=1,2,\dots,27.$$

4.2.3. The variable coupling conditions require that the power level in period t of the jth mode of operation of power plant i should be between the bounds P_{ijmin} and P_{ijmax}, i.e.

$$P_{ijmin} \cdot \left(x^{t}_{ij-1} - x^{t}_{ij}\right) \leqq P^{t}_{ij} \leqq P_{ijmax}\left(x^{t}_{ij-1} - x^{t}_{ij}\right).$$

Tating into account 4.1.2.4. we get the conditions:

$$\left(x^{t}_{ij-1} - x^{t}_{ij}\right)\cdot\left(P_{ijmax} - P_{ijmin}\right) - \sum_{k=1}^{r(i,j)} P^{tk}_{ij} \geqq 0,$$

$i=1,2,\ldots E;$ $j=1,2,\ldots,m(i);$ $t=1,2,\ldots,27.$

4.2.4. Start and stop conditions. These conditions ensure the implication $x^{t}_{ij} = 1 \Rightarrow x^{t+1}_{ij} = 1$ in the shut off periods and the implication
$x^{t}_{ij} = 0 \Rightarrow x^{t+1}_{ij} = 0$ in the start periods.

Denote t_1 the last period preceeding the examined day, $x^{t_1}_{ij}$ the realized value of the mode of operation in the above period, t_2 the serial number of the beginning of the second shut down phase, t_3 and t_4 the serial numbers of the beginning of the first and second starting phase resp., ℓ_1, ℓ_2, ℓ_3, ℓ_4 the lengths of the corresponding phases /in periods/ in the previous sequence.

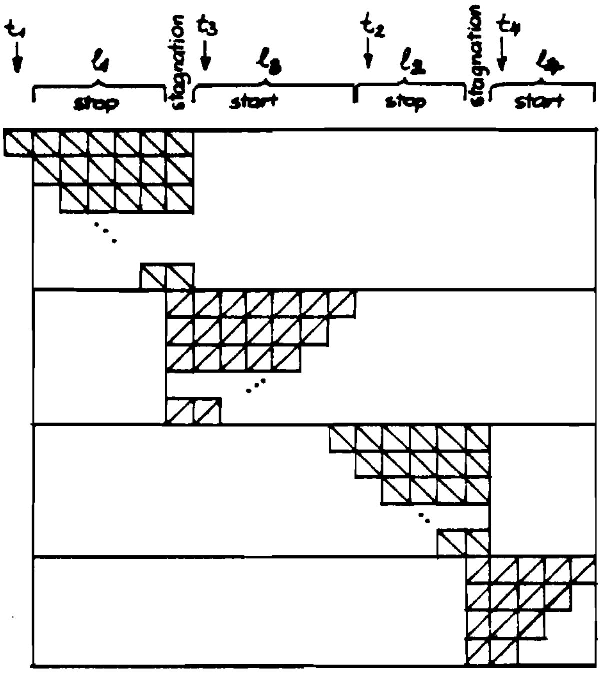


Fig.6.
Structure of the stop and start conditions

The shut off conditions are:

4.2.4.1. $-(\ell_1+1)\, x_{ij}^{t_1} + \sum_{k=1}^{\ell_1+1} x_{ij}^{k} \geqq 0,$ $i=1,2,\dots,E;$
$j=1,2,\dots,m(i)-1.$

4.2.4.2. $\{-(\ell_1+1)+t\}.\, x_{ij}^{t} + \sum_{k=1+t}^{\ell_1+1} x_{ij}^{k} \geqq 0\,,$

$i=1,2,\dots,E;$ $j=1,2,\dots,m(i)-1;$ $t=1,2,\dots,\ell_1.$

4.2.4.3. $-(\ell_2+t)\, x_{ij}^{t_2+t} + \sum_{k=t_2+t+1}^{t_2+\ell_2} x_{ij}^{k} \geqq 0,$

$i=1,2,\dots,E;$ $j=1,2,\dots,m(i)-1;$ $t=-1,0,1,\dots,\ell_2-1.$

The start conditions are the following:

4.2.4.4. $x_{ij}^{t_3-4} + \sum_{k=t_3}^{t_3+t-1} x_{ij}^{k} - (t+1).\, x_{ij}^{t_3+t} \geqq 0$

$i=1,2,\dots,E;$ $j=1,2,\dots,m(i)-1;$ $t=\ell_3-1,\ \ell_3-2,\dots,1.$

$x_{ij}^{t_3-4} - x_{ij}^{t_3} \geqq 0,$ $i=1,2,\dots,E;$ $j=1,2,\dots,m(i)-1.$

4.2.4.5. $$x_{ij}^{t_4-4} + \sum_{k=t_4}^{t_4+t-1} x_{ij}^{k} - (t+1)\, x_{ij}^{t_4+t} \geqq 0,$$

$i=1,2,\dots,E;\quad j=1,2,\dots,m(i)-1;\quad t=\ell_4-1,\ \ell_4-2,\dots,1.$,

$x_{ij}^{t_4-4} - x_{ij}^{t_4} \geqq 0,\quad i=1,2,\dots,E;\quad j=1,2,\dots,m(i)-1.$

Fig.6. shows the structure of the matrix of these conditions.

4.2.5. Fuel constraints. These are constraints with lower and upper bounds, prescribed for the daily production of some power plants. Using 4.1.2.6. we can write them as follows:

$$E_{imin} \leqq \sum_{t=1}^{27} a_t \cdot \left\{ \sum_{j=1}^{m(i)} \left(x_{ij-1}^{t} - x_{ij}^{t} \right) P_{ijmin} + \sum_{k=1}^{r(ij)} P_{ij}^{tk} \right\} \leqq E_{imax}$$

where E_{imin}, E_{imax} are the given bounds, the i's are the subscripts of the power plants with fuel constraints.

4.2.6. Network conditions.

According to the agreement in 3.2., the restrictions resulting from the electrical properties of the network will be taken into account in the three voltage check periods of the day. These conditions are the branch-load, the voltage and the reactive power source conditions. We describe only the content and form of these, the coefficients in the conditions depend on the network /which can be different during the three investigated periods/ and a particular program system was designed for their determination.

<u>The branch-load conditions</u> ensure that the power transmission lines, cables and transformers forming the meshed system which transmits the power from the power plants to the consumers should not be over loaded. These conditions define the load caused by the effective power, viz. with the help of linear approximation of the exact quadratic expressions which yield a very good approximation in the solution domain characterizing the stable operation of the power systems. The form of the condition system is

4.2.6.1. $$-C_T \leq A \cdot \begin{pmatrix} P \\ X \end{pmatrix} \leq C_T,$$

where A is the matrix of the coefficients. The number of its rows is equal to that of the branches, the number of its columns equals that of the sum of the power and mode of operation variables taken into account in the relevant period. C_T contains the loadability of the lines.

The number of these constraints is very large. We may, however, delete many of them and keep only a few that correspond to critical branches.

<u>The voltage conditions</u> ensure the voltage staying within prescribed limits at the nodes of the network. These involve also quadratic formulas where again linear approximation is used resulting in a properly accurate solution in the domain of operation.

The form of these conditions is:

$$4.2.6.2. \quad V_{min} \leqq B \cdot V \leqq V_{max}$$

where B is the matrix of the derived coefficients having as many rows as the number of the nodes of the network, while the number of its columns equals that of the voltage variables. B contains a unit matrix, V_{min} and V_{max} are the allowed minimal and maximal voltage thresholds of the nodes respectively. Actually the system of constraints contains all conditions corresponding to nodes with adjustable voltage, however for the remaining nodes it is sufficient to take into account only a few critical constraints.

Reactive source conditions ensure the reactive power of the reactive sources /performing the voltage control/ not exceeding the allowed leading lagging power maxima, respectively. The reactive powers of the reactive sources are expressed by the voltages of the relevant nodes that we linearize around a given basepoint. This condition has the form

$$4.2.6.3. \quad Q_{min} + \Delta Q_{min} \cdot \underline{x} \leq C \cdot V + Q_{const} \leq Q_{max} + \Delta Q_{max} \cdot \underline{x}$$

where Q_{min}, Q_{max} limit the allowed leading and lagging power, respectively in s nodes, ΔQ_{min}, ΔQ_{max} contain the reactive power threshold changes resulting from the mode of operation change, C is the sxs matrix defining the change of the reactive supplies, Q_{const} is a constant vector with s elements, these elements being the reactive power supplies of the sources defined by the initial state of the vector.

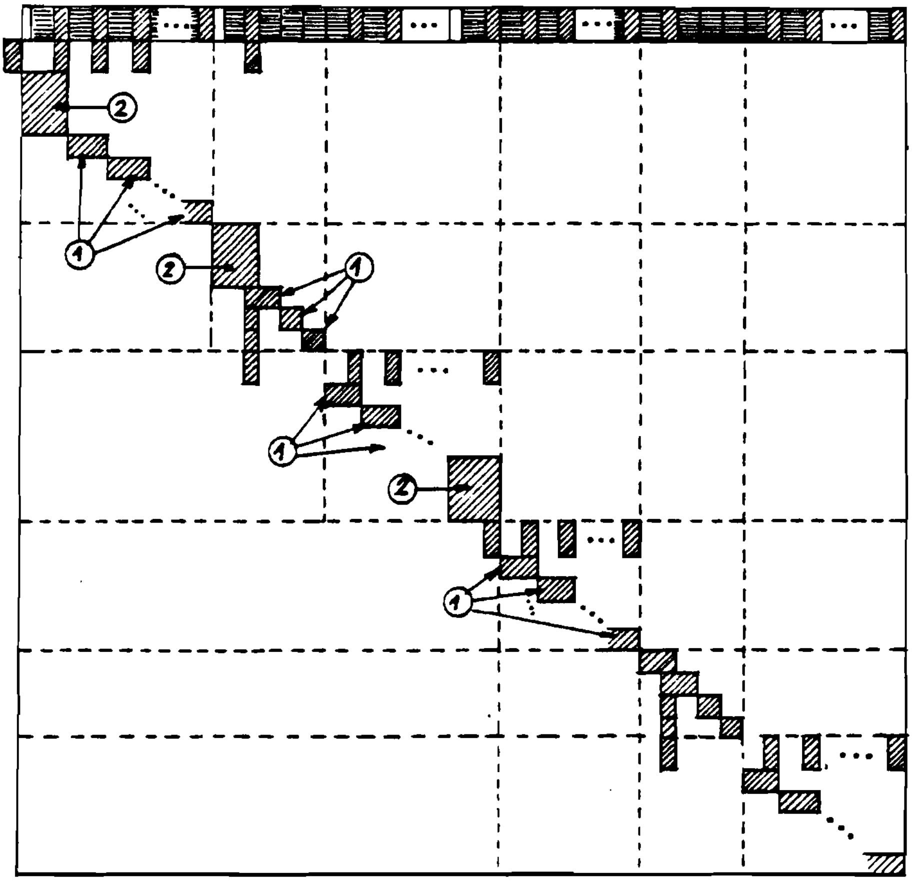


Fig.7.

Structure of the coefficient matrix of the whole model, where ① and ② have the structures given in Fig.8.and Fig.9.

4.3. Definition of the objective function. The objective function to be minimized consists of three parts:

$$K = K_1 + K_2 + K_3$$

where K_1 is the cost of power production, K_2 the cost of still-stand and K_3 the cost entailed by the network loss.

4.3.1. Definition of K_1. Denote C_{ij}^{k} the slope of the kth linear section of the function approximating the one-hour production cost curve of the jth mode of operation of power plant i , and C_{ij}^{o} the production cost of the level P_{ijmin}.

With these notations the cost of production on the level P_{ij}^{t} amounts to

4.3.1.1. $$K_{ij}\left(P_{ij}^{t}\right)= C_{ij}^{o} + \sum_{k=1}^{r(i,j)} C_{ij}^{k} P_{ij}^{tk}$$

if in the i-th power plant just the j-th mode of operation works. Thus

4.3.1.2. $$K_1 = \sum_{t=1}^{27} a_t \cdot \sum_{i=1}^{\Xi} \sum_{j=1}^{m(i)} e_{ij}^{o} \cdot \left(x_{ij-1}^{t} - x_{ij}^{t}\right) + \sum_{k=1}^{r(i,j)} C_{ij}^{k} \cdot P_{ij}^{tk}$$

Note that $c^{1}_{ij} < c^{2}_{ij} \dots < c^{r(ij)}_{ij}$ always holds, from which the fulfilment of the requirement 4.1.2.3. follows for such a solution which satisfies the coupling condition 4.2.3. and for which K_1 is minimal.

4.3.2. Definition of K_2. Fig.5. shows the cost function of the still-stand /or restarting/ of the j-th mode of operation of power plan i as the function of the duration of the still stand. The function can be described by the formula

$$4.3.2.1. \quad g_{ij}(\tau) = g_{ij}(0) + (g_{ij}(\infty) - g_{ij}(0)) \cdot (1-e^{-C_{ij}\tau}),$$

where $g_{ij}(0)$, $g_{ij}(\infty)$ and C_{ij} are the constants characterizing the power plant and the mode of operation, $g_{ij}(0)$ denotes the cost of starting without still-stand, and $g_{ij}(\infty)$ the cost of the so-called cold starting.

In accordance with the assumption 3.3.b, if a mode of operation is stopped with ℓ periods before the beginning of the stagnation phase, then its effect in the cost function will be taken into account with the value $g(4+2\ell)$. The corresponding value will be constructed with the help of properly chosen coefficients as a sum consisting of terms corresponding to the duration of the still-stand, - and the complete still-stand cost will take the form

4.3.2.2. $$K_2 = \sum_{t=1}^{27} \sum_{i=1}^{E}{}' \sum_{j=1}^{m(i)-1} d_{ij}^{t} \cdot x_{ij}^{t}$$

where d_{ij}^{t} is the properly chosen coefficient defined by the utilization of the function $g(\tau)$.

4.3.3. Definition of K_3.

4.3.3.1. $$K_3 = \sum_t K_3^t$$

where t runs through the indices of the tree voltage check periods.

The determination of the components of K_3^t - i.e. of the coefficients participating in its definition,- is a part of the procedure serving for the determination of the network conditions. We disregard its description, and give only the formulae :

$$4.3.3.2. \quad K_3^t = \sum_{i=1}^{E} \sum_{j=1}^{m(i)} \left(a_{ij}^t x_{ij}^t + \sum_{k=1}^{r(ij)} b_{ij}^t F_{ij}^{tk} \right) + \sum_{\ell=1}^{s} h_\ell^t v_\ell^t + C_1^t + C_2^t .$$

5. A survey of the model structure

Fig.7. is the schematical representation of the above described model. In its survey we point out that the conditions of the model have the following properties:

1. The fuel constraints contain besides the voltage variables all variables belonging to the given power plants and so practically they connect the variables of all the 27 periods.

2. The start-stop conditions contain the mode of operation variables of the corresponding phase, - these conditions connect the periods belonging to the given phases.

3. The connection among the particular phases is realized by the mode of operation variables belonging to the stagnation phase, these at the same time connect the periods belonging to the stagnation phase.

4. Further conditions of the model contain variables belonging to single periods only, the

structures of these conditions are shown in Figs.8. and 9., respectively, - depending on the corresponding period being one without network conditions.

The size of the model - in choosing everywhere r(ij)= 1 for the approximation of the cost function and taking the real size of the power system into account - is at most as follows:

the number of variables: 35 power variables for each period, 21 mode of operation variables and in the **voltage check** periods maximum 3o voltage variables, i.e. the number of continuous variables is 945 + 90 and the number of 0-1 variables is 441. The number of constraints amounts to about 1700, from these 420 conditions are start-stop conditions containing only 0-1 variables.

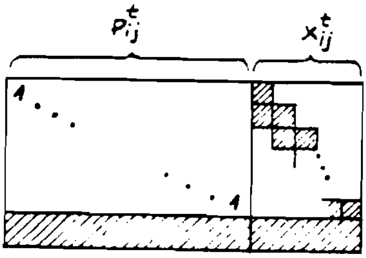


Fig.8.

Structure of the conditions in a "normal" period

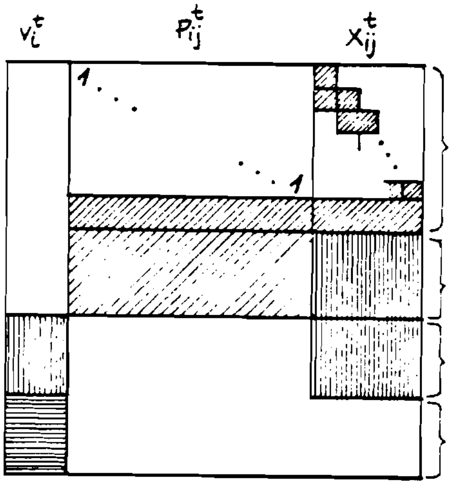


Fig.9.

Structure of the conditions in a special period

6. HOW TO SOLVE THE MODEL?

In order to complete our work we had to write a computer program for the CDC 3300 computer of the Hungarian Academy of Sciences for the solution of the problem. From among the possible ways we had the idea to apply the Benders decomposition method to solve the whole problem. This was rejected because, on one hand, it can happen that we will obtain a feasible solution only in the last step, so that if on account of computer time limitation the run had to be interrupted, the results till then would not contain the necessary information. On the other hand, there is a large number of variables of the pure 0-1 problems to be solved in the iterations of the decomposition and their constraints do not have favourable special structure. We thought of a version of the branch-and-bound algorithm in which the relevant linear programming problem could have been solved by the Dantzig-Wolfe decomposition, but because of the large number of the 0-1 variables we have rejected this idea, too.

Finally we have accepted the following algorithm:

1./ We disregard the fuel constraints.

2./ We solve the remaining large-scale mixed integer programming problem - in which the connections among the periods are ensured by the start-stop conditions and the mode of operation variables of the stagnation phases - the following way(Fig.1o.) :

We solve successively the three mixed integer programming problems corresponding to the voltage check periods. We allow in the solution of the first problem every mode of operation applicable on the given day. In the solution of the second problem we allow only that modes of operations which are realizable from the modes of operations in the solution of the first problem by shut off. For the third problem we allow that modes of operations, realizable from the solution of the second problem by starting.

Thereafter we solva the intermediate problems and the problems corresponding to the following periods successively, by taking always the variables of the modes of operation of the neighbouring, already solved problems and the connections of the periods to the start-stop phases into account.

In every case the Benders decomposition method will be applied for the solution of the problem corresponding to one period.

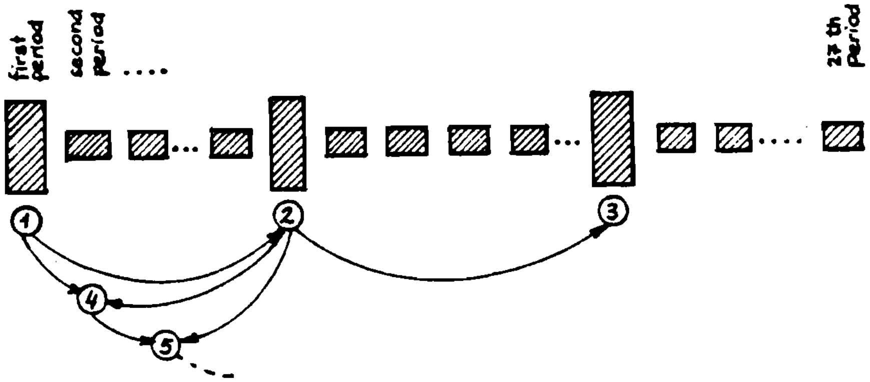


Fig.1o.

The successive mode of solving the mixed-integer programming problem without fuel-conditions, where the numbers in circle indicate the order of the executions of computations

3./ We check whether the fuel constraints are satisfied for the obtained solution. If yes, then the algorithm ends, else the following iterative procedure will be applied.

4./ If in a power plant the daily power production is less then what is prescribed then the production cost coefficients of the given power plant will be multiplied by a multiplier less than 1, and if the daily power production is greater than what is prescribed, then they will be multiplied by a multiplier greater than 1. The values of the mode of operation variables will be fixed and the corresponding linear programming problem will be solved. If in the course of the solution the fuel constraints are satisfied by the new outputs obtained, the iteration ends.

Otherwise there are two cases: i/ if in the course of the iteration processes we have already found solutions indicating underproduction and overproduction, too, then we will proceed according to paragraph 5; ii/ else we will modify again the cost coefficients and repeat the solution of the linear programming problem.

5./ The mode of operation values of the solution accepted as optimum are the fixed modes of operation and the production level will be defined by such a linear combination of any particular solution indicating the underproduction and overproduction which satisfies the fuel constraint.

Remark: The physical background and the preliminary survey of the data ensures that the described algorithm works well, i.e. it cannot occur that a mixed problem corresponding to a period has no feasible solution or that we obtain only such solutions in the 4-th step which violate this constraint only in the same direction.

7. CONCLUDING REMARKS

This paper gives only a short survey of the most important features of the model, without any claim to completeness. A brief sketch of the whole computer program system is shown on Fig.11, and a study covering also details not discussed in this paper /e.g. computation of loss, determination of the network conditions etc./ is under preparation.

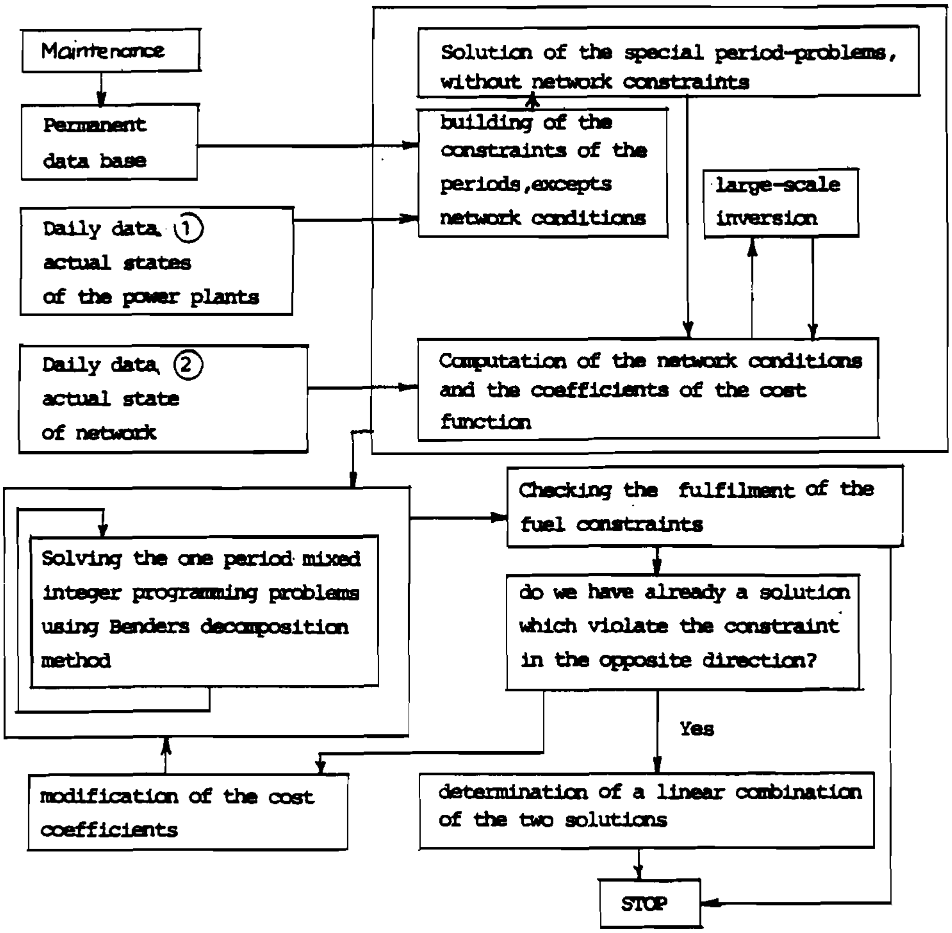

Maintenance
Permanent data base
Daily data ① actual states of the power plants
Daily data ② actual state of network
Solution of the special period-problems, without network constraints
building of the constraints of the periods, excepts network conditions
large-scale inversion
Computation of the network conditions and the coefficients of the cost function
Solving the one period mixed integer programming problems using Benders decomposition method
Checking the fulfilment of the fuel constraints
do we have already a solution which violate the constraint in the opposite direction?
Yes
modification of the cost coefficients
determination of a linear combination of the two solutions
STOP

Fig.11.

REFERENCES

1./ I.Hano, Y.Tamura, S.Narita, K.Matsumote:
Real time Control of System voltage and reactive power IEEE Trans PAS-88, No-10, 1969, pp. 1544-1559.

2./ C.M.Shen, M.A.Langhton:
Power System load scheduling with security constraints using dual linear programming. Proc. IEE. Vol. 117, No.-11, November 1970, pp.2117-2127.

3./ A.de Perna, E.Mariani:
Programmazione giornaliera delle centrali idroelettriche a bacine e a serbatoio in un sistema di produzione misto. L'Energia Elettrica-No.7, 1971, pp.437-448.

4./ B.Stott:
Review of Load Flow Calculation Methods. Proc. of IEEE Vol.62, No.-7, 1974. pp.916-929.

5./ O.I.Elgerd:
Electrical Energy Systems Theory:

An Introduction. McGraw Hill Book Co., 1971.

6./ Benders,J.F.:
Partitioning procedures for solving mixed variable programming problems. Numerische Mathematik, 1/238-252, 1962.

A MATHEMATICAL MODEL FOR THE DETERMINATION OF OPTIMAL CROP PRODUCTION STRUCTURES

Zsolt Harnos

Bureau for Systems Analysis
of the State Office for Technical Development
Budapest

In assessment of the agro-ecological potential, the main goal was to determine the maximal amount of plant production in terms of optimal utilization of the possibilities offered by the natural environment and to investigate the consequences of such a policy.

A two level hierarchical model was constructed for the analysis of crop production.

The models are described by systems of inequalities parametrized in the right hand side.

The constraints can be grouped as follows:

- area constraints,
- constraints of the production structure,
- crop rotation conditions ensuring the continuity of production,
- constraints regulating the extent of land reclamation and irrigation investment.

The solutions are special Pareto optimal points of the feasibility set.

During recent years, throughout the world, increasing attention has been paid toward assessing natural resources, working out possibilities for their utilization. Today this assessment includes not only the energy resources, raw materials but also the so called "biological resources". It is expecially important to be familiar with the interaction between the natural environment and plant and animal production to discover the hidden reserves in biological resources, the possibilities and limits of their utilization.

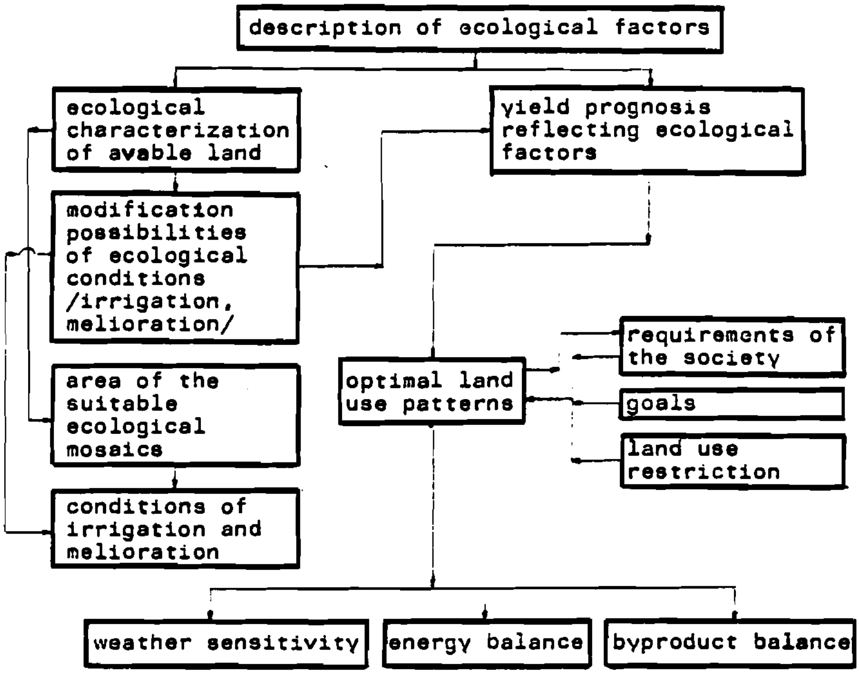
In Hungary, work on the estimation of agroecological potentials started in 1978 at the initiative of the Hungarian Academy of Sciences and was finished in the spring of this year.

At the assessment of the agroecological potential, the main goal was to determine the maximal amount of plant production as a result of optimal utilization of the possibilities offered by the natural environment and to investigate the consequences of such a policy.
In concrete terms, this meant the determination of land use patterns optimally utilizing the ecological conditions that

- can be realized in principle,
- meet the requirements of the society,
- and are optimal with respect to some goal.

Realizability means the use of data and hypotheses in the model that can be expected by reasonable standards to be valid at the turn of the millennary.

Meeting the requirements of the society means, the capability to supply the society with all the products determined by the projected structure of consumption.
Optimality means an in some sense optimal compliance of the land use structure with the ecological conditions.
After this short introduction, the presentation of the model describing crop production follows, with the structure of the model shown in the figure below.


description of ecological factors
ecological characterization of avable land
yield prognosis reflecting ecological factors
modification possibilities of ecological conditions /irrigation, melioration/
requirements of the society
optimal land use patterns
goals
area of the suitable ecological mosaics
land use restriction
conditions of irrigation and melioration
weather sensitivity
energy balance
byproduct balance

The first problem was to determine the attainable level of yields in 2000 given the natural environment of Hungary /precipitation, temperature, soil, relief, hydrology etc./ an the genetic potential of the species. For this end a yield prognosis was prepared, the structure of the resulting data baeis is shown in Table 1.
The methodology and detailedness is described in the following papers [3], [9].

The model describing crop production is based on this data basis.
The main goals of the computations were:

- the assessment of production capacity of crop production under different circumstances,
- the analysis of the relationships between land use patterns complying with the natural conditions and the required total production /social demand/,
- the analysis of the development of land use pattern and total production as functions of the quantity and quality of available land,
- the analysis of the dependence cf land use patterns and total production on the amount of investments into land reclamation and on their way of realization,
- the analysis of the relationships between irrigation and land use patterns
  etc.

The large number of crops and habitats considered resulted in about 5000 variablee. This situation, in fact, determined the method; as the only solvable problem in this case is the one using linear programming techniques, the same being true even after excessive aggregation.

A two level hierarchic model was constructed for the analysis of crop production.

The first so called regional model describes the problem in an aggregated form. The so called ecological regions constitute the land units here. /See Figure 2./

The requirements of the society with respect to the production structure and land reclamation investment conditions and others are formulated in the constraints of this model.

The result gives a rough, regional allocation of the investments and land use. The global analysis of the crop production system and that of the dependence of land use and product structure on the conditions and the goals is carried out by using this model.
Detailed computations considering ecological mosaics are carried out on the other level.

The whole of the country was divided into four large regions as is shown in Figure 2., and the crop production activity in them are described by separate problems. The structure of these models is similar to that of the regional model which will be outlined in the sequel. It is the regions are considered homogeneous in the regional model while the same is true only for the ecological mosaics in the others.
The constraints of the detailed models /as far as the product structure, the allocation of land reclamation investments and even the goal function are concerned/ were formulated on the basis of the results of the regional model.

Our computations give detailed information about the land use pattern being in good compliance with the ecological conditions and about the allocation both in space and time order of land reclamation investments.
Before going into the details of the constraints of the regional model we shortly give a formal definition of the model system.

The regional model is described by a system of inequalities parametrized in the right hand side:

$$\begin{aligned} &\underline{\underline{A}}\,\underline{x} \leq \underline{b}_o + \lambda(\underline{b}_1 - \underline{b}_o) \\ &\underline{x} \geq \underline{0} \\ &\lambda \in [0,1] \end{aligned} \qquad /1/$$

Let us denote the set of the solutions of the above system by Ω.

Our task is to determine an $x^* \in \Omega$, with all the goal functions

$$\varphi_i(\underline{x}) = \langle \underline{c}_i, \underline{x} \rangle \qquad i \in I = \{1,2,\dots,\ell\}$$

reaching their optima, that is

$$\varphi_i(\underline{x}^*) = \max_{\underline{x}\in\Omega} \varphi_i(x), \qquad i \in I.$$

This optimization problem, however, has no solution in general [4] , and for this reason we have to find special Pareto-optima, that is such $\underline{z}^* \in \Omega$ for which

$$\underline{\varphi}(z^*) = \max\{\underline{y} : \underline{y} = \varphi(\underline{x}), \ \underline{x} \in \Omega\}$$

The maximum here is taken over R^ℓ with respect to the ordering induced by the natural positive cone R^ℓ_+.

That is to say:

$$\underline{\varphi}(\Omega) \cap (\varphi(z) + R^\ell_+) = \{\underline{\varphi}(\underline{z}^*)\}$$

Two, so called compromise solutions were determined from the set of Pareto optimal points.

In the first step the utopia point in $\mathbb{R}^{l}$ was determined for the problem / 1 /.

The i-th coordinate of the utopia point is $\beta_i = \varphi_i(\underline{x}^{(i)})$ where $\underline{x}^{(i)}$ the solution of the problem:

$$\begin{aligned} \underline{\underline{A}}\,\underline{x} &\leq \underline{b}_o + \lambda(\underline{b}_1 - \underline{b}_o) \\ \underline{x} &\geq \underline{0} \\ \lambda &\in [0,1] \\ \varphi_i(\underline{x}) &\longrightarrow \max \end{aligned}$$

We considered two new goal functions by using the utopia point, point,

$$\Psi_1(\underline{x}) = \sum_{i=1}^{l} \left(1 - \frac{\langle \underline{c}_i, \underline{x} \rangle}{\beta_i}\right)$$

and

$$\Psi_2(\underline{x}) = \max_{1 \leq i \leq l} \left(\beta_i - \langle \underline{c}_i, \underline{x} \rangle\right)$$

then we minimized them on the set Ω.

These solutions are Pareto-optimal points of the system / 1 /. The solutions of the regional model produced land use patterns on the regional level. By their use, the production structure and the extent of land reclamation and irrigation were determined.

Taking them as constraints and taking their corresponding goal functione, the linear programming problem describing the crop production of the four large regions were solved.

Now we arrived to the description of the main relationships and to the explanation of our choice of methodology.

The constraints can be grouped as follows:

- area constraints,
- constraints of the product structure,
- crop rotation conditione ensuring the continuity of production,
- constraints regulating the extent of land reclamation and irrigation investment.

Cropland was considered to be homogeneous in the regional model, with three kinds of possible activity:

- production corresponding the present situation,
- production corresponding to the situation after land reclamation /melioration/,
- production on both reclaimed and irrigated land.

The area of irrigable and reclaimed land was limited in each region.

The total area cultivated in the three possible ways had to be equal to the total cropland in the region. The total available cropland in the regions was changed according to the amount of land under non agricultural use.

The demand that crop production had to meet consisted of two parts:

- home consumption,
- exports.

At formulating the demand, the following points were to be considered:

- immediate public consumption,
- consumption ensuring the continuity of production and reproduction.

The public consumption is the function of the number of the population and eating habits, in the firts place.

Three different consumption structures were considered:
consumption corresponding to the present Hungarian, West-European and physiologically right nutrition.

This is the point where animal husbandry is linked into the system.

The fodder needs of an appropriate stock of cattle and sowing seed for keeping the level of production had to be reckoned with to ensure the continuity of food production.

This consumption model served as the basis for the determination of the minimal amount of products to be produced. Upper bounds were given for crops that cannot be exported and home consumption is also limited.

The third group of constraints is for the control of the territorial structure of the production. Is it the territorial constraints determined for each region that ensure the realizibility of the rotation plan.

These are of two kinds:

- those given in the form of a limit for the ratio between the area occupied by the crops or groups of crops, respectively

- those limiting the area occupied by certain crops or groups of crops from above or below.

Similar conditions were formulated for irrigated or reclaimed land and for the ratio between irrigated and dry cultivation. All the above mentioned parameters were expressed in natural units and the same is true for the constraints, as well. There was, in fact, one single condition of a non ecological character, and this was the extent of land reclamation inveetments.

This ie a eignificant meane for increasing yiald, but it cannot be expected that all the reclamation work will have been finished in the near future.

In the course of our inveetigations, more than 20 different forms of land reclamation were considered, with different investment requirements. The rise of yield due to land reclamation being known, investment coete in current prices were sufficient to determine the optimal allocation and time order of land reclamation projects. The volume of material investment was limited. The solutione under the different investment constraints gave the opportunity to determine the expedient location and time order of land reclamation projects.

The structure of the outlined model can be seen in the figure below:

$$\begin{pmatrix} & A_t & \\ & A_y & \\ A_1 & & 0 \\ & A_2 \; \ddots & \\ 0 & & A_{35} \end{pmatrix} \underline{x} = \begin{pmatrix} \underline{b}_t \\ \underline{b}_y \\ \underline{b}_1 \\ \cdot \\ \cdot \\ \cdot \\ \underline{b}_{35} \end{pmatrix} \qquad /\ 2\ /$$

$$\underline{b}_t = \underline{b}_o^t + \lambda(\underline{b}_1^t - \underline{b}_o^t)$$

$$\underline{b}_o^y \leq \underline{b}^y \leq \underline{b}_1^y$$

$$\underline{b}_o^k \leq \underline{b}^k \leq \underline{b}_1^k \qquad k = 1,\ldots,35$$

Some of the lower bounds equal to zero while some of the upper bounds may be infinite, meaning that there is no limitation. The system of inequalities means a series of problems of an ever growing size but of constant structure. The matrices A_t and A_y were the same in all cases while in the matrices A_k, relationships controlling the land use pattern were gradually extended. The solution in the less constrained cases made great differences between the production areas of the individual crops. By the gradual extension of the conditions, however, the land use pattern reached a stable form, that is from a certain step onwards the different goals did not made the land use pattern change significantly.

The knowledge of such stable systems is important, because the product mix can be changed without substantial modifications of the structure of the agricultural production, and hence the planning of the agricultural infrastructure can be brought into harmony with the stable - though versatile - land use pattern.

The description of the parameters serving as a basis of the production and of the main forms of the factors influencing production is herewith finished.

This is described in a concise form by the inequality system

$$\underline{\underline{A}}\,\underline{x} \leq \underline{b}_o + \lambda(\underline{b}_1 - \underline{b}_o)$$

$$\underline{x} \geq \underline{0}$$

$$\lambda \in [0,1]$$

The possible land use patterns are represented by the solutions of this system.

The main problem here is to choose the criterion of optimality.

The usual goals in economic planning - like the maximization of net income, the minimization of costs - were not suitable as both the costs /inputs/ and the products were counted in natural units.

Hence, goals could be formulated by the way of some fictive price system, and so we used a number of comparative value systems. "Price systems", in this case, were needed only for the analysis of sensitivity of the system and not for the determination of some sort of profit.

The comparative value systems were based on some indicator of the internal content of the products like e.g. protein content, energy content, grain unit and so forth, and then the optimal product and land use structure under the different limitation levels were analized.

Obviously, because of the extreme characteristics of such value systems, an economy cannot adapt a production structure being optimal with respect to them, but the results themselves are interesting as they show the maximal possibilities in some directions.

Knowing these maximal possibilities, compromise solutions with respect to certain groups of the goal functions or to all of them were also determined.

THE DATA BASIS AS AFFECTED BY AGROECOLOGICAL CONDITIONS

Crop : $n = 1,2,\dots,13$
Region : $k = 1,2,\dots,35$

		climatic year types				area of the soil type	land suitable for		yield	rise of yield due to	
		1	2	3	4		melioration	irrigation		melioration	irrigation
		yield t/ha				in hectares			t/ha	in t/ha	in t/ha
soil types	1	$a^1_{n,k,1}$	$a^2_{n,k,2}$	...	....	$T_{k,1}$	$T^m_{k,1}$	$T^\ell_{k,1}$	$a_{n,k,1}=\sum_{i=1}^{n} a^i_{n,k,1}P_i$	$b_{n,k,1}$	$c_{n,k,1}$
	2	$a^1_{n,k,2}$	$a^2_{n,k,2}$	...	....	$T_{k,2}$	$T^m_{k,2}$	$T^\ell_{k,2}$	$a_{n,k,2}=\sum_{i=1}^{n} a^i_{n,k,i}P_i$	$b_{n,k,2}$	$c_{n,k,2}$
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
the frequency of the climatic year types		P_1	P_2	P_3	P_4						
the characteristic data of the region						$T_k=\sum_{j=1}^{34} T_{k,j}$	$T^m_k=\sum_{j=1}^{34} T^m_{k,j}$	T^L_k	$a_{n,k}=\frac{\sum_{j=1}^{34} a_{n,k,j}}{}$	$b_{n,k}=\frac{\sum b_{n,k,j}T^m_{k,j}}{T^m_{k,j}}$	$c_{n,k}=\frac{\sum c_{n,k,j}T^\ell_{k,j}}{T^\ell_{k,j}}$

In the table : $0 \le T^\ell_{k,j} \le T^m_{k,j} \le T_{k,j}$; $a_{n,k,j}= b_{n,k,j}= c_{n,k,j} =0$ if $T_{k,j}= 0.$

Where : $0 \le p_i \le 1,$ $p_1+p_2+p_3+p_4 = 1$

Table 1.

References

1 R.K. Eyvinelson, E.O. Heady and U.K. Srivasta -
A Model Incorporating Farm Size and Land
Classes

2 A.D. Meister, E.O. Heady, K.J. Nicol, R.W. Strohbehn -
U.S. Agricultural Production in Relation to
Alternative Water, Environmental, and Export
Policies
Center for Agricultural and Rural Development
Iowa State University Card. R. 65/1976/

3 Zs. Harnos, B. Győrffy -
Estimation of Crop Yield Potentials as Affected
by Agro-ecological Conditions
7-th IIASA Global Modelling Conference 1979,
Laxenburg

4 J.G. Lin -
Maximal Vector and Multiobjective Optimization
JOTE Vol. 18. /1976/

5 V.M. Ozernoi and M.G. Gaft -
Multicriterion Decision Problems -
Conflicting Objectives in Decision
Edited by D.E. Bell, R.L. Keeney, H. Raiffa
1. Int. Series on Appl. Systems Analysis /1977/

6 M. Peschel and C. Riedel -
Use of Vector Optimization in Multiobjective
Decision Making
Conflicting Objectives in Decision
Edited by D.E. Bell, R.L. Keeney, H. Raiffa
1. Int. Series on Appl. Systems Analysis /1977/

7 M.E. Salukvadze -
Vector optimization problems control theory
/in russian/
Mecmiereba" Tbilisi, 1975

8 I. Vályi -
Statistical Evaluation of Experts' Estimates
IIASA Task Force Meeting on "Modelling of
Agricultural-Environmental Processes Related
to Crop Production"/1980/

9 A.P. Wierzbicki -
The Use of Reference Objectives in Multiobjective
Optimization Theoretical Implications and
Practical Experience
IIASA Working Paper A-2361 /1979/

10 P.L. Yu and G. Leitmann -
Compromise Solutions, Domination Structures
and Salukvadze's Solution
JOTE Vol. 13. /1974/

HOLISTIC PREFERENCE EVALUATION IN MULTIPLE CRITERIA OPTIMIZATION*

James K. Ho

Applied Mathematics Department
Brookhaven National Laboratory
Upton, N.Y.

This paper presents an interactive, parametric linear programming method: HOPE (for holistic preference evaluation) to solve the multiple criteria linear programming problem. The method uses the decision maker's ordinal priority ranking of the criteria to structure the parametrizations, and by successively eliciting his holistic preference among alternative efficient solutions, refines the structure until a most preferred solution is established. Advantages over existing techniques include ease of implementation and use, absence of explicit analysis of marginal utilities (or trade-off curves), and an intuitive interpretation as a learning process for the decision maker to "weigh" the criteria. Numerical examples from forest management and university planning are given.

*The submitted manuscript has been authored under contract EY-76-C-02-0016 with the U.S. Department of Energy. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

1. Introduction

With the advent of high speed digital computers and sophisticated implementation of the simplex method, linear programming (LP) [4] has become one of the most powerful algorithmic tools in operations research and management science. Using LP, decision makers can determine optimal solutions from among all feasible solutions in a decision process that can be mathematically modelled as the optimization of a linear function in the decision variables subject to a set of linear inequality constraints.

Yet one of the major limitations to the utility of the well developed LP approach is that only a single objective function can be optimized at a time. In practice, most decision processes involve necessarily a multitude of conflicting criteria. For example, in the planning or control of any operation, minimal cost, maximal reliability and optimal performance are all desirable objectives. However, such criteria cannot in general be optimized simultaneously, and the best compromise solution, in some appropriate sense, is sought.

The multiple criteria linear programming (MCLP) problem is then

$$\begin{array}{ll} \text{"maximize"} & c_k x,\ k=1,\ldots,K \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

where

$x \epsilon R^n \times 1;$

$c_k \epsilon 1 \times R^n,\ k=1,\ldots,K;$

$A \epsilon R^m \times R^n;$

$b \epsilon R^m \times 1;$

and the quotation marks indicate that the meaning of optimality has yet to be specified.

Many methods have been proposed in recent years. For a survey, the reader is referred to [3] and [14]. Some methods (eg. [3]) are based directly on LP and are therefore easy to implement and deploy. Others such as [2] have intuitive appeal to the decision maker. Still others, such as [11], have a more rigorous theoretical basis. To be operational they all depend on various assumptions that limit their robustness in dealing with diverse, real-life problems. To date, no proposed method seems to be attractive enough by all the above standards[1] to become an integral part of the practice of linear programming.

In this paper, a method is presented to extend the algorithmic tools of LP for MCLP. Section 2 reviews the concepts of utility, preferences, priorities and efficiency that are useful in defining optimality for multiple criteria decision processes. The conceptual bases as well as limitations of existing approaches: direct assessment [8], goal programming [3] and multiobjective programming [14] can be viewed in a unified framework. For a detailed discussion, the reader is referred to [12]. In Section 3, a unifying approach which will be termed Holistic Preference Evaluation and abbreviated as HOPE is developed. It is shown that HOPE combines many advantages of existing approaches while circumventing some of their difficulties.

[1] Ironically, this begins to sound like a multiple criteria problem in itself.

After a discussion of the basic assumptions, an algorithm for HOPE is presented in Section 4. As it is based simply on the iterative application of parametric LP, the HOPE algorithm can be implemented quite easily. Convergence is finite, and although heuristic in nature, justification for its robustness is given in Section 5. Numerical examples are presented in Section 6 to demonstrate how the HOPE procedure works in realistic situations. The simple numerical example in [20], the forest management model in [16], and the academic department planning model in [11] are used. The results provide ample evidence that HOPE can be a robust method for MCLP. Concluding remarks and an outline of further development are given in Section 7. The test problems and relevant data for the larger examples are included in Appendices A and B.

2. Utility, Preferences and Priorities

To define optimality for MCLP, it is assumed that the decision maker's (DM) value judgment can be expressed by an additive utility function

$$u : R^K \rightarrow R \quad \text{such that}$$

$$u \equiv \sum_{k=1}^{K} u_k \quad \text{where, for each } k=1,\dots,K,$$

$$u_k : R \rightarrow R \quad \text{is monotone increasing.}$$

Indeed, $u_k(c_kx)$ is interpreted as the utility [7] to the DM attained by the k^{th} criterion while $u(c_1x,\dots,c_Kx)$ is the overall utility when x is chosen. Often, each u_k is further assumed to be linear, concave or at least quasi-concave.

Let $X \equiv \{x \epsilon R^{n} x 1 | Ax=b, x \geq 0\}$ be the set of feasible solutions to MCLP, assumed bounded for simplicity, and $V=\{v \epsilon R^K | v=(v_1,\dots,v_K); v_k=c_kx, k=1,\dots,K, x \epsilon X\}$ be the corresponding feasible set in criteria space.

Definition. For a DM with utility function u, $x^* \epsilon X$ is an optimal solution to MCLP if $u(c_1x^*,\dots,c_Kx^*) \geq u(c_1x,\dots,c_Kx) \forall x \epsilon X$.

A necessary condition for x to be optimal is that of efficiency [10]. (also known in the literature as nondominance [18], Pareto optimality, noninferiority or admissibility).

Definition. $x^0 \epsilon X$ is efficient if $\nexists\ x^1 \epsilon X \ni c_kx^0 \leq c_kx^1$, $k=1,\dots,K$ with strict inequality for at least one k.

A solution is efficient if it is not possible to increase the value of any criterion without diminishing that of at lease one other.

Let $E \equiv \{x \in X | x$ is efficient$\}$ be the set of all efficient solutions to MCLP.

Proposition 1. If x^* is optimal, then $x^* \in E$.

Proof. Follows from monotonicity of each u_k and additivity of u. Therefore, only efficient solutions need be considered in the optimization of MCLP. The following proposition provides a very useful characterization of efficiency.

Proposition 2. $x^0 \in E$ if and only if it maximizes

$$\bar{u} = \sum_{k=1}^{K} \lambda_k c_k x \text{ over } X$$

for some $\lambda_k > 0$, $k=1,\ldots,K$.

Proof. See [10] or [13].

Without loss of generality, $\{\lambda_k\}$ is normalized so that

$$\sum_{k=1}^{K} \lambda_k = 1.$$

There is considerable mathematical interest to develop enumeration methods that seek to determine all efficient solutions [13]. However, MCLP cannot be optimized without further knowledge about u. In general, it is quite difficult to assess u explicitly. See, e.g., [5], [8] and [15]. In fact, the explicit form of u is irrelevant if the weak ordering of $x \in X$ induced by u is assumed. See e.g., [1] and [6]. This means that given two solutions, the DM can determine by his preference judgment whether he prefers one to the other or that he is indifferent about the two. It is then possible to derive algorithms that iterate from one solution to another that is preferred by the DM. This is the preference programming approach discussed in [12]. Examples are the multiobjective programming methods proposed in [11], [16] and [20]. Shortcomings of this approach arise from its reliance on local (e.g., adjacent solutions) or marginal preference (e.g., tradeoffs) analysis that may call for infinite sensitivity in the DM's preference judgment. To alleviate this difficulty more ellaborate techniques have been introduced, e.g. in [16].

A different assumption about the DM's value judgment is often valid in practice, namely his priorities. Intuitively, it means that the DM considers some criteria more important than others. Formally, an ordinal ranking of the criteria is considered. For simplicity, suppose $c_{1,} c_2, \ldots, c_K$ are given in descending order of priority (importance). By appropriate scaling, the underlying utility function u can be assumed to satisfy

$$u_1(y) \geq u_2(y) \geq \cdots \geq u_K(y) \; \forall y \in R.$$

If in addition, the DM can estimate his cardinal ranking (or degree of priority) of the criteria, the priority programming approach discussed in [12] can be applied to MCLP. Examples include the whole class of goal programming methods [3] as well as the method in [2]. Shortcomings of this approach arise from the need to quantity priority: as penalty weights in goal programming and as concession levels in [2].

A unified framework is presented in [12] for the above approaches to MCLP. In summary, given an MCLP, if the DM's utility function is known explicitly, it can be optimized directly. If reliable techniques are available to assess this function, they can be used before optimization. If the DM can weakly order the solutions, preference programming can be applied. If the DM can rank the criteria, priority programming can be used. If none of the above assumptions holds, the best one can do is to enumerate all the efficient solutions. Within this framework, another case is possible, that in which both priorities and preferences are assumed. This forms the basis of the holistic preference evaluation approach presented in the following sections.

3. Holistic Preference

Definition. When a DM is able to

(i) weakly order the solutions of an MCLP by his preference judgment and

(ii) rank the criteria according to ordinal priority,

we say that his holistic preference can be assessed.

In this sense, holistic preference is interpreted as a value judgment that extends from the pairwise comparison of solutions to the pairwise comparison of criteria as a whole. However, the ability to do (i) and (ii) in the above definition does not necessarily mean that the overall value judgment will be consistent. For example, suppose a priority ranking implies that

$$(1)\quad u_1(y) \geq u_2(y) \geq \cdots \geq u_K(y)\ \forall y \in R.$$

The utility function $u \equiv \sum_{k=1}^{K} u_k$ induces on X a weak ordering by the definitions $x^1 \preccurlyeq x^2$ if

$$(2)\quad u(c_1 x^1, \ldots, c_K x^1) \leq u(c_1 x^2, \ldots, c_K x^2).$$

Let $v_{1,} v_2$ be such that $u_1(v_1)=u_2(v_2)$. By (1) and monotonicity of u_k, we have

$$v_1 \leq v_2 \ .$$

Hence, $u(v_1, v_2, v_3, \ldots, v_K) \leq u(v_2, v_2, v_3, \ldots, v_K)$.· Let x^1 and x^2 correspond to $(v_1, v_2, v_3, \ldots, v_K)$ and $(v_2, v_2, v_3, \ldots, v_K)$ respectively. By (2),

$$x^1 \preccurlyeq x^2.$$

If the DM prefers x^1 to x^2 when actually asked to compare the two, his preference judgment is inconsistent with his priority ranking. When this happens, his holistic preference is said to be inconsistent.

In practice, while it is plausible to expect preference and priority judgment by the DM, the a priori assumption of consistency will in general be too restrictive. This is by no means a reflection on the integrity, intelligence or competence of the DM. Inconsistency may arise naturally from imcomplete information about the problem. If the MCLP is nontrivial at all, the DM may have little or no a priori knowledge about the variance of individual criterion over the feasible set or the convariance (interdepndence) among different criteria. He may assign high priority to two of the criteria to make sure that they attain acceptable values. If the two criteria turn out to be highly correlated, his preference judgment should reveal that one of the criteria could have been assigned a lower priority. Moreover, in a repeated choice situation, as is typical of iterative procedures in preference programming, it is not uncommon for a DM to be inconsistent by changing his mind as he learns more about the alternatives. For example, given two initial solutions, he may prefer the one that exaggerates his priorities with the hope of achieving further improvement. As the iterative process evolves he learns that those are actually the most attractive alternatives. So he may end up choosing the less radical solution. For a discussion of the adaptive displacement of preferences, the reader is referred to [19]. For these reasons, if any operational method to solve MCLP by evaluation of the DM's holistic preference is to be robust, it must allow the DM to uncover inconsistency in his judgment and make appropriate adjustments. This way the experience of solving an MCLP can be interpreted as a learning process for the DM. As he accrues information about his alternatives, he may refine his preference and priority judgment, not unlike acquiring skill in playing a game.

The holistic preference evaluation (HOPE) procedure to be developed in this paper assumes (i) and (ii) only in an operational sense. As long as the DM believes that he can perform these tasks, HOPE may be used. Of course, he is eventually expected to settle for a consistent value judgment in order for the solution thus obtained to be meaningful.

Next we consider scaling.

Definition. An MCLP with the criteria $(c_1,\ldots,c_K)$ ordered in descending priority (importance) is said to be properly scaled if the optimal solution x^* maximizes

$$\sum_{k=1}^{K} \lambda_k c_k x \text{ over } X$$

for some λ_k satisfying

$$\lambda_k > 0,\ k=1,\ldots,K,$$

$$\sum_{k=1}^{K} \lambda_k = 1; \text{ and}$$

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_K.$$

Again, we assume proper scaling operationally. If it holds, HOPE proceeds to determine $\{\lambda_k\}$ and hence x^*. When it fails, the DM's holistic preference should reveal that it is the case and indicate the proper directions for rescaling. In general, $\{\lambda_k\}$ is not unique and actually provides a sufficient degree of freedom so that x^* can be determined by HOPE over a reasonably wide range of scaling.

4. Holistic Preference Evaluation (HOPE): An Algorithm

Given an MCLP it is assumed that

(I) the DM's holistic preference can be assessed;

(II) the DM can learn to be consistent; and

(III) the DM can learn to discover improper scaling.

To solve MCLP, it suffices to determine a set of weights $\{\lambda^*_k\}$ that correspond to the optimal solution x^*. Let $(c_1,\ldots c_K)$ be the criteria ordered in descending priority. Then proper scaling implies the existence of $\{\lambda^*_k\}$ such that

(3) $$\lambda_1^* \geq \lambda_2^* \geq \cdots \geq \lambda_K^* > 0 \quad \text{and}$$

(4) $$\sum_{k=1}^{K} \lambda_k^* = 1.$$

The HOPE algorithm determines successively $\lambda_K^*, \lambda_{K-1}^*, \ldots, \lambda_1^*$, in that order. Each iteration of the algorithm involves a number of LP's defined on X with parametric objective functions based on $(c_1,\ldots,c_K)$. The parametric solutions generated at each iteration are presented to the DM who then selects the one he prefers most. This choice will be used to define the set of parametrizations in the following iteration. The algorithm is centered around the idea of probing the distribution of weights while enforcing conditions (3) and (4). Initially, all criteria are divided into two groups: high priority and low priority. For each possible division, efficient solutions are generated by parametrizing a complementary pair of high and low priority weights assigned to each criterion in the corresponding groups. Since only one parameter is involved, standard parametric LP techniques can be applied. The DM is asked to choose the most preferred solution from this first and probably rather crude approximation. Next, the high priority weights of this solution are temporarily fixed while the above process is applied to refine the low priority weights until the lowest priority weight is determined. The latter is then fixed and the algorithm is repeated with one less weight to be resolved.

In each parametrization, the criteria whose weights are being parameterized are called active. Their indices will be consecutive, say i, i+1,...,k. The weights $\lambda_1,\ldots,\lambda_{i-1}$ will be temporarily fixed at values assigned in previous iterations. The weights $\lambda_{k+1},\ldots,\lambda_K$ will be permanently fixed at $\lambda^*_{k+1},\ldots,\lambda^*_K$ since they have already been determined. The active criteria are partitioned into two contiguous groups: the head and the tail. For example, if $(c_i,c_{i+1},\ldots,c_j)$ is the head, then $(c_{j+1},\ldots,c_k)$ is the tail. The head is the group with higher priorities. c_j will be called the vedette. It identifies the head-tail partitioning. Every criterion in the head is assigned equal weight: λ_h, and every criterion in the tail: λ_t. Bounds for λ_h and λ_t are determined in the previous iteration as $\bar{\lambda}_h$ and $\underline{\lambda}_t$ respectively. The parametrization involves decreasing λ_h while increasing λ_t until they are equal. It is identified as

$$P[k,1,2,\ldots,i-1,j]$$

meaning that

(i) λ_k^* is to be determined next;

(ii) $\lambda_1,\lambda_2,\ldots,\lambda_{i-1}$ are temporarily fixed;

(iii) $\lambda_{k+1}^*,\ldots,\lambda_K^*$ have been determined and hence fixed;

(iv) c_j is the vedette;

(v) $(c_i,c_{i+1},\ldots,c_j)$ is the head; and

(vi) $(c_{j+1},\ldots,c_k)$ is the tail.

Figure 1 gives a schematic representation of a parametrization. The parametric objective function for the LP is

$$c(\beta) = \sum_{\ell=1}^{i-1} \bar{\lambda}_\ell c_\ell + \sum_{\ell=i}^{j} (\bar{\lambda}_h-\beta)c_\ell + \sum_{\ell=j+1}^{k} (\underline{\lambda}_t+\beta)c_\ell + \sum_{\ell=k+1}^{K} \lambda_\ell^* c_\ell \tag{5}$$

$$0\le\beta\le\bar{\beta}$$

where

$\bar{\lambda}_\ell$, $\ell=1,\ldots,i-1$, are computed in previous iterations;

λ_ℓ^*, $\ell=k+1,\ldots,K$, are computed in previous iterations;

$\underline{\lambda}_t$ is the lower bound for the weight on the tail;

$\bar{\lambda}_h$ is the upper bound for the weight on the head;

$\lambda_t=\underline{\lambda}_t+\beta$ is the parametrized weight on the tail;

$\lambda_h=\bar{\lambda}_h-\beta$ is the parametrized weight on the head; and

$[0,\bar{\beta}]$ is the range of the parametrization.

The bounds and range in $c(\beta)$ are computed as follows.

Let $$w_1 = [1-\sum_{\ell=1}^{i-1} \bar{\lambda}_\ell - \sum_{\ell=j+1}^{k} \lambda_{k+1}^* - \sum_{\ell=k+1}^{K} \lambda_\ell^*]/j-i+1$$

(6)

$$w_2 = [1-\sum_{\ell=1}^{i-1} \bar{\lambda}_\ell - \sum_{\ell=i}^{j} \bar{\lambda}_{i-1} - \sum_{\ell=k+1}^{K} \lambda_\ell^*]/k-j.$$

If $\quad w_1 \leq \bar{\lambda}_{i-1}$,

$$\text{then} \begin{cases} \underline{\lambda}_t = \lambda^*_{k+1} \\ \bar{\lambda}_h = w_1 \end{cases} ;$$

$$\text{otherwise} \begin{cases} \underline{\lambda}_t = w_2 \\ \bar{\lambda}_h = \bar{\lambda}_{i-1} \end{cases} .$$

The two cases are necessary to ensure that condition (3) holds. In either case, $\bar{\beta}$ is given by

$$\bar{\beta} = (\bar{\lambda}_h - \underline{\lambda}_t)/2.$$

The parametric LP for P[k,1,2,...,i-1,j] is then

$$(7) \qquad \underset{x \in X}{\text{maximize}}\ c(\beta),\ 0 \leq \beta \leq \bar{\beta} .$$

The optimal solution to (7) is piecewise constant over intervals of β in the range $[0,\bar{\beta}]$, and can be solved by standard parametric LP methods [4].

At iteration n, if λ_k is to be determined next, p_n parametric LP's of the form (7) will be considered. Depending on the outcome of iteration n-1, p_n may vary from 1 to k-1. Solutions from these p_n problems are presented to the DM who must then identify his most preferred solution in the set. As this solution corresponds to an interval of value for the parameter β, the DM may decide on the particular value of β^* within this interval that will be used in iteration n+1. If he has no particular preference, the midpoint of the interval will be used. In addition, his most preferred solution in iteration n may appear in more than one parametrization. In this case, the DM should exercise his holistic preference judgment to choose the preferred configuration as well. Otherwise, his previous decisions may be examined to infer a choice. If none is available, the algorithm will choose the parametrization with the most even distribution of weights. This is a logical choice as the absense of preference implies that the DM's priorities cannot be very distinct. In any case, at the end of iteration n, a β^* is determined. If the corresponding parametrization has a single criterion in the tail, namely c_k, then λ_k^* is determined by

$$(8) \qquad \lambda_k^* = \underline{\lambda}_t + \beta^* .$$

Otherwise, each c_ℓ in the head will be assigned the weight

$$(9) \qquad \bar{\lambda}_\ell = \bar{\lambda}_h - \beta^*$$

which will then be temporarily fixed in iteration n+1.

The algorithm can now be stated.

The HOPE Algorithm.

Step 0. Initialize: set $k=K$, $n=0$, $\lambda^*_{K+1}=0$.

Step 1. Free all undetermined weights: set $i=1$.

Step 2. Update iteration count: set $n=n+1$, number of parametrizations $p_n=k-i$. Initialize: set $j=i-1$, $S=\emptyset$.

Step 3. Choose vedette: set $j=j+1$. If $j=k$, go to Step 5.

Step 4. Solve parametric linear program (7) for $P[k,1,2,\ldots,i-1,j]$
Enter solutions in S.
Return to Step 3.

Step 5. Elicit DM's holistic preference: select most preferred solution x_n in S and corresponding value of parameter β^*.
Also, verify scaling and priorities.

Step 6. Analyze x_n: if only one criterion in tail, go to Step 7.
Otherwise, set $\bar{\lambda}_\ell=\bar{\lambda}_h-\beta^*$ for each of the h criteria in head ($h=j-i+1$). Update number of weights to be temporarily fixed: set $i=i+h$. Return to Step 2.

Step 7. λ^*_k is determined: set $\lambda^*_k=\underline{\lambda}_t+\theta^*$. Set $k=k-1$. If $k=1$, stop.
Otherwise, return to Step 1.

A flow diagram for the HOPE algorithm is given in Figure 2. An illustration of all possible outcomes for $K=4$ is given in Figure 3.

The number of iterations required by the algorithm is bounded by

$$n_{\max} = \sum_{j=1}^{K-1} j \, .$$

The total number of parametrizations examined is bounded by

$$(\Sigma p_n)_{\max} = \sum_{j=1}^{K-1} j(K-j). \tag{11}$$

The actual number of parametrizations required is usually much less and can be made so, especially when K is large (say, K>5) by the following consideration. For perfect generality, the algorithm is stated in such a way that each time it returns from Step 7 to Step 1, all possible distributions for the undetermined weights are considered to be of potential interest. In practice, outcomes in previous iterations can usually be used to rule out further considerations of various parametrizations. For example, referring to Figure 3, suppose P[42] and P[43] produce solutions that are significantly inferior to P[41]. Then, after λ_4 is determined, P[32] is extremely unlikely to produce attractive solutions and may therefore be suppressed.

Finally, a discussion of scaling and priority checks will complete the description of the HOPE algorithm. Whenever the DM has reasons to suspect that the values of certain criteria are consistently too high or too low, a scaling and priority check should be signalled. This happens if the DM's preferences seem to lie beyond the range $[0,\bar{\beta}]$ for the parameter β in all parametrizations in an iteration. If a pairwise priority interchange can be identified and approved by the DM, it should be executed and the algorithm restarted. Otherwise, a unilateral scaling will be performed on specified criteria. Scaling is recommended only when order of magnitude changes deem necessary.

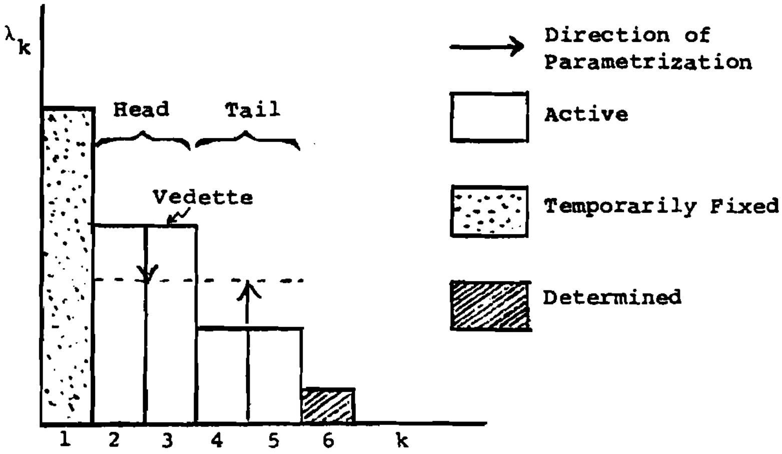


Figure 1. A typical parametrization in HOPE .

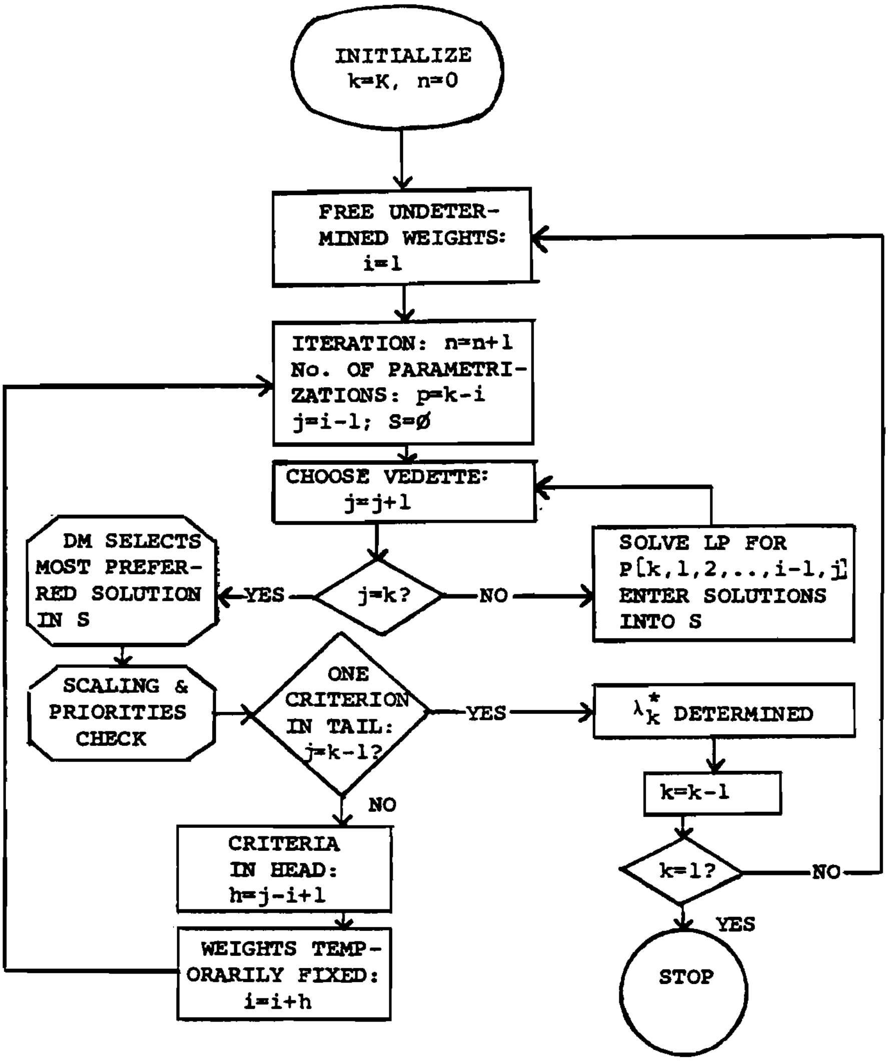

INITIALIZE
k=K, n=0
FREE UNDETERMINED WEIGHTS: i=1
ITERATION: n=n+1
No. OF PARAMETRIZATIONS: p=k-i
j=i-1; S=Ø
CHOOSE VEDETTE: j=j+1
DM SELECTS MOST PREFERRED SOLUTION IN S
YES
j=k?
NO
SOLVE LP FOR P[k,1,2,...,i-1,j]
ENTER SOLUTIONS INTO S
SCALING & PRIORITIES CHECK
ONE CRITERION IN TAIL: j=k-1?
YES
λ_k^* DETERMINED
NO
k=k-1
CRITERIA IN HEAD: h=j-i+1
k=1?
NO
YES
WEIGHTS TEMPORARILY FIXED: i=i+h
STOP

Figure 2. Flow Diagram for HOPE.

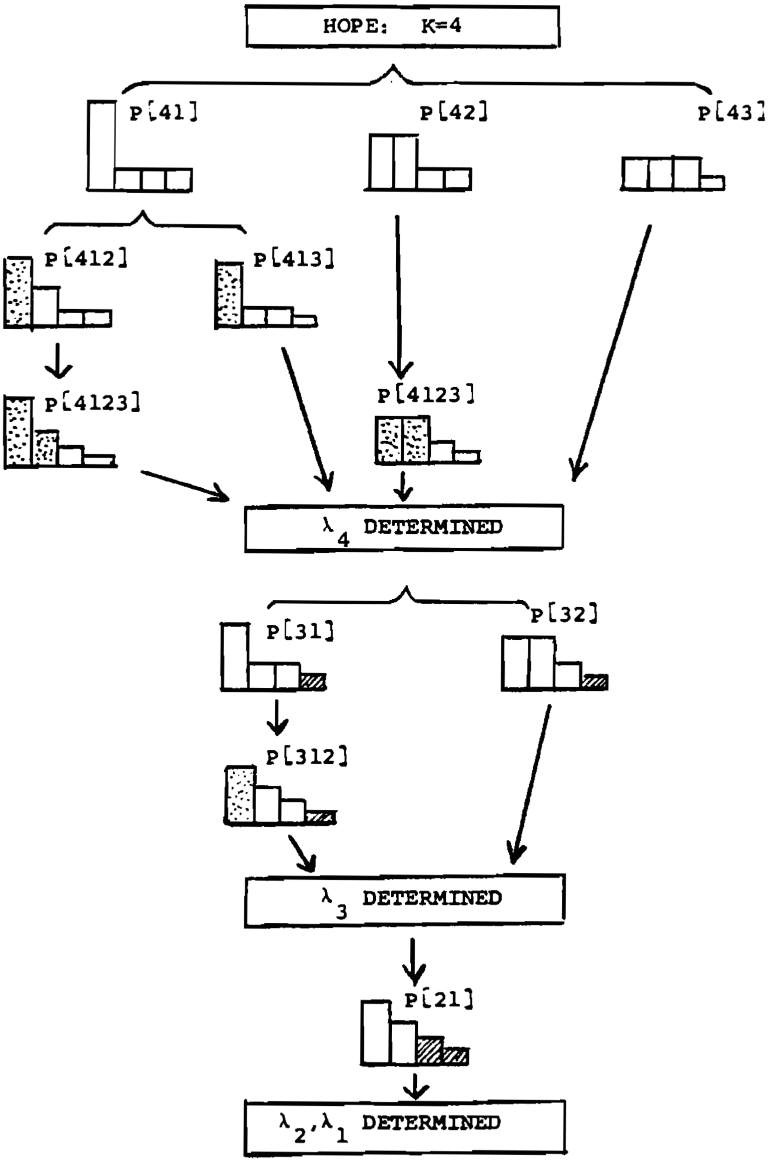


Figure 3. HOPE illustrated for K=4.

5. Justification of HOPE

The HOPE algorithm is a finite procedure to parametrize conditions (3) and (4) in order to discover the optimal solution to MCLP, defined as the one most preferred by the DM. As pointed out in [9], the exact, full parametrization of all possible combinations of the weights becomes very difficult for K>2. HOPE is essentially a method of nested bicriterion programming that allows the successive refinement of crude initial approximations. The main argument for the robustness of HOPE relies on the fact that the weights λ^* are in general not unique, regardless of whether the corresponding x^* is the unique optimal solution to MCLP or not. This can be seen from the following propositions. Let

$$e = (1,1,\ldots,1)\varepsilon R^K,\ y\varepsilon R^K, C = \begin{bmatrix} c_1 \\ \vdots \\ c_K \end{bmatrix} \varepsilon R^K x R^n.$$

Proposition 3. $x^0 \varepsilon E$ if and only if the LP

(12)
$$\begin{aligned} \text{maximize}\quad & e\,y \\ \text{subject to}\quad & I\,y - Cx = -Cx^0 \\ & Ax = b \\ & y,\quad x \geq 0 \end{aligned}$$

has an optimal solution with y=0.

Proof. Follows from the definition of efficiency.

Proposition 4. Let (λ^0,π^0) be a dual optimal solution to (12). Then $\lambda^0 > 0$ and x_0 solves

(16)
$$\underset{x\varepsilon X}{\text{maximize}}\ \lambda^0 Cx$$

Proof. Dual optimality of $(\lambda^0, \pi^0) \Rightarrow$

(13) $e-\lambda^0 \leq 0$

(14) $\lambda^0 C-\pi^0 A \leq 0$

(15) $-\lambda^0 Cx^0+\pi^0 b = \max ey = 0$

Now x^0 is primal feasible to (16) by definition. π^0 is dual feasible to (16) by (14). By (15), complementary slackness holds. Therefore, (x^0, π^0) is an optimal primal-dual pair of solution to (16). And (13) implies $\lambda^0 > 0$.

Hence, λ^0 from (12) may be used to characterize $x^0 \epsilon E$. Now consider x^* and (12) with $x^0 = x^*$. $x^* \epsilon E$ implies $y=0$ and the optimal basis in (12) will in general be degenerate. This is certainly true if, for example, x^* is an extreme point of X. Consequently, there exists in general a multitude of (linearly independent) λ^* that satisfy Proposition 2 for x^* as well as condition (4). Call this set Λ. Let Λ_K be its restriction to λ_K. Recalling that c_K has lowest priority, λ_K is expected to be small (<<1) so that relatively large perturbations would still be insignificant. This implies that even crude approximationsof Λ should intersect Λ_K. The HOPE algorithm does exactly that. The series of P[K...] parametrizations seek $\lambda_K \epsilon \Lambda_K$. Once this holds, the above argument can be repeated inductively. Note that as the relative margin of error decreases for the higher priority weights, the precision of the parametric approximation increases. For instance, once $\lambda_3, \ldots, \lambda_K \epsilon \Lambda$ have been determined, P[21] determines exactly the corresponding λ_2 and λ_1.

Another argument for the robustness of HOPE is that no special assumption about the underlying utility function is made. Of course, caution must be exercised in the general case to take into account of nonextremal as well as local optima. In the first instance, all optimal solutionsto (16) for a given λ^0 should be examined. Similarly, for non-unimodal utility, the DM can choose several solutions at any stage of HOPE and branch out the refinement procedure for local optimal solutions.

In terms of implementation, HOPE involves simply the iterative application of parametric linear programming. It can therefore be incorporated as a natural extension of the algorithmic tools of well developed LP technology. To the DM the basic concepts of HOPE are easy to understand and maybe even to accept. In actual use, the DM has only to examine efficient solutions. Moreover, starting from iteration 1 on the DM is offered a holistic view of the alternatives which becomes more and more clear as the process evolves. This is in contrast to most preference programming methods that rely on local or marginal utility analysis.

Finally, it should be remarked that the primary purpose of HOPE is to identify the optimal solution x^* to an MCLP. This is done by approximating the weights $\{\lambda_k^*\}$ that characterize x^*. However, $\{\lambda_k^*\}$ are not meant to be an evaluation of the DM's utility function (except in the special case where it is known to be linear[1]).

[1] This special case is exploited in the numerical examples in Section 6 for the sole purpose of simplifying the simulation of the DM's response. The reader should not be confused about the significance of the utility function.

This function u is in general too complicated to be meaningfully represented by $\Sigma \lambda_k^* c_k$. In practice, apart from assuming its separable additivity and monotonicity, this author prefers to leave u out of the picture. Nonetheless, the DM may still attach whatever intuitive interpretation he chooses to λ^*. Thus HOPE can be regarded as a learning process for the DM to "weigh" his criteria. Or if one decides on using λ^* as a measure of the DM's holistic preference then HOPE is truly a procedure for holistic preference evaluation.

6. Numerical Examples

In this section, the results of the application of HOPE to four test problems are reported. Although they are not based on experience involving decision makers in actual applications, they should still be very useful as a demonstration of the efficacy of the algorithm. This is especially true since the last three problems are drawn from real-life multiple criteria decision processes reported in the literature. Problem I is the simple numerical example used by Zionts and Wallenius in [20]. Problem II is the academic department planning model formulated by Geoffrion, Dyer and Feinberg in [11]. As [11] did not provide sufficient data to reconstruct the problem therein, fictitious but realistic values of the parameters are used here. These are recorded in Appendix A. Problems III and IV are two cases of the forest management model studied by Steuer and Schuler in [16]. The data are given in Appendix B. Each of the four problems makes a particular point about HOPE and together they provide considerable insight into the approach.

As the algorithm has not yet been implemented as a fully automatic and interactive computer program, the tests were run by batching each parametric LP as a separate job on a CDC 7600 at Brookhaven National Laboratory. The LP code used was CDC'c APEX III with parametric options.

To simplify the test runs and to ensure their reproducibility, a linear utility function is specified in each case to simulate preference judgment by a DM. The reader is reminded that linearity assumptions are not necessary in practice.

Problem I. Example in [20].

$$K = 3,\ x \epsilon R^6,\ A \epsilon R^2 \times R^6$$

$$c_1 = (3,\ 1,\ 2,\ 1,\ 0,\ 0)$$

$$c_2 = (1,\ -1,\ 2,\ 4,\ 0,\ 0)$$

$$c_3 = (-1,\ 5,\ 1,\ 2,\ 0,\ 0)$$

$$A = \begin{bmatrix} 2 & 1 & 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 2 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 60 \\ 60 \end{bmatrix}$$

The entire set of extreme point solutions are listed in Table 1. The ones in parenthesis are inefficient. As in [20], it is assumed that the DM's (implicit) utility function is

$$u = 0{\cdot}58c_1 + 0{\cdot}21c_2 + 0{\cdot}21c_3$$

which is maximal at solution B with a value of 42·96. Applying HOPE, the DM first ranks (c_1, c_2, c_3) in descending order of priority. The results are summarized in Table 2. Note that the solution on each line is achieved by the value of β on that line up to but excluding the value of β on the following line, The first iteration involves parametrizations P[31] and P[32]. The DM's preferred solution appears in both cases. His holistic preference implies that c_2 and c_3 are considered equal and less important than c_1.

Therefore, he chooses solution B in P[31] and $\beta^{*}= 0{\cdot}21$, the midpoint of the interval corresponding to B. Next, P[312] is considered. As no improvement results from shifting weight from c_3 to c_2, the DM concludes that c_2 and c_3 should have equal weight and chooses $\lambda_3^{*} = \beta^{*} = 0{\cdot}21$. P[21] provides a final check by shifting weight from c_1 to c_2 for possible improvement. None results, and so $\beta^{*}=0.0$ $\lambda_2^{*} = 0{\cdot}21$ and $\lambda_1^{*} = 0{\cdot}58$.

This simple example illustrates the fundamental concept of HOPE. Because there are so few alternatives, diverse combinations of weights give rise to the samesolution. Identification of the most preferred solution in an iteration does not suffice. Priorities and hence holistic preference must be called into play. In this case, priorities actually play the major role. In complex problems with abundant alternatives the β intervals will diminish and the effect of preferences will become more significant.

TABLE 1. PROBLEM I: All extreme point solutions

Solution	x_1	x_2	x_3	x_4	x_5	x_6	c_1	c_2	c_3
A	18	0	6	0	0	0	66	30	-12
B	12	0	0	12	0	0	48	60	12
C	0	12	12	0	0	0	36	12	72
D	0	6	0	18	0	0	24	66	66
(E)	20	0	0	0	20	0	60	20	-20
F	0	15	0	0	45	0	15	-15	75
(G)	0	0	15	0	0	45	30	30	15
H	0	0	0	20	0	20	20	80	40
(I)	0	0	0	0	60	60	0	0	0

TABLE 2. PROBLEM I: Solution by HOPE

n	P	β	Solution	u	β^*	⇒
1	[31]	0.00	A	42.06		
		0.20	B	42.96	→ 0.21	
		0.22	D	41.64		
		0.33	"	"		
	[32]	0.00	B	42.96		
		0.13	H	36.80		
		0.16	D	41.64		
		0.33	"	"		
2	[312]	0.00	B	42.96		
		0.21	"	"	→ 0.21	λ_3=0.21
3	[21]	0.00	B	42.96	→ 0.00	λ_2=0.21
		0.04	D	41.64		λ_1=0.58
		0.19	"	"		
						x^*=B

Problem II. Academic Department Planning Model in [11].

$$K = 6,\ x \in R^{24},\ A \in R^{7} \times R^{24}$$

This is a planning problem for the operation of a single academic department on a large university campus. The constraints reflect work balance, budget balance, man-power ceiling, policies and commitments of the department. The criteria have the following meaning.

f_1: course sections offered - graduate division,

f_2: course sections offered - lower division,

f_3: course sections offered - upper division,

f_4: teaching assistant time used for support,

f_5: releases for departmental service duty,

f_6: additional activities of the regular faculty.

DM's priorities: $(c_1, c_2, c_3, c_4, c_5, c_6) = (f_5, f_4, f_6, f_1, f_3, f_2)$

DM's utility function:

$$u = 0.4c_1 + 0.3c_2 + 0.2c_3 + 0.07c_4 + 0.02c_5 + 0.01c_6$$

which is maximized at $v^* = (68.25, 20.0, 1.08, 100, 30, 20)$ with the value $u(v^*) = 41.3$.

Table 3 contains the solutions to Problem II examined by HOPE. To simplify presentation, only parametrizations giving the preferred solution in each iteration have been entered in Table 4. The optimal v^* is determined correctly by HOPE in 7 iterations with the weights

$\wedge$ = (0.395, 0.295, 0.185, 0.085, 0.025, 0.015) which is a very good approximation of the implicitly assumed linear utility function u.[1] This example illustrates that even with six criteria, the number of iterations required may still be relatively low. The upper bound for K=6 is 15.

[1] See remark at end of Section 5.

TABLE 3. PROBLEM II: Solutions examined

v	$c_1=f_5$	$c_2=f_4$	$c_3=f_6$	$c_4=f_1$	$c_5=f_3$	$c_6=f_2$	u
A	68.3	20.0	1.08	100	30.0	20.0	41.3
B	77.0	3.2	1.67	"	"	"	39.9
C	53.8	32.6	0.12	"	"	32.6	39.3
D	52.0	32.4	0.0	"	"	36.3	38.5
E	"	20.0	"	131	"	20.0	36.8
F	"	"	"	100	61.2	"	35.2
G	"	"	"	"	30.0	51.2	34.9
H	24.0	"	"	159	"	20.0	27.5
I	"	"	"	100	"	79.2	24.0
J	0.0	"	"	183	"	20.0	19.6
K	"	"	"	100	113.2	"	15.5
L	"	0.0	"	203	30.0	"	15.0
M	"	"	"	100	133.2	"	9.9

TABLE 4. PROBLEM II: Solution by HOPE

n	P	β	v	u	β^*	⇒
1	[63]	0.00	A	41.3	→ 0.03	
		0.06	C	39.3		
		0.12	D	38.5		
		0.15	G	34.9		
		0.17	I	24.0		
2	[61235]	0.00	A	41.3	→ 0.015	λ_6=0.015
		0.03	"	"		
3	[53]	0.00	A	41.3	→ 0.06	
		0.12	E	36.8		
		0.18	"	"		
4	[51234]	0.00	A	41.3	→ 0.01	
		0.02	E	36.8		λ_5=0.025
		0.06	E	"		
5	[43]	0.00	A	41.3	→ 0.06	
		0.12	E	36.8		λ_4=0.085
		0.21	"	"		
6	[32]	0.00	A	41.3	→ 0.10	
		0.20	"	"		λ_3=0.185
7	[21]	0.00	B	39.9		
		0.06	A	41.3	→ 0.11	λ_2=0.295
		0.16	"	"		λ_1=0.395
						v^*=A

Problem III. Forest Management Model in [16].

$$K = 5, x = R^{31}, A \in R^{13} \times R^{31}$$

The problem is to optimize management plans for the multitude of goods and services obtainable from public forest land. There are eight acreage limitation equality constraints, one budget limitation inequality constraint and four sustaining timber yield inequality constraints in the model. The criteria represent activity levels in

timber production (z_1),
dispersed recreation (z_2),
hunting forest species (z_3),
hunting open land species (z_4), and
grazing (z_5).

As reported in [16], the real DM in this case ranked the criteria (z_2, z_3, z_4, z_1, z_5) in descending order. The method in [16] led to the determination of solution J in Table 5[1] as the optimal solution. Applying HOPE with the above priority ranking of the criteria, the DM will discover that the value of z_3 never exceeds that in solution N. If he switches the priorities of z_2 and z_3 at any stage of HOPE (even down to P[21]) and continues, he can still discover solution J. However, we present the results of a complete run of HOPE after the switch

[1] There are slight discrepencies between the numerical values in [16] and those in Table 5. This is caused by the fact that we started with data presented in the Appendix of [16] which have been rounded off or truncated to two decimal places.

The DM's new priorities:

$$(c_1, c_2, c_3, c_4, c_5) = (z_3, z_2, z_4, z_1, z_5)$$

Solution J corresponds (among other possibilities) to the utility function

$$u = 0.50c_1 + 0.25c_2 + 0.12c_3 + 0.08c_4 + 0.05c_5.$$

which is maximized at a value of 19735 by solution J. HOPE establishes the optimality of J in 10 iterations with the weights λ = (0.53, 0.23, 0.13, 0.08, 0.03).

This example illustrates how HOPE can be used to discover inconsistency in the DM's holistic preferences and how the DM can regard HOPE as a learning process to evaluate his own value judgment.

TABLE 5. PROBLEM III: Solutions examined

v	$c_1=z_3$	$c_2=z_2$	$c_3=z_4$	$c_4=z_1$	$c_5=z_5$
A	18098	23178	2328	24269	909
B	"	28282	4880	"	1249
C	18091	28006	"	27356	"
D	18078	27849	"	29187	"
E	18002	26961	"	39500	"
F	17989	27004	4901	"	1275
G	17615	30058	5730	22338	2270
H	17596	29953	"	23950	"
I	17578	29940	"	24258	"
J	17338	28419	"	39500	"
K	17337	28420	"	"	"
L	17336	"	5732	"	2273
M	17334	28495	5730	38909	2270
N	17100	31121	6506	14881	3318
O	17069	30428	6401	23948	3177
P	17047	31333	6506	8662	3318
Q	17032	31350	"	8230	"
R	17025	31359	"	8017	"
S	17023	28623	6119	"	2795
T	16992	30533	6506	23477	3318
U	16963	30514	"	23942	"
V	16936	30507	"	24241	"
W	16696	29641	"	33457	"
X	16693	29560	"	34098	"
Y	16631	28939	"	39500	"
Z	16600	28984	"	"	"

TABLE 6. PROBLEM III: Solution by HOPE

n	P	s	u	s^*	⇒
1	[51]	0.09 0.11 0.13	19558 19735 19734.9	→ 0.12	
2	[512]	0.03 0.04 0.12	19004 19735 "	→ 0.08	
3	[5123]	0.02 0.03 0.08	19004 19735 "	→ 0.06	
4	[51234]	0.00 0.06	19735 "	→ 0.03	$\lambda_5=0.03$
5	[41]	0.08 0.11 0.15	19558 19735 19734.9	→ 0.13	
6	[412]	0.02 0.03 0.13	19734.8 19735 "	→ 0.08	
7	[4123]	0.00 0.03 0.08	19734.9 19735 "	→ 0.05	$\lambda_4=0.08$
8	[31]	0.08 0.12	19735 19734.9	→ 0.10	
9	[312]	0.00 0.10	19735 "	→ 0.05	$\lambda_3=0.13$
10	[21]	0.00 0.04 0.16	19558 19735 19734.9	→ 0.10	$\lambda_2=0.23$ $\lambda_1=0.53$
					$v^*=J$

Problem IV. Forest Management Model in [16].

This is the same as Problem III with a different linear utility function to simulate the DM's preferences. In each of the first three problems, the optimal solution actually appears in the results of the first iteration. Subsequent iterations serve primarily as a verification that no improvement can be made. This is typical when the MCLP is not very complex and the DM's utility function is linear. Problem IV illustrates that even in the linear case, it may require more than one iteration to uncover the optimal solution.

DM's utility function:

$$u = 0.45c_1 + 0.45c_2 + 0.045c_3 + 0.045c_4 + 0.01c_5$$

which has a maximum at 22771.

HOPE generates the optimal solution in iteration 2 and establishes its optimality in six iterations. The process is summarized in Table 7.

TABLE 7. PROBLEM IV: Solution by HOPE

n	P	β	u	β^*	⇒
1	[52]	0.013	22677		
		0.035	22769	→ 0.04	
		0.045	22768		
2	[5124]	0.015	22770		
		0.020	22771	→ 0.02	λ_5=0.02
		0.029	22769		
3	[42]	0.015	22769		
		0.022	22771	→ 0.026	
		0.030	22770		
4	[4123]	0.014	22769		
		0.023	22771	→ 0.025	λ_4=0.045
		0.026	"		
5	[32]	0.000	22771	→ 0.006	
		0.012	22769		λ_3=0.051
		0.019	22768		
6	[21]	0.319	22756		
		0.362	22771	→ 0.377	λ_2=0.428
		0.391	"		λ_1=0.456

7. Conclusions

In this paper, a parametric linear programming method is proposed to solve the multiple criteria optimization problem. The approach uses the decision maker's preference judgement as well as his priority ranking of the criteria. Based on heuristic arguments and empirical evidence, the algorithm is observed to be robust in terms of

i) implementation: requires only parametric LP software;

ii) user friendliness: easy to understand, requires only multiple choice response;

iii) general applicability: requires no special assumptions about the DM's utility function;

iv) intuitive appeal: may be interpreted as holistic preference evaluation, or a learning process in "weighing" the criteria.

Further development involves:

i) implementation as an extension of the capability of existing algorithmic tools in LP;

ii) experimentation in diverse, real decision processes, e.g. energy policy analysis, where the conflicting criteria may be costs , resource depletion, environmental impact, nuclear proliferation, etc.;

iii) generalization to nonlinear criteria, e.g. concave objective functions, using results in [9]; and

iv) comparison with other methods [17].

Appendix A

Data for Problem II in MPS format.

The model is described in [11]. The values of the parameters used in Problem II are tabulated as follows.

j	y_{1j}	t_j	c_j	a_j	s_j	r_j	g_j	b_j	m_{1j}
1	10	4	10	0	1	1	8	3.5	100
2	15	5	10	1	1	2	8	2.4	10
3	20	2	10	0	0	0	8	0.6	30
4	5	8	4	0	0	0	8	2.0	-
5	3	8	2	0	0	0	8	2.6	-

The bounds implied by (15) in [11] are dropped. Note also that the summation in inequalities (9) and (10) in [11] should be over $1 \leq k \leq 5$. In the following, Ri is the i^{th} constraint.

```
NAME          PROBLEM II
ROWS
 N  F1
 N  F2
 N  F3
 N  F4
 N  F5
 N  F6
 E  R1
 L  R2
 L  R3
 L  R4
 L  R5
 L  R6
 G  R7
COLUMNS
    X11       R1          1.0          F1          1.0
    X12       R1          1.0          F2          1.0
    X12       R7          1.0
    X13       R1          1.0          F3          1.0
    X21       R1         -4.0          R2          1.0
    X21       R3          3.5          R4         -0.5
    X21       F6          0.25
    X22       R1          -5.0       R2          1.0
    X22       R3          2.4        R5          -0.625
    X22       F6          0.333333
    X23       R1          -2.0       R2          1.0
    X23       R3          0.6        R6          -0.25
    X23       R7          -2.0
    X24       R1          -8.0       R2          1.0
    X24       R3          2.0
    X25       R1          -8.0       R2          1.0
    X25       R3           2.6
    X31       R1          -4.0
    X31       R2           1.0       R3          3.5
    X32       R2           1.0       R3          2.4
    X32       R1          -5.0
    X33       R1          -2.0       R2          1.0
    X33       R3          0.6        R7          -2.0
    X34       R1          -8.        R2           1.
    X34       R3          2.
    X35       R1          -8.        R2          1.
    X35       R3          2.6
    X41       R1          8.         R4          1.
    X41       F5          8.
    X42       R1          8.         R5          1.
    X42       F5          8.
    X43       R1          8.         R6          1.
    X43       F4          8.
    X44       R1          8.
    X45       R1          8.
RHS
    RHS       F6          4.916667
    RHS       R1          120.       R2          21.
    RHS       R3          33.3       R4          3.5
    RHS       R5          3.         R6          2.5
    RHS       R7          20.
BOUNDS
 LO B1        X11         100.
 LO B1        X12         10.
 LO B1        X13         30.
 FX B1        X21         0.
 UP B1        X22         5.
 UP B1        X23         10.
 UP B1        X24         1.
 UP B1        X25         1.
ENDATA
```

Appendix B

Data for Problem III in MPS format

```
NAME          PROBLEM III
ROWS
 N  Z1
 N  Z2
 N  Z3
 N  Z4
 N  Z5
 L  R1
 L  R2
 L  R3
 L  R4
 L  R5
 L  R6
 L  R7
 L  R8
 L  R9
 L  R10
 L  R11
 L  R12
 DL R13       Z1           1.0
COLUMNS
    X1        Z1           13.72       Z2              2.0
    X1        Z3           3.0         R1              1.0
    X1        R9           3.06        R10             7.94
    X2        Z3           2.4         R1              1.0
    X2        Z1           10.23       Z2             2.0
    X2        R9           1.53        R10            5.96
    X3        Z1           8.21        Z2             2.0
    X3        Z3           2.1         R1             1.0
    X3        R9           0.6         R10            4.76
    X4        Z1           6.73        Z2             2.0
    X4        Z3           2.3         R1             1.0
    X4        R9           1.38        R10            3.97
    X5        Z2           2.5         Z3             2.1
    X5        R1           1.0         R9             0.35
    X6        Z2           2.5         Z3             1.8
    X6        R1           1.0
    X7        Z1           12.30       Z2             2.0
    X7        Z3           4.0         R2             1.0
    X7        R9           3.08        R11            7.10
    X8        Z1           9.80        Z2             2.0
    X8        Z3           3.2         R2             1.0
    X8        R9           1.54        R11            5.68
    X9        Z1           8.59        Z2             2.0
    X9        Z3           2.8         R2             1.0
    X9        R9           0.6         R11            4.97
    X10       Z1           6.05        Z2             2.0
    X10       Z3           3.0         R2             1.0
    X10       R9           1.38        R11            3.55
    X11       Z2           2.5         Z3             2.8
    X11       R2           1.0         R9              .35
    X12       Z2           2.5         Z3             2.4
    X12       R2           1.0
    X13       Z1          4.08        Z3             1.0
    X13       R3          1.0         R9             5.05
    X14       Z1          2.33        Z3             0.8
    X14       R3          1.0         R9             2.54
    X15       Z1          2.22        Z3             0.8
    X15       R3          1.0         R9             0.6
    X16       Z1          0.57        Z3             0.8
    X16       R3          1.0         R9             2.16
    X17       Z3          0.8         R3             1.0
    X17       R9          0.35
    X18       Z3           0.7        R3             1.0
    X19       Z2          4.0         Z3             4.0
    X19       Z5          0.7         R4             1.0
    X19       R9          1.0
    X20       Z2          2.0         Z3             2.0
    X20       Z5          0.17        R4             1.0
    X21       Z1          9.32        Z2             2.0
    X21       Z3           1.0        R5             1.0
    X21       R9           3.02       R12            5.6
    X22       Z1           6.98       Z2             2.0
    X22       Z3           0.8        R5             1.0
    X22       R9           1.51       R12             4.2
    X23       Z1           5.81       Z2             2.0
    X23       Z3            0.8       R5             1.0
    X23       R9            0.5       R12            3.47
    X24       Z1           4.65       Z2             2.0
    X24       Z3           0.8        R5             1.0
    X24       R9           1.28       R12            2.8
    X25       Z2           2.5        Z3             0.8
    X25       R5           1.0        R9             0.35
    X26       Z2           2.5        Z3             0.7
    X26       R5           1.0
    X27·      Z2           4.0        Z3             4.0
    X27       R6           1.0
    X28       Z2           4.0        Z4             2.0
    X28       Z5           0.8        R7             1.0
    X28       R9           1.0
    X29       Z2           3.0        Z4             1.5
    X29       Z5           0.2        R7             1.0
    X30       Z2           3.0        Z4             4.0
    X30       Z5           1.8        R8             1.0
    X30       R9           2.0
    X31       Z2           2.0        Z4             3.0
    X31       Z5           0.45       R8             1.0
RHS
    RHS       R1          1600.       R2            900.
    RHS       R3           135.       R4            800.
    RHS       R5          3558.       R6           1052.
    RHS       R7          1701.       R8            776.
    RHS       R9          8000.       R10          8355.
    RHS       R11         5900.       R12            19700.
    RHS       R13        39500.
ENDATA
```

REFERENCES

1. Arrow, K. J., Social Choice and Individual Value, Wiley, NY, 1963.

2. Benayoun, R., J. de Montgolfier, J. Tergny, and O. Laritchev, "Linear programming with multiple objective functions: STEP-Method (STEM)", MATHEMATICAL PROGRAMMING 1, 1971, pp. 366-375.

3. Charnes, A. and W.W. Cooper, "Goal programming and multiple objective optimization", EJOR 1, 1977, pp. 39-54.

4. Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963.

5. Eckenrode, R.T., "Weighting multiple criteria", MANAGEMENT SCIENCE 12, 1965, pp. 180-192.

6. Edwards, W., "The theory of decision making", PSYCHOLOGICAL BULLETIN 51, 1954, pp. 380-417.

7. Fishburn, P.C., Utility Theory for Decision, Wiley, NY, 1970.

8. Fishburn, P.C., "Methods of estimating additive utilities", MANAGEMENT SCIENCE 13, 1967, pp. 435-453.

9. Geoffrion, A.M., "Solving bicriterion mathematical programs", OPERATIONS RESEARCH 15, 1967, pp. 39-54.

10. Geoffrion, A.M., "Proper efficiency and the theory of vector maximization", J. Math. Anal. and Appl. 22, 1968, pp. 618-630.

11. Geoffrion, A.M., J.S. Dyer and A. Feinberg, "An interactive approach for multi-criterion optimization, with an application to the operation of an academic department" MANAGEMENT SCIENCE 19, 1972, pp. 357-368.

12. Ho, J.K., "Multiple criteria decision making: a unified framework", AMD 820, Brookhaven National Laboratory, February 1979.

13. Philip, J., "Algorithms for the vector maximization problem", MATHEMATICAL PROGRAMMING 2, 1972, pp. 207-229.

14. Roy, B., "Problems and methods with multiple objective functions", MATHEMATICAL PROGRAMMING 1, 1971, pp. 239-266.

15. Srinivasan, V. and A. D. Shocker, "Estimating the weights for multiple attributes in a composite criterion using pairwise judgments", PYSCHOMETRIKA 38, 1973, pp. 473-493.

16. Steuer, R. E. and A. T. Schuler, "An interactive multiple objective linear programming approach to a problem in forest management", OPERATIONS RESEARCH 26, 1978, pp. 254-269.

17. Wallenius, J., "Comparative evaluation of some interactive approaches to multicriterion optimization", MANAGEMENT SCIENCE 21, 1975, pp. 1387-1396.

18. Yu, P.L, "Cone convexity, cone extreme points, and nondominated solutions in decision problems with multiobjectives", JOURNAL OF OPTIMIZATION THEORY AND APPLICATION 14, 1974, pp. 319-377.

19. Zeleny, M., "Adaptive displacement of preferences in decision making", TIMS Studies in the Management Science 6, 1977, pp. 147-157.

20. Zionts, S. and J. Wallenius, "An interactive programming method for solving the multiple criteria problem", MANAGEMENT SCIENCE 22, 1976, pp. 652-663.

AN IMPLEMENTATION OF THE REFERENCE POINT APPROACH FOR MULTIOBJECTIVE OPTIMIZATION

M. Kallio,* A. Lewandowski,* W. Orchard-Hays**

*System and Decision Sciences, IIASA
**Energy Systems Program, IIASA†

This paper studies the reference point approach of Wierzbicki for multiobjective optimization. The method does not necessarily aim at finding an optimum under any utility function but rather it is used to generate a sequence of efficient solutions which are interesting from the decision maker's point of view. The user can interfere via suggestions of reference values for the vector of objectives. The optimization system is used to find (in a certain sense) the nearest Pareto solution to each reference objective.

The approach is expanded for adaptation of information which may accumulate on the decision maker's preferences in the course of the interactive process. In this case any Pareto point is excluded from consideration if it is not optimal under any linear utility function consistent with the information obtained. Thus, the Pareto points being generated are the "nearest" ones among the rest of the Pareto points.

Wierzbicki's approach is implemented on an interactive mathematical programming system called SESAME and developed by Orchard-Hays. It is now capable of handling large practical multicriteria linear programs with up to 99 objectives and 1000 to 2000 constraints. The method is tested using a forest sector model which is a moderate sized dynamic linear program with twenty criteria (two for each of the ten time periods). The approach is generally found very satisfactory. This is partly due to the simplicity of the basic idea which makes it easy to implement and use.

†Currently at:

Energy Information Administration
U.S. Department of Energy, Washington, D.C.

1. INTRODUCTION

In many practical decision situations there is a need to find a compromise between a number of conflicting objectives. Furthermore, the decision may involve several decision makers in partly conflicting, partly cooperative situations. Mathematically such decision problems can often be formulated as a multiobjective optimization problem or in the framework of game theory. In this paper we concentrate on the former approach for developing decision aid techniques for the problem. For an overview on various approaches, see, for instance Bell et al. (1977), Starr and Zeleny (1977), and Wierzbicki (1979 b).

In our opinion, the reference point optimization method with penalty function scalarization (Wierzbicki 1979 a) is an appropriate tool for studying such problems. This approach has several desirable properties:

-- it applies to convex and nonconvex cases
-- it can easily check Pareto-optimality of a given decision
-- it can be easily supplemented by an *a posteriori* computation of trade-off coefficients for the objectives
-- it is numerically well-conditioned and easy for implementation

-- the concept of reference point optimization makes it possible to take into account the desires of a decision maker directly, without necessarily asking him questions about his preferences.

In this paper we will focus on the interactive use of reference point optimization for multiobjective linear programming with a single decision maker. However, we believe that the same approach proves to be useful for group decision problems as well. The reference point optimization will be reviewed first and some preliminary results will be given. Thereafter, we develop an approach for employing information which may be revealed on the decision maker's preferences in the course of the interactive process. The multiobjective method has been computerized in the SESAME-system, a large interactive mathematical programming system designed for IBM 370 under VM/CMS (Orchard-Hays 1978). A sample of numerical experiments will be reported at the end of the paper.

2. REFERENCE POINT OPTIMIZATION

Let A be in $R^{m\times n}$, C in $R^{p\times n}$, and b in R^m and consider the multicriteria linear program (MCLP):

$$\text{(MCLP.1)} \qquad Cx = q$$

$$\text{(MCLP.2)} \qquad Ax = b$$

$$\text{(MCLP.3)} \qquad x \geq 0 \quad ,$$

where the decision problem is to determine an n-vector x of decision variables satisfying (MCLP.2-3) and taking into account the p-vector q of objectives defined by (MCLP.1). We will assume that each component of q is desired to be as large as possible.

An objective vector value $q = \bar{q}$ is *attainable* if there is a feasible x for which $Cx = \bar{q}$. Let q_i^*, for $i = 1,2,\ldots, p$, be the largest attainable value for q_i; i.e., $q_i^* = \sup \{q_i | q \text{ attainable}\}$. The point $q^* \equiv (q_1^*, q_2^*,\ldots, q_p^*)^T$ is the *utopia point*. If q^* is attainable, it is a solution for the decision problem. However, usually q * is not attainable. A point $\bar{q}$ is *strictly Pareto inferior* if there is an attainable point q for which $q > \bar{q}$. If there is an attainable q for which $q \geq \bar{q}$ and the inequality is strict at least in one component, then $\bar{q}$ is *Pareto inferior*. An attainable point $\bar{q}$ is *weakly Pareto-optimal* if it is not strictly Pareto inferior and it is *Pareto-optimal* if there is no attainable point q such that q $\bar{q}$ with a strict inequality for at least one component. Thus a Pareto optimal point is also weakly Pareto optimal, and a weakly Pareto optimal point may be Pareto inferior. For brevity, we shall call a Pareto optimal point sometimes a *Pareto point* and the set of all such points the *Pareto set*.

What we call a *reference point* or *reference objective* is a suggestion $\bar{q}$ by the decision maker (or the group of them) reflecting in some sense an aspiration level for the objectives. According to Wierzbicki (1979 a), we consider for a reference point $\bar{q}$ a penalty scalarizing function $s(q-\bar{q})$ defined over the set of objective vectors q. Characterization of functions s, which result in Pareto optimal (or weakly Pareto optimal) minimizers of s over attainable points q is given by Wierzbicki (1979 b). See also Wierzbicki (1980) when the relations of reference point optimization to satisficing decision making are discussed.

If we regard the function $s(q-\bar{q})$ as the "distance" between the points q and $\bar{q}$, then, intuitively, the problem of finding such a minimum point means finding among the Pareto set the *nearest* point $\hat{q}$ to the reference point $\bar{q}$. However, as it will be clear later, our function s is not necessarily related to the usual notion of distance. Having this interpretation in mind, the use of reference points optimization may be viewed as a way of guiding a sequence $\{\hat{q}^k\}$ of Pareto points generated from the sequence $\{\bar{q}^k\}$ of reference objectives. These sequences will be generated in an interactive process and such interference should result in an interesting set of attainable points $\hat{q}^k$. If the sequence $\{\hat{q}^k\}$ converges, the limit point may be seen as a solution to the decision problem.

Initial information to the decision maker may be provided by maximising all objectives separately. Let $q^i = (q^i_j)$ be the vector of objectives obtained when the i^{th} objective is maximized for all i. Then the matrix (q_j^i), i,j, = 1,..., p, yields information on the range of numerical values of objective functions, and the vector $q^* = (q_i^i)$ is the utopia point. It should be stressed, however, that such initial information is not a necessary part of the procedure and in no sense limits the freedom of the decision maker.

We denote $w \equiv q - \bar{q}$, for brevity. Then, a practical form of the penalty scalarizing function s(w), where minimization results in a linear programming formulation, is given as follows:

$$s(w) = -\min\{\rho \min_i w_i, \sum w_i\} - \varepsilon w \quad . \qquad (1)$$

Here ρ is an arbitrary penalty coefficient which is greater than or equal to p and $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_p)$ is a nonnegative vector of parameters. In the special case of $\rho = p$, (1) reduces to

$$s(w) = -\rho \min_i w_i - \varepsilon w \quad . \qquad (2)$$

So far in our experience, form (1) of the penalty scalarizing function has proven to be most suitable. Other practical forms have been given in Wierzbicki (1979a).

For any scalar $\hat{s}$ the set $S_{\hat{s}}(\bar{q}) \equiv \{q | s(w) \geq \hat{s}, w = q - \bar{q}\}$ is called a level set. Such sets have been illustrated for function (1) with $\varepsilon = 0$ in Figure 1 for $\rho = p$, for $\rho > p$ and for a very large value for ρ. In each case, if $w \not\geq 0$, then s(w) is given by (2); i.e., the functional value is proportional to the worst component of w if $\varepsilon = 0$. If $\rho = p$, the same is true for $w \geq 0$ as well. If $w > 0$, then for large enough ρ (see the case $\rho >> p$) s(w) is given by $\sum w_i$. In the general case, when $\rho > p$, the situation is shown in the middle of Figure 1. When $w \geq 0$ and its components are close enough to each other (that is, $(\rho-1)w_1 \geq w_2$ and $(\rho-1)w_2 \geq w_1$, for p = 2), then s(w) is given by $\sum w_i$. Otherwise, formula (2) applies again.

For $\varepsilon = 0$, scalarizing function (1) guarantees only weak Pareto optimality for its minimizer. However, as will be shown in Lemma 1 below, if $\varepsilon > 0$, then Pareto optimality will be guaranteed.

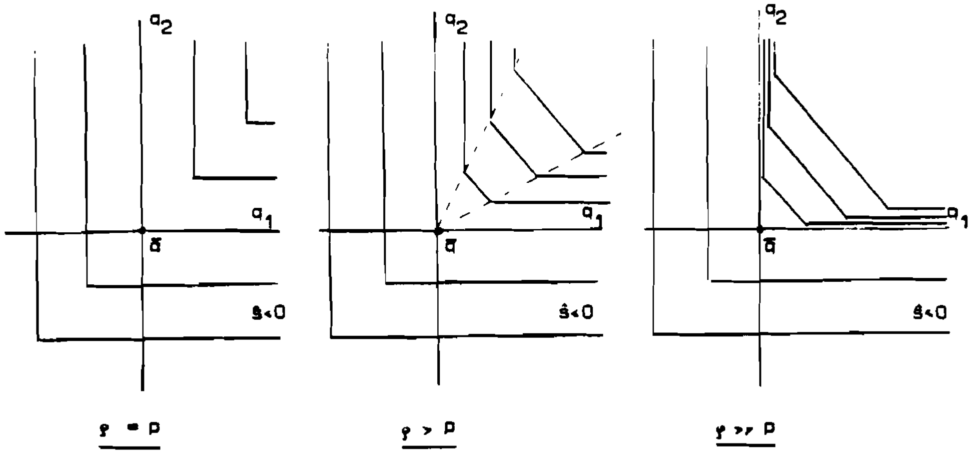


Figure 1. Level sets for penalty scalarizing functions (1) and (2) for $\varepsilon = 0$.

The problem of minimizing $s(q-\bar{q})$ defined by (1) over the attainable points q, can be formulated as a linear programming problem. In particular, if we again denote $w = q - \bar{q} = Cx - \bar{q}$ and introduce an auxiliary decision variable y, this minimization problem can be stated as the following problem (P):

find y, w, and x to

(P.1) $\quad \min \quad y - \varepsilon w$

(P.2) $\quad \text{s.t. } Ey + Dw \leq 0$

(P.3) $\quad -w + Cx = \bar{q}$

(P.4) $\quad Ax = b$

(P.5) $\quad x \geq 0$,

where E and D are appropriate vectors and matrices. Furthermore, $D \leq 0$, and if $w = \hat{w}$ and $y = \hat{y}$ are optimal for (P), then $\hat{s} = \hat{y} - \varepsilon\hat{w}$ is the minimum value attained for the penalty function s. The detailed formulation of (P) is given in the Appendix. The optimal solution for (P) will be characterized by the following result:

LEMMA 1. *Let $(y,w,x) = (\hat{y},\hat{w},\hat{x})$ be an optimal solution and δ, μ, and π the corresponding dual vectors related to constraints (P.2), (P.3), and (P.4), respectively. Denote by $\hat{q} = C\hat{x}$ the corresponding objective vector, and by $\hat{s} = \hat{y} - \varepsilon\hat{w}$ the optimal value for the penalty function, and by Q the attainable set of objective vectors q. Then $\hat{q} \in Q \cap S_{\hat{s}}(\bar{q})$ and the hyperplane $H = \{q | \mu(\hat{q}-q) = 0\}$ separates Q and $S_{\hat{s}}(\bar{q})$. Furthermore, $\mu \geq \varepsilon$ and $q = \hat{q}$ maximizes μq over $q \in Q$; i.e., $\hat{q}$ is Pareto optimal if $\varepsilon > 0$, and $\hat{q}$ is weakly Pareto optimal if $\varepsilon \geq 0$.*

Remark. As illustrated in Figure 2, the hyperplane H approximates the Pareto set in the neighborhood of $\hat{q}$. Thus the dual vector μ may be viewed as a vector of trade-off coefficients which tells roughly how much we have to give up in one objective in order to gain (a given small amount) in another objective. As seen in Figure 2, the assumptions of Lemma 1 might be satisfied provided $\varepsilon \geq 0$ is sufficiently small.

Proof. Clearly $\hat{q}$ is attainable (i.e., $\hat{q} \in Q$) and by definition $\hat{q} \in S_{\hat{s}}(\bar{q})$. In order to prove the separability assertion we show that (i) $\hat{q}$ minimizes μq over $S_{\hat{s}}(\bar{q})$ and that (ii) $\hat{q}$ maximizes μq over Q. Noting that $q = w + \bar{q} = Cx$, these two problems may be stated as follows:

$$\text{minimize } \mu w + \mu\bar{q}$$
$$\text{st.}$$

$$\text{P(i)} \qquad \begin{aligned} y - \varepsilon w &\geq \hat{s} \\ Ey + Dw &\leq 0 \end{aligned} \quad ,$$

and

$$\text{maximize } \mu Cx$$
$$\text{st.}$$

$$\text{P(ii)} \qquad \begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

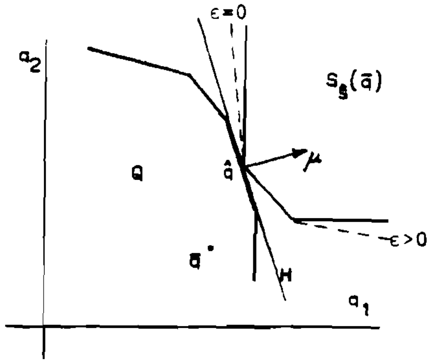


Figure 2. An illustration of Lemma 1.

Letting the dual multipliers for the first constraint of P(i) be equal to -1, we can readily check, based on the optimality conditions for (P), that $\hat{y}$, $\hat{w}$, $\hat{x}$, δ, μ, and $-\pi$ satisfy the optimality conditions for P(i) and P(ii). Based on dual feasibility, we have $\mu = \varepsilon - \delta D$ and $\delta \leq 0$. Because $D \leq 0$, we have $\mu \geq \varepsilon$. Thus, if $\varepsilon > 0$ ($\varepsilon \geq 0$), then $\hat{q}$ is (weakly) Pareto optimal.||

3. EMPLOYING INFORMATION ON PREFERENCES

While applying the reference point optimization a sequence $\{\bar{q}^k\}$ of reference points and the corresponding sequence $\{\hat{q}^k\}$ Pareto points will be generated. Usually these sequences reveal partially the decision makers preferences. For instance, after obtaining a Pareto point $\hat{q}^{k-1}$, a new reference point $\bar{q}^k$ may be chosen so that $\bar{q}^k$ is preferred to $\hat{q}^{k-1}$. In the following we intend to exploit such information. In such a procedure we shall not necessarily generate the nearest Pareto point to a reference point. We will restrict the Pareto points being generated to those which are consistent (in the sense defined below) with the information gained from the interactive process.

Initially, we will assume a linear utility function $\lambda^* q$, where λ^* is a vector such that q is preferred to q' if and only if $\lambda^* q > \lambda^* q'$, for all q and q'. The vector λ^* is not known explicitly. However, because each objective q_i is to be maximized, we have $\lambda^* \geq 0$; i.e., $\lambda^* d^i \geq 0$ for each unit vector d^i. Furthermore, other information concerning λ^* may be obtained during the interactive procedure. As above, if the decision maker prefers $\bar{y}^k$ to $\hat{y}^{k-1}$, then, denoting $d = \bar{y}^k - \hat{y}^{k-1}$, we have $\lambda d > 0$. In general let d^i, for $i = 1,2,\ldots, I_k$, be the vectors of *preferred directions* (including the unit vectors) being revealed by iteration k of the procedure. This implies that

$$\lambda^* \in \Lambda^k \equiv \{\lambda \mid \lambda d^i \geq 0 \ , \ \text{for } i = 1,2,\ldots, I_k\} \ , \qquad (3)$$

i.e., λ^* is in the dual cone of the cone spanned by the vectors d^i. (Actually, λ^* is in the interior of Λ^k.) See also Zionts and Wallenius (1976).

Let Q^k be the set of Pareto points which are *consistent* with respect to Λ^k in the sense that $\hat{q} \in Q^k$ if and only if there is $\lambda \in \Lambda^k$ such that $\lambda\hat{q} \geq \lambda q$, for all attainable $q \in Q$. We shall now discuss an approach to provide a Pareto point $\hat{q} \in Q^k$ related to a reference point $\bar{q}$. For this purpose we rewrite (P.3) as

$$(\bar{P}.3) \qquad -w + Cx - \sum_{i=1}^{I_k} d^i z_i = \bar{q} \quad ,$$

where the scalars z_i are nonnegative decision variables. This revised problem will be referred to as problem $(\bar{P})$. An interpretation of this problem is to find the nearest Pareto point (among all Pareto points) to the cone, which is spanned by the vectors d^i of preferred directions and whose vertex is at the reference point $\bar{q}$. Another characterization of the revised problem $(\bar{P})$ is given as follows:

LEMMA 2. *If $\varepsilon > 0$, $w = \hat{w}$ is optimal for the revised problem $(\bar{P})$, and $\hat{q} = \bar{q} + \hat{w}$, then $\hat{q} \in Q^k$; i.e., $\hat{q}$ is a Pareto point which is consistent with respect to the information obtained in Λ^k.*

Proof. Let $(y,w,x,z_i) = (\hat{y},\hat{w},\hat{x},\hat{z}_i)$ be optimal for $(\bar{P})$ and, as before, δ, μ, and π the optimal dual solution. Define $\bar{\bar{q}} = \bar{q} + \sum_i d^i z$. Then the above also solves (P) with the reference point $\bar{\bar{q}}$. Thus, by Lemma 1, $\mu \geq \varepsilon > 0$ and $\hat{q}$ maximizes μq over attainable points q. By the optimality condition for z_i, we have $\mu d^i \geq 0$, for all i. Thus $\mu \in \Lambda^k$, and therefore, $\hat{q}$ is a pareto point consistent with Λ^k.||

In practice, the decision makers utility function is usually not linear. However, in the neighborhood of his most desired solution the utility function has usually a satisfactory linear approximation and, therefore, the above procedure may still be useful. Because of nonlinearity, the vectors d^i of preferred directions may appear conflicting for a linear utility function; i.e., the set Λ^k reduces to a single point (the origin) and the vectors d^i span the whole space. Of course, this may occur also for reasons other than the nonlinearity. For instance, lack of training in using the approach may easily result in conflicting statements on preferences. In either case, such conflict results in an unbounded optimal solution for the revised problem $(\bar{P})$. In such a case, we suggest that the oldest vectors d^i (the ones generated first) will be deleted as long as boundedness for $(\bar{P})$ is obtained. This approach seems appealing in accounting both for the learning process of the user (decision maker) and for his possible nonlinear utility function.

4. COMPUTER IMPLEMENTATION

A package of SESAME/DATAMAT programs has been prepared for automating the use of the multicriteria optimization technique utilizing user-specified reference points. The scalarizing function defined in (1) was adopted for this implementation. A model revision into the form of (P) is carried out and a *neutral solution* corresponding to a reference point $\bar{q} = 0$ is computed and recorded first. Each time a new reference point $\bar{q}$ is given, the optimal solution for (P) is found starting with the neutral solution and using parametric programming, that is parametrizing the reference point as $\theta\bar{q}$ with θ increasing from 0 to 1. Some optional algorithmic devises have been implemented to force the sequence of Pareto points to converge. As it will be clear later, such a procedure does not guarantee an optimal solution (under any utility function) but often it is expected to be useful for generating interesting Pareto points. There is no explicit limit to the size of model which can be handled except that the number of objectives cannot exceed 99.

The package of programs is referred to as the MOCRIT Package, or simply MOCRIT. The standard package consists of three files: a SESAME RUN file, a DATAMAT program file, and a dummy data file which exists merely for technical reasons. There are essentially four programs in MOCRIT: (1) REVISION, which reformulates the model into the form of (P) and creates the neutral solution, (2) START, which initializes the system for a interactive session, (3) SESSION, which utilizes the standard technique of reference point optimization, and (4) CONVERGE, which forces the sequence of Pareto points to converge. The use of REVISION and SESSION is mandatory. START is a convenience to obviate the need to enter various SESAME parameters for each session. CONVERGE is an option; it cannot be used meaningfully before SESSION has been executed at least once. CONVERGE is actually a prologue to SESSION which it activates as a terminal step.

These "programs" are really RUN decks consisting of appropriate SESAME commands. There are corresponding decks (DATAMAT programs) which are executed automatically by the RUN decks. All four MOCRIT programs terminate by returning to the SESAME environment in manual mode. Regular SESAME commands and procedures can be interspersed manually from the terminal at such times. (For details, see Orchard-Hays 1977).

4.1 The REVISION Program

The purpose of this program is to revise an existing linear programming model containing two or more functional rows into a form suitable for multiobjective optimization. The existing model file must have been previously created with DATAMAT (or CONVERT) in standard fashion. This file is not altered; a new file containing the revised model is created instead.

After creating the new model, REVISION further solves the model with a reference point of all zero, and obtains thereby the neutral solution. This initial solution must be obtained only once and the optimal basis is recorded on a disk file for further use.

REVISION also creates another file containing two tables. One is used to record selected results form the neutral solution. The other is used by the START program to set the various SESAME parameters for the revised model, i.e., model name, model file name, RHS name, name of RANGE set if any, and name of BOUND set. Thus it is unnecessary to set these for subsequent sessions.

The reference point $\bar{q}$ as well as the model parameters dependent on the coefficients ρ and ε are specified initially in the revised model as symbolic names. When their values are decided on, they are specified numerically at run time without generating the whole model over again. For instance, to obtain the neutral solution, REVISION requires coefficients ρ and ε. Their values are obtained via an interactive response. If it is subsequently changed (see the SESSION program) the neutral solution will, in general, no longer be feasible. This may not be done normally but, if necessary, a new neutral solution can be obtained as shown in Orchard-Hays (1979).

A user-specified number of columns will be reserved for the preferred directions d^i ; i.e., for the decision variables z^i. Also the d^i vectors are specified initially in the revised model as symbolic names. Their values are initially set strictly positive so that the z^i variables do not appear at a positive level in an optimal solution of $(\bar{P})$. Afterwards, these positive vectors will be (cyclically) replaced by preferred directions whenever they are generated in the course of the interactive process.

4.2 The START and SESSION Programs

After a model has been revised and the neutral solution obtained and recorded, the model is ready for use with the interactive multiobjective procedure. Such use is referred to as a *session*. A session is initiated by executing the START program. All this does is define the necessary SESAME parameters unique to the model.

After executing START but before executing SESSION, the reference point must be defined. This is done with the SESAME procedure VALUES which is quite flexible with respect to formats and functions. If necessary, also the value of the coefficients ρ and ε may be changed at this point. After the reference point has been defined, execution of SESSION results in the following sequence of events.

(i) Any existing solution file is erased.

(ii) The problem set-up procedure is called and the existing reference point is incorporated for use in parametric programming.

(iii) The basis of the neutral solution is recalled.

(iv) The simplex procedure is called. After a basis inversion and check of the solution, the neutral solution is recovered.

(v) The parametric programming procedure is called to parametrize the reference point $\theta\bar{q}$ over the parameter values $\theta \in [0,1]$.

(vi) A SESAME procedure is called to record selected portions of the solution.

(vii) DATAMAT is called to execute a program to display results at the terminal (and to print off-line) and also to record necessary information for possible subsequent use by CONVERGE.

(viii) The control is returned to SESAME in manual mode.

If it is desired to try another reference point, we call the procedure VALUES again and then rerun SESSION. This may be done repeatedly.

If it is desired to get a print-out of the full solution (or selected portions) in standard LP solution format after return from SESSION, it can be obtained using the SESAME procedures in the usual way (see Orchard-Hays 1977). An example of part of the results displayed at the terminal is given in Figure 3. Each row carrying user-defined labels F1 to I10 refers to an objective. The column REFER.PT defines the reference point $\bar{q}$, column SUB.FN yields the Pareto point $\hat{q}$ obtained, and column W is just the difference $\hat{q} - \bar{q}$ of the above two columns. Column DUAL is the (negative of the) vector μ of trade off coefficients defined in Lemma 1.

```
                 REFER.PT      SUB.FN        W        DUAL

F1       =         2048         2670        622        -.99
F2       =         1398         2020        622        -.56
F3       =          688         1310        622        -.63
F4       =          508         1130        622        -.65
F5       =          358          980        622        -.65
F6       =         -161          461        622        -.63
F7       =         1489         2111        622        -.57
F8       =         2599         3221        622        -.49
F9       =         4709         5331        622       -1.12
F10      =         5849         6471        622        -.67
I1       =         2035         2657        622       -1.33
I2       =         2889         3511        622        -.40
I3       =         2328         2950        622        -.76
I4       =         3348         3970        622        -.98
I5       =         4368         4990        622       -1.17
I6       =         4328         4950        622       -1.28
I7       =         5349         5971        622       -1.29
I8       =         5859         6481        622       -1.24
I9       =         7339         7961        622       -2.81
I10      =         7849         8471        622       -1.68
```

Figure 3. An example of results displayed in a session. (The reference point is $\bar{q}^5$ of Section 5.2).

4.3 The COVERGE Program

The CONVERGE program may be used instead of SESSION after the latter has been executed at least once. The VALUES procedure must be executed first, as usual, to define a new reference point. However, this reference point, denoted by $\bar{\bar{q}}$, is not actually used. Let $\hat{q}^k$ be the last Pareto point obtained (by either SESSION or CONVERGE). A new reference point is computed from $\bar{\bar{q}}$ in two stages as follows. First $\bar{\bar{q}}$ is projected on the hyperplane H defined in Lemma 1, passing through $\hat{q}^k$ and orthogonal to the dual vector μ. This projection q^* is given by

$$q^* = \bar{\bar{q}} + [\mu(\hat{q}^k - \bar{\bar{q}})/\mu\mu^T]\mu^T \quad . \qquad (4)$$

The new reference point $\bar{q}^{k+1}$ is then chosen from the line segment $[q^*, \hat{q}^k]$; i.e., a point $\bar{q}^{k+1} = q^* + \theta(\hat{q}^k - q^*)$ is chosen for some $\theta \in [0,1]$. The following options have been considered: (i) choose $\theta = 0$ (i.e., choose $\bar{q}^{k+1}$ as the projection q^*), or (ii) choose the smallest $\theta \in [0,1]$ so that $\max_i(\bar{q}^{k+1} - \hat{q}_i^k) \leq y^k$, where y^k is a user-specified tolerance. The value for y may either be entered directly or it may be specified as a percentage of the "distance" between the previous reference point $\bar{q}^k$ and the Pareto point $\hat{q}^k$; i.e., $y^k = \beta^k \max_i(\bar{q}_i^k - \hat{q}_i^k)$, where β^k is a coefficient entered by the user. This latter option may be used meaningfully only if the reference point $\bar{q}^k$ is not a Pareto inferior point, for instance, a point obtained by CONVERGE in the preceeding session. For an illustration of the modified reference point, see Figure 4.

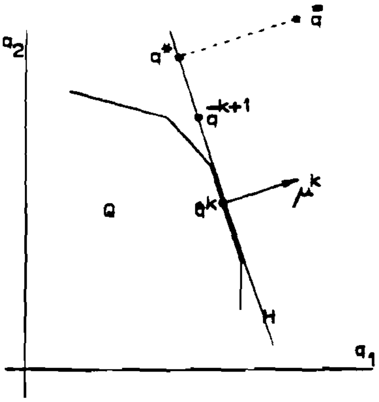


Figure 4. Modification of the reference point in CONVERGE.

Note that

$$y^k \geq \max_i(\bar{q}_i^{k+1} - \hat{q}_i^k) \geq \max_i(\bar{q}^{k+1} - \hat{q}^{k+1}) \geq 0 \quad . \tag{5}$$

Thus, if $y^k \geq 0$ and the sequence $\{y^k\}$ converges to zero, then the sequence of optimal values for (P) converges to zero.

Remark. A limit point of $\{\hat{q}^k\}$ is not necessarily a solution to the multicriteria optimization problem, because the convergence is mechanically forced without taking the decision maker's preferences properly into account. The only purpose of the CONVERGENCE routine is to provide some algorithmic help to converge to a, hopefully, interesting Pareto point.

5. COMPUTATIONAL EXPERIENCE

For testing purposes we used a ten period dynamic linear programming model developed for studying long-range development alternatives of forestry and forest based industries in Finland (Kallio et al. 1978). This model comprises two subsystems, the forestry and the industrial subsystem, which are linked to each other through raw wood supply. The forestry submodel describes the development of the volume of different types of wood and the age distribution of different types of trees in the forests within the nation. In the industrial submodel various production activities, such as saw mill, panels production, pulp and paper mills, as well as further processing of primary wood products, are considered. For a single product, alternative technologies may be employed so that the production process is described by a small Leontief model with substitution. Besides supply of raw wood and demand for wood products, production is restricted through labor availability, production capacity, and financial resources. All production activities are grouped into one financial unit and the investments are made within the financial resources of this unit. Similarly, the forestry is considered as a single financial unit.

A key issue between forestry and industry is the income distribution which is determined through raw wood price. Consequently, we have chosen two criteria: (i) the profit of the wood processing industries, and (ii) the income of forestry from selling the raw wood to industry. These objectives are considered separately for each time period of the model. Thus, the problem in consideration has 20 criteria altogether.

Of course, both the average raw wood price and quantity of wood sold must be implicit in such a model. In order to handle this in a linear programming framework, we use interpolation. We consider two exogeneously given wood prices for each type of raw wood and for each period. The quantities sold at each price are endogeneous and the average wood price results from the ratio of these quantities. The complete model after REVISION consists of 712 rows and 913 columns.

We experiment first with different values for the penalty coefficient ρ. Then, fixing $\rho = p$ (the number of objectives) we generate a sequence $\{\bar{q}^k\}$ of reference points and compute the corresponding sequence $\{\hat{q}^k\}$ of pareto points as solutions to (P). The influence of accumulated information on preferences will be experimented with thereafter. Experience with CONVERGE will then be reported briefly. All these experiments have been carried out with an early version of MOCRIT for which $\varepsilon = 0$. A sample of runs with our current version for which $\varepsilon > 0$ will be reported finally.

5.1 Influence of the Penalty Coefficient

Using the scalarizing function (1) we experimented with different values of the penalty coefficient ρ and with different reference points $\bar{q}$. As pointed out in Section 2, unless the reference point $\bar{q}$ is Pareto inferior, the Pareto point $\hat{q}$ obtained as a solution of (P) is independent of ρ, namely the one corresponding to the max min criterion of the scalarizing function (2). On the other hand, if $\bar{q}$ is Pareto inferior, then $\hat{q}$ in general depends on ρ. In the extreme case of $\rho = p$, we again obtain the max min solution.

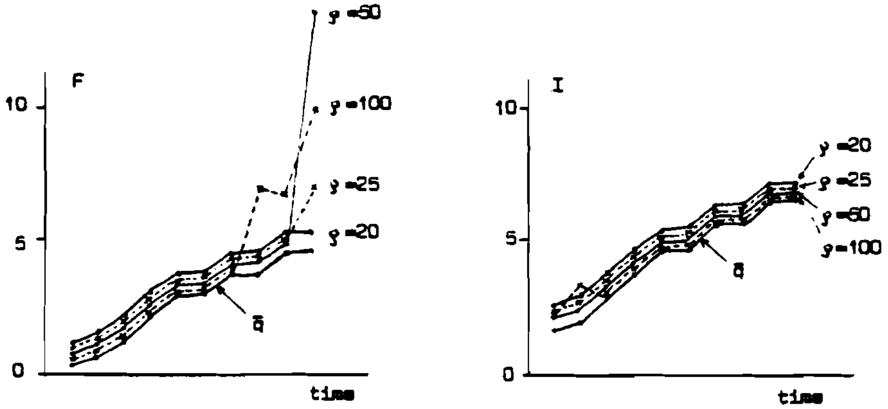


Figure 5. Experiments with different penalty coefficients and with the reference point about 90 percent of a pareto point.

In an experiment illustrated in Figure 5 an attainable reference point $\bar{q}$ has been chosen and the values 20(=p), 25, 50 and 100 have been applied to ρ. As $\bar{q}$ now is Pareto inferior the Pareto trajectories obtained are dependant on ρ. For $\rho = p$, a constant deviation $\hat{w}_i = \bar{q}_i - \hat{q}_i = 0.4$ is obtained for each objective i. When ρ increases the minimum guaranteed for each w_i decreases. Simultaneously as ρ increases, the behavior of the Pareto-tragectories $\hat{q}$ gets worse in that large spikes appear in these trajectories.

In this example $\bar{q}$ actually is about 90 percent of a Pareto-solution. When a (Pareto-inferior) reference point is moved further from the Pareot-set according to our experience, the behavior of the Pareto-trajectories get more sensitive to the value of ρ; i.e., spikes appear already with values of ρ relatively close to p, and for a given $\rho > p$, the spikes grow worse when $\bar{q}$ moves further from the Pareto-set.

5.2 Experiments with a Sample of Reference Points

For further tests we set $\rho = p$, generated a sequence of nine reference points $\bar{q}^k$, $k = 0,1,\ldots, 8$, and the corresponding Pareto solutions. The results have been illustrated in Figures 6 and 7, for $\bar{q}^k$, $k = 3,4,\ldots, 8$. The continuous trajectories refer to the reference point, and those drawn in broken lines refer to the Pareto point. As an overall observation we may conclude, that the trajectory of the Pareto solution tends to be the reference trajectory shifted up or down. (See also Figure 5 for $\rho = 20$.) However, this is not always the case. In Figure 6 (a) the Pareto trajectory has a very large spike. Such undesirable unsmoothness may be due to a multiplicity of optimal solution which are very different from each other. In our dynamic case, for instance, the first

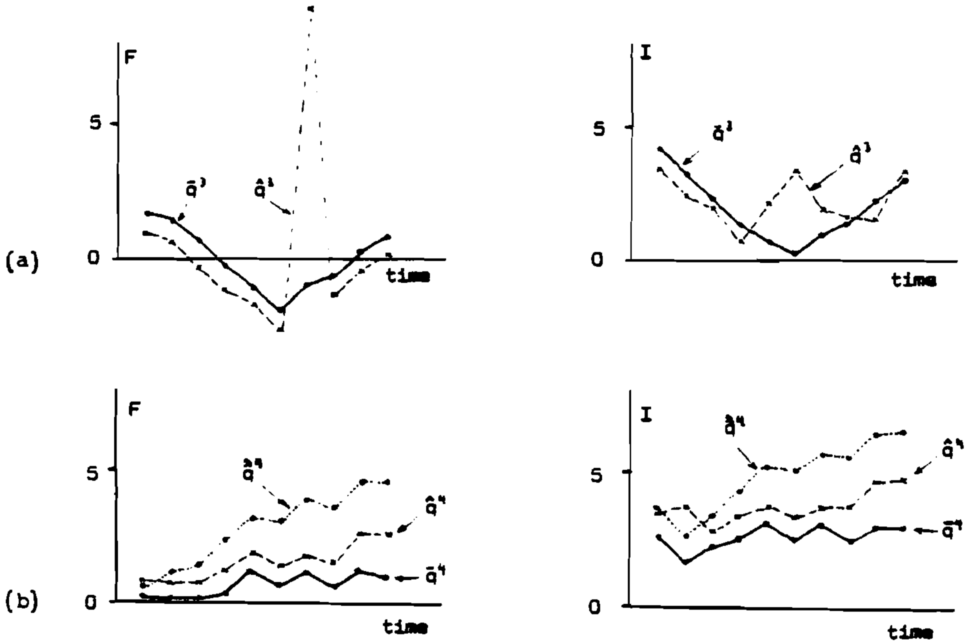


Figure 6 A sample of sessions

periods may totally determine the optimal objective function value for (P) and the multiple optimal solutions result from the variety of alternatives left for the later periods.

Next, the influence of the accumulated information on preferences was experimented. Again, let $\hat{q}^k$ be the Pareto-trajectory corresponding to the reference trajectory $\bar{q}^k$, $k = 0,1,\dots,8$. For the purpose of our numerical tests we assume that the differences $d^k = \bar{q}^k - \hat{q}^{k-1}$ reveal the decision makers preferences in a way that d^k is a preferred direction, for $k = 1,2,\dots,8$. All vectors d^k, for $i \leq k$, will be made available when applying the reference point $\bar{q}^k$ in the revised problem $(\bar{P})$. Thus, all information gained on preferences is being used. The Pareto points resulting as optimal

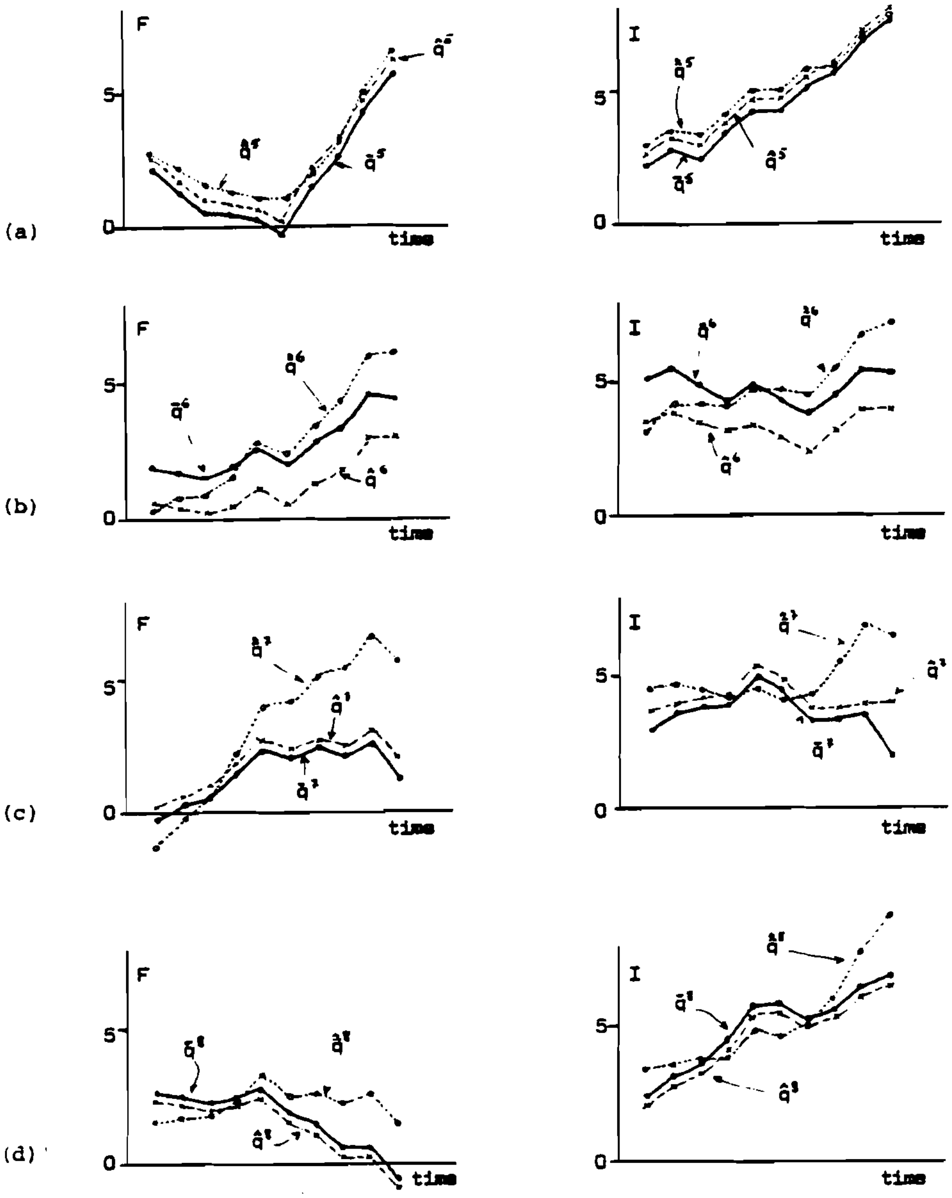


Figure 7. A sample of sessions (continued).

solutions for $(\bar{P})$ have been illustrated in dotted lines in Figures 6 and 7. For k = 1,2, and 3, the additional information did not have any influence on the Pareto point; i.e., the same solutions $\hat{q}^k$ were obtained as before. However, thereafter a significant change was observed in most cases, and in addition, the obtained revised Pareto point seems more appealing than the one obtained from problem (P) (see Figures 6(b), 7(b), and 7(c), for instance). On the other hand, we may observe that the revised trajectories usually resemble the shape of the reference trajectory to a lesser degree than do the trajectories obtained form problem (P). These observations suggest that perhaps in practice both Pareto trajectories ought to be computed in each session.

5.3 Forcing Convergence

In Section 4 we developed procedures for modifying the users suggested sequence of reference points in such a way that the Pareto points obtained are forced to converge. One of these procedures was controlled by a sequence $\{\beta^k\}$ of percentages, and another by a sequence $\{y^k\}$ of tolerances. Both of them were tested using the same sequence $\{\bar{q}^k\}_{k=0}^{8}$ of reference points of section 5.2 .

First we discuss the case of using the β-factors. After obtaining the initial solution $\hat{q}^0$, the CONVERGE program was applied for each suggestion $\bar{q}^k$. The results obtained when a constant value $\beta^k = .5$ (for all k) was used, indicate that practically no change in $\hat{q}^k$ occurs after $k \geq 2$. The same phenomenon was discovered for $\beta^k = .9$ (for all k). Thus the convergence proved to be extremely fast. An explanation for this phenomenon may be found from the fact that the hyperplane (on which the reference points are projected) is close to the Pareto set in the neighborhood of the last

Pareto point obtained. This in turn is likely to result in a sequence of objective function values for (P), which converges fast to zero.

For the other procedure, we chose the bounds y^k as $y^k = 10/2^k$. The converge appeared to be now reasonably fast, but not too fast. Thus, the user has a fair chance to control the sequence of Pareto points being generated.

5.4 A sample of runs with $\varepsilon > 0$.

All the previous runs were made with the parameter vector $\varepsilon = 0$. As indicated by Lemma 1, this may not guarantee Pareto-optimality for the trajectories $\hat{q}^k$. However, even then, a sufficient but not a necessary condition for Pareto-optimality is that the dual vector μ is strictly positive. This condition in fact was satisfied in many cases of the previous runs, and it is likely that most other cases (which did not satisfy this sufficient condition) resulted in a Pareto optimal trajectory as well. In any event, more recently we have experimented also with our current version of MOCRIT to see whether the main qualitative results obtained in Section 5.2 hold also when $\varepsilon > 0$ (i.e., when Pareto-optimality for the $\hat{q}$ trajectories is guaranteed).

Figure 8 shows a sample of reference trajectories and the respective Pareto trajectories when $\rho = p$ each component of ε is set to 10^{-6}. Similarly as observed in Section 5.2, the Pareto trajectories tend now to result from a shift in the reference trajectories. More importantly, sharp spikes, which occasionally were obtained in Section 5.2 (see Figure 6 (a), for instance), did not result in our four examples of Figure 8 nor in other experiments which we did with $\varepsilon > 0$.

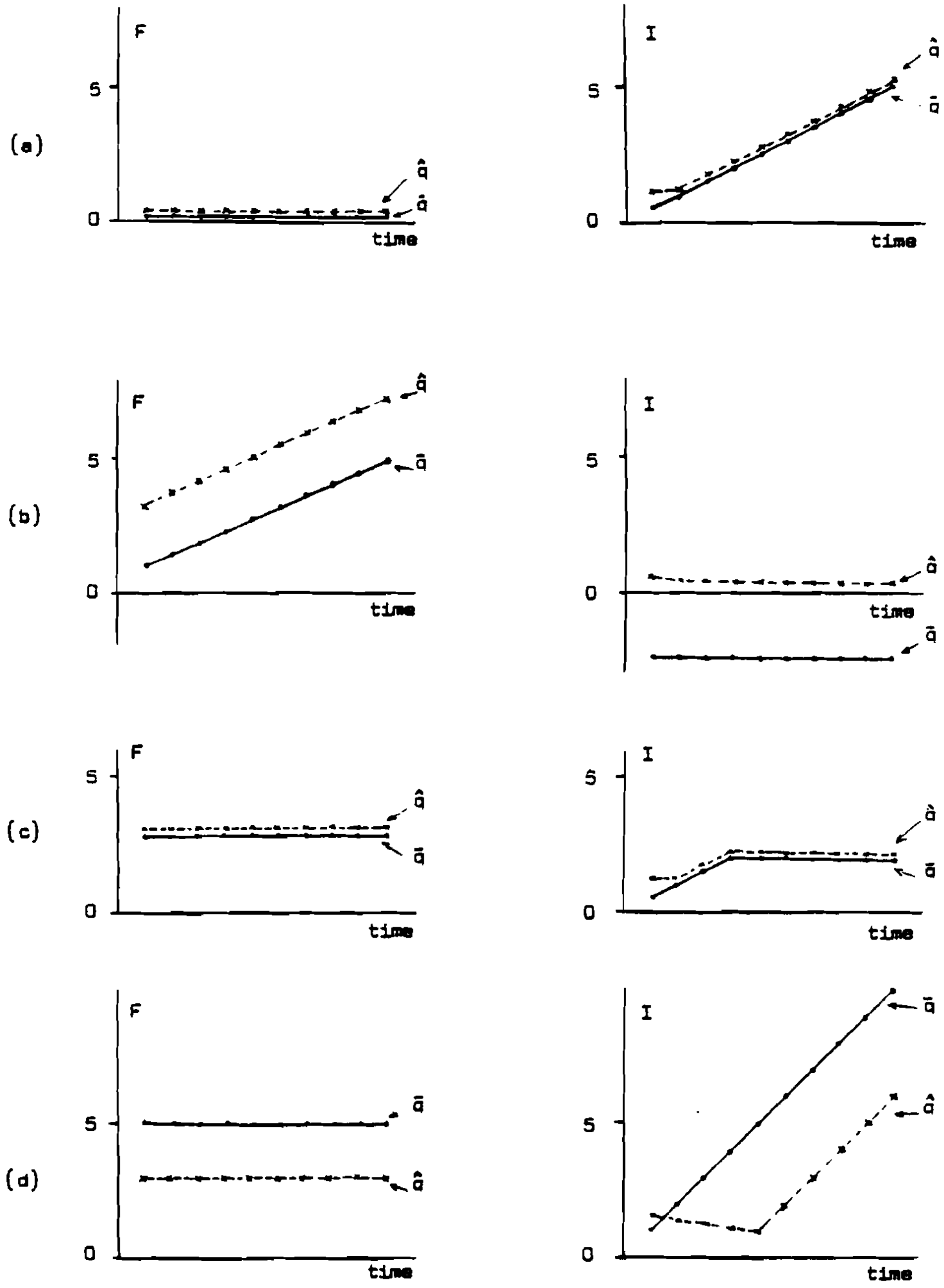


Figure 8 A sample of sessions with $\varepsilon > 0$.

Naturally, it would be desirable to repeat the experiments of Section 5.2 with $\varepsilon > 0$. However, these runs were made half a year earlier with a slightly different version of the model, and this version is no longer available. Nevertheless, the authors feel that no drastic new conclusions can be expected from further testing, and therefore, additional extensive and resource consuming experimenting has been neglected.

6. SUMMARY AND CONCLUSIONS

In this paper we have investigated the reference point approach for linear multiobjective optimization (Wierzbicki 1979a, b). In our opinion, the basic concept proves to be very useful, in particular, because of its simplicity. The method does not necessarily aim at finding an optimum under any utility function, but rather it is used to generate a sequence of interesting Pareto points. In order to guarantee usefulness of the information being generated, we let the decision maker interfere with the model system. In the course of such an interactive process he suggests reference objectives which normally reflect his desired levels of various objectives. The optimization system is used to find, in some sense, the nearest Pareto point to each reference objective.

As a measure of distance between the reference points and the Pareto set we use the penalty scalarizing function (1) which in our experience has very favorable properties: first, the problem of finding the nearest Pareto point to a reference point amounts to linear programming problem, and second, it allows the user a reasonable control over the sequence of Pareto points generated, given that the penalty coefficient ρ is close to the number of objectives p and a small $\varepsilon > 0$ is chosen. To clarify the latter point we have observed that, if $\rho >> p$ and $\varepsilon = 0$, the scalarizing function has an undesirable property of favoring arbitrarily one or a few components of the objective vector. In such a case, the objective levels at the Pareto point and at the reference objective may be close to each other in all except one component where the Pareto point is far superior to the reference objective. In dynamic cases this phenomenon usually causes spikes in trajectories of the objectives (see Figure 5 for large values of the penalty coefficient ρ). However, this phenomenon has not been observed if $\rho = p$ and $\varepsilon > 0$.

We have expanded the reference point approach for the adaptation of information which accumulates on the decision maker's preferences in the course of the interactive process. In this case we exclude from consideration every Pareto point which is not optimal under any linear utility function consistent with the information obtained so far. Thus the Pareto point being generated is the nearest one among the rest of the Pareto points.

We have implemented the reference point approach using the interactive mathematical programming system, called SESAME (Orchard-Hays 1978). The package of programs consists of essentially two parts: first, a DATAMAT program which reformulates a linear programming model in the form $(\bar{P})$ of reference point optimization, and second, a routine to carry out an interactive iteration (i.e., to insert a reference objective, and to compute and display the pareto point). The current implementation employs the scalarizing function (1) with the components of vector ε being all equal. The system is now capable of handling large practical multicriteria linear programs with up to 99 objectives and one or two thousand constraints.

For computational experimentation we used a dynamic LP model of a forest sector with about 700 rows and 900 columns. There are two objectives defined for each of the ten time periods of the model, i.e., there are twenty objectives in total. We experimented first with different values of the penalty coefficient ρ. The results suggest that for ρ one should use a value which is equal to or slightly larger than p, the number of objectives. Based on this observation, we set $\rho = p = 20$ for further numerical test runs. Samples of reference points have been tried out and the overall performance of the method has been found to be satisfactory. For $\varepsilon = 0$, however, we observed occasional undesirable unsmoothness in the computed trajectories of the two objectives (see Figure 6(a)). This may be due to the fact that only weak Pareto optimality is guaranteed, for $\varepsilon = 0$ (see Lemma 1). Indeed, as discussed in Section 5.4, this problem seems to disappear when $\varepsilon > 0$ (and pareto optimality is guaranteed).

A general observation is that the Pareto trajectories tend to agree with the reference objectives shifted up or down. This property was found not to be valid when experimenting with the extension of employing cumulative information on preferences. However, after this information began to influence the solution the Pareto trajectories generally appeared more appealing than those obtained disregarding this information (see Figures 7(b) and 7(c)).

A reader familiar with the goal programming approach might observe the similarity of the algorithm discussed in this paper, to goal programming algorithms. In fact, the algorithm has been derived from the reference point approach to multiobjective optimization which is a generalization of goal programming: in particular, the algorithm works as well for Pareto-dominated reference objective points which cause difficulties in typical goal programming. Moreover, the questions of eliminating weakly Pareto-optimal solutions and of employing cumulative information on users preferences have not been considered in typical goal programming.

APPENDIX

Derivation of Problem (P)

Denote by $W \equiv \{w | -w+Cx=\bar{q},\ Ax=b,\ x\geq 0\}$ the feasible set for vector w. Then the reference point optimization problem, when the scalarizing function (1) is applied, is as follows:

$$\min_{w\in W}\{-\min\{\rho \min_{i} w_i,\ \sum_{i} w_i\} -\varepsilon w\}$$

$$= \min_{w\in W}\{\max\{\max_{i}(-\rho w_i)\ ,\ -\sum_{i} w_i\} - \varepsilon w\}$$

$$= \min_{w\in W}\{\max\{\max_{i}(-\rho w_i-\varepsilon w)\ ,\ -\sum_{i} w_i - \varepsilon w\}\}$$

$$= \min_{\substack{w\in W\\ z\in R}}\{z | z\geq -\rho w_i-\varepsilon w,\ \text{for all } i,\ z \geq -\sum_{i} w_i - \varepsilon w\}$$

$$= \min_{\substack{w\in W\\ y\in R}}\{y-\varepsilon w | -y-\rho w_i \leq 0,\ \text{for all } i,\ -y-\sum_{i} w_i \leq 0\} \quad ,$$

where we have substituted $y = z + \varepsilon w$.

REFERENCES

Bell, D., R. Keeney, and H. Raiffa (eds.) (1977) Conflicting Objectives in Decisions. IIASA International Series on Applied Systems Analysis. New York: Wiley.

Kallio, M., A. Propoi, and R. Seppälä. A Model for the Forest Sector. Laxenburg, Austria: International Institute for Applied Systems Analysis. Forthcoming.

Orchard-Hays, W. (1977) A Simplified Introduction to VM/CMS and SESAME/DATAMAT Software. Technical Report (unpublished). Laxenburg, Austria: International Institute for Applied Systems Analysis.

Orchard-Hays, W. (1978) Anatomy of a mathematical programming system. In: Design and Implementation of Optimization Software, edited by H. Greenberg. Netherlands: Sijthoff and Noordhoff.

Orchard-Hays, W. (1979) Multi-objective Criterion Optimization. User's Guide (unpublished). Laxenburg, Austria: International Institute for Applied Systems Analysis.

Starr, M., and M. Zeleny (eds.) (1977) Multiple criteria decision making. TIMS Studies in the Management Sciences, 6.

Wierzbicki, A. (1979a) The Use of Reference Objectives in Multiobjective Optimization. Theoretical Implications and Practical Experience. WP-79-66. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Wierzbicki, A. (1979b) A Methodological Guide to Multiobjective Optimization. WP-79-122. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Wierzbicki, A. (1980) A Mathematical Basis for Satisficing Decision Making. WP-80-90. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Zionts, S., and J. Wallenius (1976) An interactive programming method for solving the multiple criteria problem. Management Science 22:652-663.

A MODEL FOR THE FOREST SECTOR†

M. Kallio,* A. Propoi,** R. Seppälä***

*System and Decision Sciences, IIASA
**Institute for Systems Studies, USSR Academy of Sciences, Moscow
***Mathematics Department, The Finnish Forest Research Institute, Helsinki

This paper describes a dynamic linear programming model for studying long-range development alternatives of forestry and forest based industries at a national and regional level. The Finnish forest sector is used as an object of implementation and for numerical examples. Our model is comprised of two subsystems, the forestry and the industrial subsystem, which are linked to each other through the wood supply. The forestry submodel describes the development of the volume and age distribution of different tree species within the nation or its subregions. In the industrial submodel we consider various production activities, such as saw mill industry, panel industry, pulp and paper industry, as well as further processing of primary products. For a single product, alternative technologies may be employed. Thus, the production process is described by a small Leontief model with substitution. Besides supply of wood and demand of wood products, production is restricted through labor availability, production capacity, and financial resources. The production activities are grouped into financial units and the investments are made within the financial resources of such units. Objective functions related to GNP, balance of payments, employment, wage income, stumpage earnings, and industrial profit have been formulated. Terminal conditions have been proposed to be determined through an optimal solution of a stationary model for the whole forest sector.

The structure of the integrated forestry-forest industry model is given in the canonical form of dynamic linear programs for which special solution techniques may be employed. Two versions of the Finnish forest sector models have been implemented for the interactive mathematical programming system called SESAME, and a few numerical runs have been presented to illustrate possible use of the model.

†The authors wish to thank William Orchard-Hays for his help in implementing our forest sector model using the SESAME system, and Jari Kuuluvainen (of the Finnish Forest Research Institute), Heikki Seppälä (of the Finnish Forest Research Institute), and Margareta Soismaa (of the Helsinki School of Economics) for data collection. The work has benefited from discussions with George B. Dantzig (Stanford University), to whom we express our sincere gratitude. Partial financial support has been received from the Academy of Finland.

1. INTRODUCTION

As is the case with several natural resources, many regions of the world are now at the transition period from ample to scarce wood resources. Because the forest sector plays an important role in the economy of some countries, long-term policy analysis of the forest sector, i.e., forestry and forest industries, is becoming an important issue for these countries.

We may single out two basic approaches for analyzing long-range development of the forest sector: simulation and optimization. Simulation techniques (e.g., system dynamics) allow us to understand and to quantify basic relationships influencing the development of the forest sector (see Jegr et al. 1978, Randers 1976, Seppälä et al. forthcoming). Hence, using a simulation technique we can evaluate the consequences of a specific policy. However, using only simulation it is difficult to find a "proper" (or in some sense optimal) policy. The reason for this is that the forest sector is in fact a large-scale dynamic system and, on the basis of simulation alone, it is difficult to select an appropriate policy which should satisfy a large number of conditions and requirements. For this we need an optimization technique. Because of the complexity of the system in question, linear programming (Dantzig 1963) may be considered as the most appropriate technique for this case. It is worthwhile to note that the optimization technique itself should be used on some simulation basis; i.e., different numerical runs based on different assumptions and objective functions should be carried out to aid the selection of an appropriate policy. Specific applications of such an approach for planning an integrated system of forestry and forest industries have been presented, for instance, by Jackson (1974) and Barros and Weintraub (1979).

Already because of the nature of growth of the forests, the model should necessarily be dynamic. Therefore, in this paper we consider a dynamic linear programming (DLP) model for the forest sector. In this approach the planning horizon (e.g., a 50-year period) is partitioned into a (finite) number of time periods (e.g., 5-year periods) and for each of these shorter periods we consider a static linear programming model. A dynamic LP is then just a linear program comprising of such static models which are interlinked via various state variables (i.e., different types of "inventories", such as wood in the forests, production capacity, assets, liabilities, etc., at the end of a given period are equal to those at the beginning of the following period). In our forest sector model, each such static model comprises two basic submodels: a forestry submodel, and an industrial model of production, marketing and financing. The forestry submodel describes also ecological and land availability constraints for the forest, as well as labor and machinery constraints for harvesting and planting activities.

The industrial submodel is described by a small input-output model with both mechanical (e.g., sawmill and plywood) and chemical (e.g., pulp and paper) production activities. Also secondary processing of the primary products will be included in the model, in particular, because of the expected importance of such activities in the future.

The rate of production is restricted by wood supply (which is one of the major links between the submodels), by final demand for wood products, by labor force supply, by production capacity availability, and finally, by financial considerations.

The evaluation criterion in comparing alternative policies for the forest sector is highly multiobjective: while selecting a reasonable long-term policy, preferences of different interest groups (such as government, industry, labor, and forest owners) have to be taken simultaneously into account. It should also be noted that forestry and industry submodels have different transient times: a forest normally requires a growing period of at least 40 to 60 years whereas a major structural change in the industry may be carried out within a much shorter period. Because of the complexity of the system, it is sometimes desirable to consider the forestry and the industries on some independent basis, each with its own objective(s), and to analyze an integrated model thereafter (see Kallio et al. 1979).

The paper is divided into two parts. In the first part (Sections 2-4) we describe the methodological approach. In the second part (Section 5) a specific implementation for the Finnish forest sector is described and illustrated with somewhat hypothetical numerical examples.

2. THE FORESTRY SUBSYSTEM

Mathematical programming is a widely applied technique for operations management and planning in forestry (e.g., Navon 1971, Dantzig 1974, Kilkki et al. 1977, Newnham 1975, Näslund 1969, Wardle 1965, Ware and Clutter 1971, Weintraub and Navon 1976, Williams 1976). In this section we follow a traditional formulation of the forests' tree population into a dynamic linear programming system. We describe the forestry submodel, where the decision variables (control activities) are harvesting and planting activities, and where the state of the forests is represented by the volume of trees in different species and age groups. Because the model is formulated in the DLP framework, we single out the following: (i) state equations which describe the development of the system, (ii) constraints which restrict feasible trajectories of the forest development, (iii) planning horizon, and (iv) objective function(s).

2.1 State Equations

Each tree in the forest is assigned to a class of trees specifying the age and the species of the tree. A tree belongs to age group a ($a = 1,\ldots, N-1$) if its age is at least $(a-1)\Delta$ but less than $a\Delta$, where Δ is a given time interval (for example, five years). In the highest age group $a = N$ all trees are included which have an age of at least $(N-1)\Delta$. (Instead of age groups, we might alternatively assign trees to size groups specified by the trees' diameter.) We denote by $w_{sa}(t)$ the number of trees of species s, $s = 1,2,3,\ldots,$ (e.g., pine, spruce, birch, etc.) in age group a at the beginning of time period t, $t = 0,1,\ldots, T$.

Let $\alpha^s_{aa'}(t)$ show the ratio of trees of species s and in age group a that will proceed to the age group a' during time period t. We shall consider a model formulation where the length of each time period is Δ. Therefore, we may assume that $\alpha^s_{aa'}(t)$ is independent of t and equal to zero unless a' is equal to a+1 (or a for the highest age group). We denote then $\alpha^s_{aa'}(t) = \alpha^s_a$ with $0 \leq \alpha^s_a \leq 1$. The ratio $1 - \alpha^s_a$ may then be called the attrition rate corresponding to time interval Δ and tree species s in age group a. We introduce a subvector $w_s(t) = \{w_{sa}(t)\}$, specifying the age distribution of trees (number of trees) for each tree species s at the beginning of time period t. Assuming neither harvesting nor planting, the age distribution of trees at the beginning of the next time period t+1 will then be given by $\alpha^s w_s(t)$ where α^s is the square $N \times N$ growth matrix, describing aging and death of the trees resulting from natural causes. By our definition, it has the form

$$\alpha^s = \begin{bmatrix} 0 & 0 & & 0 \\ \alpha^s_1 & 0 & & 0 \\ 0 & \alpha^s_2 & & \\ 0 & \ldots & \alpha^s_{N-1} & \alpha^s_N \end{bmatrix} .$$

Introducing a vector $w(t) = \{w_s(t)\} = \{w_{sa}(t)\}$, describing tree species and age distribution and a block-diagonal matrix α with submatrices α^s on its diagonal, the species and age distribution at the beginning of period t+1 will be given by $\alpha w(t)$.

We denote by $u^+(t)$ and $u^-(t)$ the vectors of planting and harvesting activities at time period t. The state equation describing the development of the forest will then be

$$w(t+1) = \alpha w(t) + \eta u^+(t) - \omega u^-(t) \quad , \qquad (1)$$

where matrices η and ω specify planting and harvesting activities in such a way that $\eta u^+(t)$ and $-\omega u^-(t)$ are the incremental change in numbers of trees resulting from planting and harvesting activities, respectively.

A planting activity n may be specified to mean planting of one tree of species s which enters the first age group (a = 1) during period t. Thus, matrix η has one unit column vector for each tree species s. The nonzero element of such a column is on the row of the first age group for tree species s in equation (1).

A harvesting activity h is specified by variables $u_h^-(t)$ which determine the level of this activity (e.g., final harvesting, thinning, etc.). The coefficients ω_{ah}^s of matrix ω are defined so that $\omega_{ah}^s u_h^-(t)$ is the number of trees of species s from age group a harvested when activity h is applied at level $u_h^-(t)$. Thus, these coefficients show the age and species distribution of trees harvested when activity h is applied.

Sometimes the harvesting activities can be specified simply by the numbers of trees of species s and age a harvested during time period t. There is some danger in this specification, however, because the solution of the model may suggest that only one or very few age groups will be harvested at each time period t. This would of course be unrealistic in practice. Therefore, it is recommended that each harvesting activity is defined through a tree distribution corresponding to actual operations.

2.2 Constraints

Land. Let $H(t)$ be the vector of total acreage of different types d of land available for forests at time period t. A land type d may refer, for instance, to a soil type. Let G^s_{ad} be the area of land species d required by one tree of species s and age group a. We assume that each tree species uses only one type of land d; i.e., only one of the elements G^s_{ad}, $d = 1, 2, \ldots,$ is nonzero. Thus, if we consider more than one land type, then the tree species s may also refer to the soil. Defining the matrix $G = (G^s_{ad})$, we have the land availability restriction

$$Gw(t) \leq H(t) \quad . \tag{2}$$

In this formulation we assume that the land area $H(t)$ is exogenously given. Alternatively, we may endogenize vector $H(t)$ by introducing activities and a state equation for changing the area of different types of land. Such a formulation is justified if changes in soil type over time is considered or if some other land intensive activities, such as agriculture, are included in the model.

Besides land availability constraints, requirements for allocating land for certain purposes (such as preserving the forest as a water shed or as a recreational area) may be stated in the form of inequality (2). In such a case (the negative of) a component of $H(t)$ would define a lower bound on such an allocation, while the left hand side would yield the (negative of) land allocated in a solution of the model.

Sometimes constraints on land availability may be given in the form of equalities which require that all land which is made available through harvesting at a time period should be used in the same time period for planting new trees of the type appropriate for the soil. Forest laws in many countries even require following this type of pattern.

Labor and other resources. Harvesting and planting activities require resources such as machinery and labor. Let $R^+_{gn}(t)$ and $R^-_{gh}(t)$ be the usage of resource g at the unit level of planting activity n and harvesting activity h, respectively. Defining the matrices $R^{+}(t) = \{R^{+}_{gn}(t)\}$ and $R^{-}(t) = \{R^{-}_{gh}(t)\}$, and vector $R(t) = \{R_g(t)\}$ of available resources during period t, we may write the resource availability constraint as follows:

$$R^{+}(t)u^{+}(t) + R^{-}(t)u^{-}(t) \leq R(t) \quad . \tag{3}$$

Wood supply. The requirements for wood supply from forestry to industries can be given in the form:

$$S(t)u^{-}(t) = y(t) \quad , \tag{4}$$

where vector $y(t) = \{y_k(t)\}$ specifies the requirements for different timber assortments k (e.g., pine log, spruce pulpwood, etc.), and matrix S(t) transforms quantities of harvested trees of different species and age into the volume of different timber assortments. Note that the volume of any given tree being harvested is assigned in (4) to log and pulpwood in a ratio which depends on the species and age group of the tree.

2.3 Planning Horizon

The forest as a system has a very long transient time: one rotation of the forest may in extreme conditions require more than one hundred years. Naturally, various uncertainties make it difficult to plan for such a long time horizon. On the other hand, if the planning horizon is too short we cannot take into account all the consequences of activities implemented at the beginning of the planning horizon. As a compromise we may think of a planning horizon of 50 to 80 years. Thus, if one period represents an interval of five years, the model will constitute 10 to 16 stages. It should be noted that such a planning horizon is unnecessarily long for the industrial subsystem and too short for the forestry subsystem. In order to eliminate the latter difficulty, it is desirable to analyze a stationary regime for the forests. In this case we set $w(t+1) = w(t) = w$, for all t. Similarly planting and harvesting activities are taken independent of time; i.e., $u^{+}(t) = u^{+}$ and $u^{-}(t) = u^{-}$, for all t. The state equation (1) can then be restated as

$$w = \alpha w + \eta u^{+} - \omega u^{-} \quad . \qquad (1a)$$

Imposing constraints (2) through (4) on variables w, u^{+}, and u^{-}, we can solve the static linear programming problem and find an optimal stationary state w^{*} of the forest (and corresponding harvesting and planting activities). This approach has been used, for instance, by Rorres (1978) for finding the stationary maximum yield of a harvest. The solution of a dynamic linear program with terminal constraints

$$w(T) = w^{*}$$

yields the optimal transition to this stationary state.

Another way of introducing a stationary state is to consider an infinite period formulation and to impose constraints $w(t) = w(t+1)$, $u^{-}(t) = u^{-}(t+1)$ and $u^{+}(t) = u^{+}(t+1)$, for all $t \geq T$. If the model parameters for period t are assumed independent of time for all $t \geq T$, then the dynamic infinite horizon linear programming model may be formulated as a T+1 period problem where the last period represents a stationary solution for periods $t \geq T$, and the first T periods represent the transition from the initial state to the stationary solution.

There is a certain difference in these two approaches of handling the stationary state. In the first approach, when (5) is applied, we first find the optimal stationary solution independently of the transition period, and thereafter we determine the optimal transition to this stationary state. In the latter approach we link the transition period with the period corresponding to the stationary solution. The linkage takes place in the stationary state variables which are determined in an optimal way taking into account both time periods simultaneously.

2.4 Objective Functions

The forest management described above, has a very multiobjective nature. For example, the following objectives have been mentioned (Dantzig 1974, Steuer and Schuler 1978):

1) obtaining higher yields of round wood; 2) preserving the watershed; 3) preserving the forest as a recreational area; 4) making the forest resilient to diseases, fire, droughts, etc. Some of these objectives may be included in objective function(s), while others can be given as constraints. In Section 2.2 we considered some of these types of objectives as constraints.

A common objective which is also used as an objective function is the discounted sum of net income in forestry. This profit may be expressed as a linear combination of the decision variables:

$$\sum_{t=0}^{T-1} \beta(t)[J^{-}(t)u^{-}(t) - J^{+}(t)u^{+}(t)] \quad . \qquad (6)$$

Here $J^{-}(t)$ accounts for the mill price of the wood less transportation and harvesting costs at unit level. Vector $J^{+}(t)$ refers to planting costs at unit level and $\beta(t)$ is a discounting factor. For illustrative purposes we shall use this objective function for forestry.

2.5 Forestry Model

In summary, our forestry model may now be stated as follows. Given state equation (1), an initial state $w(0) = w^0$ and a terminal state $w(T) = w^*$, find such nonnegative controls $\{u^{-}(t)\}$ and $\{u^{+}(t)\}$ $(t = 0,1,..., T-1)$, which satisfy constraints (2) through (4), yield nonnegative state vectors $w(t)$ and maximize the aggregated profit defined in (6).

In this problem the vector $y(t)$ of wood supply, the (vector of) available land $H(t)$, and the availability of labor and other resources $R(t)$ are given exogenously. Therefore, policy analysis for forestry on the basis of only this submodel is very limited in its possibilities. We shall link below this submodel with an industrial submodel describing transformation of wood raw material into products.

Note that our formulation may also be considered as a regionalized forestry model. In this case we only have to extend the meaning of various indices (tree species s, planting activity n, harvesting activity h, land type d, resource g, and timber assortment k) to refer, in addition to the above, also to various subregions within the nation.

3. THE INDUSTRIAL SUBSYSTEM

We will now consider the industrial subsystem of the forest sector. Again the formulation is a dynamic linear programming model. We discuss first the section related to production and final demend of wood products, then the financial considerations and the complete industrial submodel thereafter.

3.1 Production and Demand

Let $x(t)$ be the vector (levels of) of production activities for period t, for $t = 0, 1, \ldots, T-1$. Such an activity i may include production of sawn wood, panels, pulp, paper, converted products, etc. For each single product j, there may exist several alternative production activities i which are specified through alternative uses of raw material, technology, etc. Let U be the matrix of wood usage per unit of production activity so that the wood processed by industries during period t is given by vector $Ux(t)$. Note that matrix U has one row corresponding to each timber assortment k (corresponding to the components of supply vector $y(t)$ in the forestry model). Some of the elements in U may be negative. For instance, saw milling consumes logs but produces raw material (industrial residuals) for pulp mills. This byproduct appears as a negative component in matrix U. We denote by $r(t) = \{r_k(t)\}$ the vector of wood raw material inventories at the beginning of period t (i.e., wood harvested but not processed by the industry). As above, let $y(t)$ be the amount of wood harvested in different timber assortments, and $z^+(t)$ and $z^-(t)$ the (vectors of) import and export of different assortments of wood, respectively during period t. Then we have the following state equation for the wood raw material inventory:

$$r(t+1) = r(t) + y(t) - Ux(t) + z^+(t) - z^-(t) \quad . \tag{7}$$

In other words, the wood inventory at the end of period t is the inventory at the beginning of that period plus wood harvested and imported less wood consumed and exported (during that period). Note that if there is no storage (change), and no import nor export of wood, then (7) reduces to $y(t) = Ux(t)$; i.e., wood harvested equals the consumption of wood. For wood import and export we assume upper limits $Z^+(t)$ and $Z^-(t)$, respectively:

$$z^+(t) \leq Z^+(t) \quad \text{and} \quad z^-(t) \leq Z^-(t) \quad . \tag{8}$$

The production process may be described by a simple input-output model with substitution. Let A(t) be an input-output matrix having one row for each product j and one column for each production activity i so that A(t)x(t) is the (vector of) net production when production activity levels are given by x(t). Let $m(t) = \{m_j(t)\}$ and $e(t) = \{e_j(t)\}$ be the vectors of import from and export to the forest sector, respectively, for products j. Then, excluding from consideration a possible change in the product inventory, we have

$$A(t)x(t) + m(t) - e(t) = 0 \quad . \tag{9}$$

Both for export and for import we assume externally given bounds E(t) and M(t), respectively:

$$e(t) \leq E(t) \quad , \tag{10}$$

$$m(t) \leq M(t) \quad . \tag{11}$$

Production activities are further restricted through labor and mill capacities. Let L(t) be the vector of different types of labor available for the forest industries. Labor may be classified in different ways taking into account, for instance, type of production, and the type of responsibilities in the production process (e.g., work force, management, etc.). Let $\rho(t)$

be a coefficient matrix so that $\rho(t)x(t)$ is the (vector of) demand for different types of labor given production activity levels $x(t)$. Thus we have

$$\rho(t)x(t) \leq L(t) \quad . \tag{12}$$

We will consider the production (mill) capacity as an endogenous state variable. Let $q(t)$ be the vector of the amount of different types of such capacity at the beginning of period t. Such types may be distinguished by region (where the capacity is located), by type of product for which it is used and by different technologies to produce a given product. Let $Q(t)$ be a coefficient matrix so that $Q(t)x(t)$ is the demand (vector) for these types of capacity. Such a matrix has nonzero elements only when the region-product-technology combination of a production activity matches with that of the type of capacity. The production capacity restriction is then given as

$$Q(t)x(t) \leq q(t) \quad . \tag{13}$$

The development of the capacity is given by a state equation

$$q(t+1) = (I-\delta)q(t) + v(t) \quad , \tag{14}$$

where δ is a diagonal matrix accounting for (physical) depreciation and $v(t)$ is a vector of investments (in physical units). Capacity expansions are restricted through financial resources. We do not consider possible constraints of other sectors, such as heavy machinery or building industry, whose capacity may be employed in investments of the forest sector.

3.2 Finance

We will now turn our discussion to the financial aspects. We partition the set of production activities i into financial units (so that each activity belongs uniquely to one financial unit). Furthermore, we assume that each production capacity is assigned to a financial unit so that each production activity employs only capacities assigned to the same financial unit as the activity itself.

Production capacity in (14) is given in physical units. For financial calculations (such as determining taxation) we define a vector $\bar{q}(t)$ of fixed assets. Each component of this vector determines fixed assets (in monetary units) for a financial unit related to the capacity assigned to that unit. Thus, fixed assets are aggregated according to the grouping of production activities into financial units, for instance, by region, by industry, or by groups of industries.

Financial and physical depreciation may differ from each other; for instance, when the former is defined by law. We define a diagonal matrix $(I-\bar{\delta}(t))$ so that $(I-\bar{\delta}(t))\bar{q}(t)$ is the vector of fixed assets left at the end of period t when investments are not taken into account. Let $\bar{K}(t)$ be a matrix where each component determines the increase in fixed assets (of a certain financial unit) per (physical) unit of an investment activity. Thus the components of vector $\bar{K}(t)v(t)$ determine the increase in fixed assets (in monetary units) for the financial units when investment activities are applied (in physical units) at a level determined by vector $v(t)$. Then we have the following state equation for fixed assets:

$$\bar{q}(t+1) = (I-\bar{\delta}(t))\bar{q}(t) + \bar{K}(t)v(t) \quad . \qquad (15)$$

For each financial unit we consider external financing (long-term debt) as an endogenous state variable. Let $\ell(t)$ be the (vector of) beginning balance of external financing for different financial units in period t. Similarly, let $\ell^+(t)$ and $\ell^-(t)$ be the (vectors of) drawings of debt and the repayments made during period t. In this notation, the state equation for long-term debt is as follows:

$$\ell(t+1) = \ell(t) + \ell^+(t) - \ell^-(t) \quad . \qquad (16)$$

We will restrict the total amount for long-term debt through a measure which may be considered as a realization value of a financial unit. This measure is a given percentage of the total assets less short-term liabilities. Let $\mu(t)$ be a diagonal matrix of such percentages, let $b(t)$ be the (endogenous vector of) total stockholders equity (including cumulative profit and stock). Then the upper limit on loans is given as

$$[I-\mu(t)]\ell(t) \leq \mu(t)b(t) \quad . \tag{17}$$

Alternatively, external financing may be limited, for instance, to a percentage of a theoretical annual revenue (based on available production capacity and on assumed prices of products). Note that no repayment schedule has been introduced in our formulation, because an increase in repayment can always be compensated by an increase of drawings in the state equation (16).

Next we will consider the profit (or loss) from period t. Let $p^+(t)$ and $p^-(t)$ be vectors whose components indicate profits and losses, respectively, for the financial units. By definition, both profit and loss cannot be simultaneously nonzero for any financial unit. For a solution of the model, this fact usually results from the choice of an objective function.

Let $P(t)$ be a matrix of prices for products (having one column for each product and one row for each financial unit) so that the vector of revenue (for different financial units) from sales $e(t)$ outside the forest industry is given by $P(t)e(t)$. Let $C(t)$ be a matrix of direct unit production costs, including, for instance, wood, energy, and direct labor costs. Each row of $C(t)$ refers to a financial unit and each column to a production activity. The (vector) of direct production costs for financial units is then given by $C(t)x(t)$.

The fixed production costs may be assumed proportional to the (physical) production capacity. We define a matrix $F(x)$ so that the vector $F(t)q(t)$ yields the fixed costs of period t for the financial units. According to our notation above, (financial) depreciation is given by the vector $\bar{\delta}(t)\bar{q}(t)$.

We assume that interest is paid on the beginning balance of debt. Thus, if $\varepsilon(t)$ is the diagonal matrix of interest rates, then the vector of interest paid (by the financial units) is given by $\varepsilon(t)\ell(t)$. Finally, let $D(t)$ be (a vector of) exogeneously given cash expenditure covering all other costs. Then the profit before tax (loss) is given as follows:

$$\begin{aligned} p^{+}(t) - p^{-}(t) = P(t)e(t) - C(t)x(t) - F(t)q(t) \\ - \bar{\delta}(t)\bar{q}(t) - \varepsilon(t)\ell(t) - D(t) \quad . \end{aligned} \tag{18}$$

The stockholder equity $b(t)$, which we already employed above, satisfies now the following state equation:

$$b(t+1) = b(t) + [I-\tau(t)]p^{+}(t) - p^{-}(t) + B(t) \quad , \tag{19}$$

where $\tau(t)$ is a diagonal matrix for taxation and $B(t)$ is the (exogenously given) amount of stock issued during period t.

Finally, we consider cash (and receivables) for each financial unit. Let $c(t)$ be the vector of cash at the beginning of period t. The change of cash during period t is due to the profit after tax (or loss), depreciation (i.e., noncash expenditure), drawing of debt, repayment, and investments. Thus we assume that the possible change in cash due to changes in accounts receivable, in inventories (wood, end products, etc.) and in accounts payable cancel each other (or that these quantities remain unchanged during the period). Alternatively, such changes could be taken into account assuming, for instance, that the accounts payable and receivable, and the inventories are proportional to annual sales of each financial unit.

Using our earlier notation, the state equation for cash is now

$$\begin{aligned} c(t+1) = c(t) + [I-\tau(t)]p^{+}(t) - p^{-}(t) + \bar{\delta}(t)\bar{q}(t) \\ + \ell^{+}(t) - \ell^{-}(t) - K(t)v(t) + B(t) \quad . \end{aligned} \tag{20}$$

3.3 Initial State and Terminal Conditions

In our industrial model, we now have the following state vectors: wood raw material inventory $r(t)$, (physical) production capacity $q(t)$, fixed assets $\bar{q}(t)$, long-term debt $\ell(t)$, cash $c(t)$, and total stockholders equity $b(t)$. For all of them we have an initial value and possibly a limit on the terminal value. We shall refer to the initial and terminal values by superscripts 0 and *, respectively; i.e., we have the initial state given as

$$\begin{aligned} &r(0) = r^0 \;, \quad q(0) = q^0 \;, \quad \bar{q}(0) = \bar{q}^0 \;, \\ &\ell(0) = \ell^0 \;, \quad c(0) = c^0 \;, \quad b(0) = b^0 \;, \end{aligned} \tag{21}$$

and a terminal state restricted, for instance, as follows:

$$\begin{aligned} &r(T) \geq r^* \;, \quad q(T) \geq q^* \;, \quad \bar{q}(T) \geq \bar{q}^* \;, \\ &\ell(T) \leq \ell^* \;, \quad c(T) \geq c^* \;. \end{aligned} \tag{22}$$

The initial state is determined by the state of the forest industries at the beginning of the planning horizon. The terminal state may be determined as a stationary solution similarly as we described for the forestry model above.

If we consider the wood supply $y(t)$ being exogenous, we now have an industrial submodel which may be analyzed independently from the forestry submodel. A more complete duscussion on objectives will be given in the next section, but for illustrative purposes, we may choose now the discounted sum of industrial profits (after tax) as an objective function:

$$\sum_{t=0}^{T-1} \beta(t)[(I-\tau(t))p^+(t)-p^-(t)] \quad . \tag{23}$$

Here $\beta(t)$ is a (row) vector where components are the discounting factors for different financial units (for period t).

3.4 Industrial Model

We may now summarize the industrial model. Given initial state (21), find nonnegative control vectors $x(t)$, $z^+(t)$, $z^-(t)$, $m(t)$, $e(t)$, $v(t)$, $\ell^+(t)$, $\ell^-(t)$ $p^+(t)$, and $p^-(t)$, and nonnegative state vectors $r(t)$, $q(t)$, $\bar{q}(t)$, $\ell(t)$, $c(t)$, and $b(t)$, for all t which satisfy constraints and state equations (7) - (20), the terminal requirements (22), and maximize the linear functional given in (23).

As was the case with the forestry model, our industrial model may also be considered being regionalized. Again various indices (such as production activities, production capacities, etc.) should also refer to subregions within the country. Various transportation costs will then be included in direct production costs. For instance for a given product being produced within a given region there may be alternative production activities which differ from each other only in the source region of raw material.

4. THE INTEGRATED SYSTEM

We will now consider the integrated forestry--forest industries model. First we have a general discussion on possible formulations of various objective functions for such a model. Thereafter, we summarize the model in the canonical form of dynamic linear programming. A tableau representation of the structure of the integrated model will also be given.

4.1 Objectives

The forest sector may be viewed as a system controlled by several interest groups or parties. Any given party may have several objectives which are in conflict with each other. Obviously, the objectives of one party may be in conflict with those of another party. For instance, the following parties may be taken into account: representatives of industry, government, labor, and forest owners. Objectives for industry may be the development of profit of different financial units. Government may be interested in the increment of the forest sector to the gross national product, to the balance of payments, and to employment. The labor unions are interested in employment and total wages earned in forestry and different industries within the sector. Objectives for forest owners may be the income earned from selling and harvesting wood. Such objectives refer to different time periods t (of the planning horizon) and possibly also to different product lines. We will now give simple examples of formulating such objectives into linear objective functions.

Industrial profit. The vector of profits for the industrial financial units was defined above as $[I-\tau(t)]p^{+}(t) - p^{-}(t)$ for each period t. If one wants to distinguish between different financial units, then actually each component of such a vector may be considered as an objective function. However, often we aggregate such objectives for practical purposes, for instance, summing up discounted profits over all time periods, summing over financial units, or as in (23), summing over both time periods and financial units.

Increment to gross national product. For the purpose of defining the increment of the forest sector to the GNP we consider the sector as a "profit center" where no wage is paid to the employees within the sector, where no price is paid for raw material originating from this sector, and where no taxes exist. The increment to the GNP is then the profit for such a center. We will now make a precise statement of such a profit which may also be viewed as the valued added in the forest sector.

Let $P'(t)$ be a price vector so that $P'(t)e(t)$ is the total revenue from selling wood products outside the forest sector. Let $C'(t)$ be the vector of direct production unit costs excluding direct labor cost and cost of raw material which originates from the forest sector. Let $\hat{R}(t)$ and $\check{R}(t)$ be vectors of unit cost of planting and harvesting activities, respectively, excluding labor costs. For simplicity, we may assume that these latter two cost components include both operating and capital cost for machinery. The direct operating costs (excluding wages and wood based raw material) is then given, for period t, by

$C'(t)x(t) + \hat{R}(t)u^{+}(t) + \check{R}(t)u^{-}(t)$. Also the import and export of wood based raw material influence the GNP. Let $\hat{Z}(t)$ and $\check{Z}(t)$ be price vectors for imported and exported wood raw material, respectively, and let $M'(t)$ be the price vector of imported wood based products (to be used as raw material). Thus, the following term should be added to the GNP of period t: $\check{Z}(t)z^{-}(t) - \hat{Z}(t)z^{+}(t) - M'(t)m(t)$. The influence of the change in the wood inventory may be neglected in our model. For the fixed costs all except the labor costs will be taken into account. Let $F'(t)$ be the vector of such costs per unit of production capacity, let $\delta'(t)$ be the vector of depreciation factors, and $\varepsilon'(t)$ the vector of interest rates (for various financial units). Then the negative increment of the fixed costs, depreciation and interest to the GNP is given by $F'(t)q(t) + \delta'(t)\bar{q}(t) + \varepsilon'(t)\ell(t)$. Summing up, the increment of the forest sector to the GNP of period t is given by the following expression:

$$P'(t)e(t) - C'(t)x(t) - \hat{R}(t)u^{+}(t) - \check{R}(t)u^{-}(t) - \hat{Z}(t)z^{+}(t)$$

$$+ \check{Z}(t)z^{-}(t) - M'(t)m(t) - F'(t)q(t) - \delta'(t)\bar{q}(t) - \varepsilon'(t)\ell(t).$$

Increment to balance of payments. The increment of the forest sector to the balance of payments has a similar expression to the one above for the GNP. The changes to be made in this expression are, first, to multiply the components of the price vector $P'(t)$ by the share of exports in the total sales $e(t)$; second, to multiply the components of the cost vectors $C'(t)$, $\hat{R}(t)$, $\check{R}(t)$, and $F'(t)$ by the share of imported inputs in each cost term; third, to multiply each component of $\varepsilon'(t)$ by the share of foreign debts (among all long-term debts) of the financial unit; and finally, to replace the depreciation function $\delta'(t)\bar{q}(t)$ by investment expenditures $K'(t)v(t)$, where $K'(t)$ is a vector expressing investments in imported goods (per unit of production capacity).

Employment. Total employment (in man-years per period) for each time period t for different types of labor, in different activities and regions, has already been expressed in the left hand side expressions of inequalities (3) and (12). The expression for forestry is given by (part of the component of) the vector $R^+(t)u^+(t) + R^-(t)u^-(t)$ and for the industry by the vector $\rho(t)x(t)$.

Wage income. For each group of the work force, the wage income for period t is obtained by multiplying the expressions for employment above by the annual salary of each such group.

Stumpage earnings. Besides the wage income for forestry (which we already defined above), and an aggregate profit (as expressed in (6)), one may account for the stumpage earnings; i.e., the income related to the wood price prior to harvesting the tree. Such income is readily obtained by the timber assortments if the components of the harvesting yield vector $y(t)$ are multiplied by the respective wood prices.

4.2 The Integrated Model

We will now summarize the integrated forestry-industry model in the canonical form of dynamic linear programming (Propoi and Krivonozhko 1978). Denote by $X(t)$ the vector of all state variables (defined above) at the beginning of period t. Its components include the trees in the forest, different types of production capacity in the industry, wood inventories, external financing, etc. Let $Y(t)$ be the nonnegative vector of all controls for period t, that is, the vector of all decision variables, such as levels of harvesting or production activities. An upper bound vector for $Y(t)$ is denoted by $\hat{Y}(t)$ (some of whose components may be infinite). We assume that the objective function to be maximized is a linear function of the state vectors $X(t)$ and the control vectors $Y(t)$, and we denote by $\gamma(t)$ and $\lambda(t)$ the coefficient vectors for $X(t)$ and $Y(t)$, respectively, for such an objective function. This function may be, for instance, a linear combination of the objectives defined above. The initial state $X(0)$ is denoted by X^0, and the terminal requirement for $X(T)$ by X^*. Let $\Gamma(t)$ and $\Lambda(t)$ be the coefficient matrices for $X(t)$ and $Y(t)$, respectively, and let $\xi(t)$ be the exogenous right hand side vector in the state equation for $X(t)$.

Let $\Phi(t)$, $\Omega(t)$, and $\psi(t)$ be the corresponding matrices and the right hand side vector for the constraints. Then the integrated model can be stated in the canonical form of DLP as follows:

find $Y(t)$, for $0 \leq t \leq T-1$, and $X(t)$, for $1 \leq t \leq T$, to

$$\text{maximize} \sum_{t=0}^{T-1} (\gamma(t)X(t)+\lambda(t)Y(t)) + \gamma(T)X(T) \quad ,$$

subject to

$$X(t+1) = \Gamma(t)X(t) + \Lambda(t)Y(t) + \xi(t) \quad , \quad \text{for } 0 \leq t \leq T-1 \quad ,$$

$$\Phi(t)X(t) + \Omega(t)Y(t) \hat{=} \psi(t) \quad , \quad \text{for } 0 \leq t \leq T-1 \quad ,$$

$$0 \leq X(t) \quad , \quad 0 \leq Y(t) \leq \hat{Y}(t) \quad , \quad \text{for all } t \quad ,$$

with the initial state

$$X(0) = x^0 \quad ,$$

and with terminal requirement

$$X(T) \hat{=} x^* \quad .$$

The notation $\hat{=}$ for the constraints and terminal requirement refers either to $=$, to $\leq$ or to $\geq$, separately for each constraint. The coefficient matrix (corresponding to variables $X(t)$, $Y(t)$, and $X(t+1)$) and the right hand side vector of the integrated forestry-industry submodel of period t are given as

$$\begin{bmatrix} -\Gamma(t) & -\Lambda(t) & I \\ \Phi(t) & \Omega(t) & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \xi(t) \\ \psi(t) \end{bmatrix} \quad ,$$

respectively. Their structure has been illustrated in Figure 1 using the notation introduced in Sections 2 and 3.

	w	r	q	$\bar{q}$	ℓ	b	c	u^+	u^-	y	x	z^+	z^-	m	e	v	ℓ^+	ℓ^-	p^+	p^-	
	$-\alpha$							$-\eta$	ω												$= 0$
		$-I$								$-I$	U	$-I$	I								$= 0$
			$\delta-I$													$-I$					$= 0$
				$\bar{\delta}-I$												$-K$					$= 0$
					$-I$												$-I$	I			$= 0$
						$-I$													$\tau-I$		$= 0$
				$-\bar{\delta}$			$-I$									K	$-I$	I	$\tau-I$	I	$= B$
	G																				$\leq H$
								R^+	R^-												$\leq R$
									S	$-I$											$= 0$
											A			I	$-I$						$= 0$
											ρ										$\leq L$
			$-I$								Q										≤ 0
					$\mu-I$											μ					$\geq -\mu B$
			F	$\bar{\delta}$	ε						C				$-P$				$+I$	$-I$	$= -D$
Upper bounds:												Z^+	Z^-	M	E						

Figure 1. The constraint matrix $\begin{bmatrix} -\Gamma(t) & -\Lambda(t) \\ \Phi(t) & \Omega(t) \end{bmatrix}$, the right hand side vector $\begin{bmatrix} \xi(t) \\ \psi(t) \end{bmatrix}$, the state vector X(t), the control vector $\gamma(t)$, and the upper bound vector $\hat{Y}(t)$ for the submodel of period t of the integrated forestry--industry model.

5. APPLICATION TO THE FINNISH FOREST SECTOR

5.1 Implementation

Two versions of the integrated model were implemented for the SESAME system (Orchard-Hays 1978) (a large interactive mathematical programming system designed for an IBM/370 and operating under VM/CMS). The model generators are written using SESAME's data management extension, called DATAMAT. An actual model is specified by the data tableaux of the generator programs.

Our two versions have been designed for the Finnish forest sector. Both of them may have at most ten time periods each of which is a five year interval. In each case, the country is considered as a single region. The main differences between our small and large version are in the number of products, financial units, and the tree species considered in the forest. Table 1 shows the dimensions of the two models.

For the small version, the seven product groups in consideration are sawn goods, panels, further processed mechanical wood products, mechanical pulp, chemical pulp, paper and board, and converted paper products. For each group we consider a separate type of production capacity and labor force. In this small version, we have aggregated all production into one financial unit. Only one type of tree represents all tree species in the forests. The trees are classified into 21 age groups. Thus, the interval being five years, the oldest group contains trees older than 100 years. Two harvesting activities were made available: thinning and final harvesting. The main timber assortments in consideration are log and pulpwood.

The larger version has the following 17 product groups: sawn goods, plywood, particle board, fiberboard, three types of further processed mechanical products, mechanical pulp, Si-pulp, Sa-pulp, newsprint, printing and writing paper, other papers, paperboard, and three types of converted paper products. Again for each such group we have a separate type of production capacity as well as labor force. The production is aggregated into seven

Table 1. Characteristic dimensions of the small and the large versions of the Finnish forest sector model.

	Small version	Large version
Number of time periods *	10	10
Length of one period in years *	5	5
Number of regions	1	1
Number of tree species	1	3
Number of age groups for trees*	21	21
Harvesting activities*	2	6
Soil types	1	1
Harvesting and planting resources	1	1
Timber assortments	2	6
Production activities	7	17
Types of labor in the industry	7	17
Types of production capacity	7	17
Number of financial units	1	7
Number of rows in a ten period LP	520	2320
Number of columns in a ten period LP	612	3188

*The value may be specified arbitrarily by the model data. The numbers show the actual values being used.

financial units: saw mills, panels production (plywood, particle board, and fiberboard), further processing of primary mechanical wood products, mechanical pulp mills, chemical pulp mills, paper and board mills, and production of converted paper goods.

Three species of trees appear in the larger version: pine, spruce, and birch. For each of these we apply the same 21 age groups as in the small version. The two harvesting activities (thinning and terminal harvesting) and the two main timber assortments (log and pulpwood) are now considered separately for each of the three tree species.

The data for both of the versions of the Finnish model was provided by the Finnish Forest Research Institute. It is partially based on the official forest statistics (Yearbook of Forest Statistics 1977/1978) published by the same institute. Validation runs (which eventually resulted in our current formulation) were carried out by contrasting the model solutions with the experience gained in the preceeding simulation study of the Finnish forest sector by Seppälä, Kuuluvainen and Seppälä (forthcoming).

5.2 Numerical Examples

For illustrative purposes we will now describe a few test runs: two with the small version and one with the larger one. Most of the data being used in these experiments corresponds approximately to the Finnish forest sector. This is the case, for instance, with the initial state; i.e., trees in the forests, different types of production capacity, etc. Somewhat hypothetical scenarios have been used for certain key quantities, such as final demand, and price and cost development. Thus, the results obtained do not necessarily reflect reality. They have been presented only to illustrate a few possible uses of the model.

For each test run a ten (five year) period model was constructed. Labor constraints both for indsutry and for forestry were temporarily relaxed. At this stage, no further processing activity for mechanical wood products but one activity for converted paper products was considered. Both wood import and export were excluded, and pulp import to be used for paper production was allowed only in the larger version of the model. The assumed demand of wood products is given in Table 2. At the end of the planning horizon, we require that in each age group there is at least 80 percent of the number of trees initially in those groups. For production capacity a similar terminal requirement is 50 percent. Initial production capacity is given in Table 3 and the initial age distribution of trees in Figure 8 below.

For the first run the discounted sum of industrial profits (after tax) was chosen as an objective function. Such an objective may reflect the industry's behavior given the cost structure, price development, and other parameters. The results have been illustrated in Figures 2 through 7. The mechanical processing activities are limited almost exclusively by the assumed demand of sawn goods and panels. The same is true for converted paper products. However, both mechanical and chemical pulp produced is almost entirely used in paper mills, and therefore, the potential demand for export has not been exploited. Neither have the possibilities for exporting paper been used fully. As shown in Figure 5, paper export is declining sharply from the level of 5 million ton/year, approaching zero towards the end of the planning horizon. This is due to the stongly increasing production of converted paper products. The corresponding structural change of the production capacity of the forest industry over the 30 year period from 1980 to 2010 is given in Table 3. (The sudden decrease in production of panels and converted paper products is a "planning horizon effect" which often appears in dynamic LP solutions. Usually it is due to inappropriate accounting for the future in terminal conditions. For instance, in our case only a reasonable state was required at the end of the planning horizon, while an optimal stationary state might have been more appropriate.)

Table 2. Assumed annual demand of wood products in Runs 1 - 3.

Period	Sawn wood Mm^3/y	Panels Mm^3/y	Mech. pulp Mton/y	Chem. pulp Mton/y	Paper and board Mton/y	Converted paper prod. Mton/y
1980-84	7.0	1.7	.02	1.2	4.8	0.5
1985-89	7.5	2.0	.01	1.1	5.8	0.7
1990-94	8.0	2.2	.01	1.0	7.0	0.9
1995-99	8.8	2.5	.01	0.9	8.3	1.2
2000-04	9.3	2.8	.01	0.8	9.8	1.6
2005-09	9.7	3.2	.01	0.7	11.6	2.1
2010-14	10.2	3.6	.01	0.7	13.2	2.9
2015-19	10.7	4.1	.01	0.6	15.1	3.8
2020-24	11.2	4.6	.01	0.6	17.1	5.1
2025-29	11.6	5.2	.01	0.6	19.2	6.9

Table 3. Production capacity initially and in 2010 according to Runs 1 - 3.

Product	Production capacity				Unit
	Initial	Year 2010			
		Run 1	Run 2	Run 3	
Sawn wood	7.0	10.2	10.2	10.2	M m^3/year
Panels	1.7	3.6	3.6	3.6	M m^3/year
Mechanical pulp	2.2	1.9	2.2	0.5	M ton/year
Chemical pulp	4.0	4.3	5.8	5.0	M ton/year
Paper (and board)	6.2	6.2	7.3	8.7	M ton/year
Converted paper and board products	0.5	2.9	2.9	2.9	M ton/year

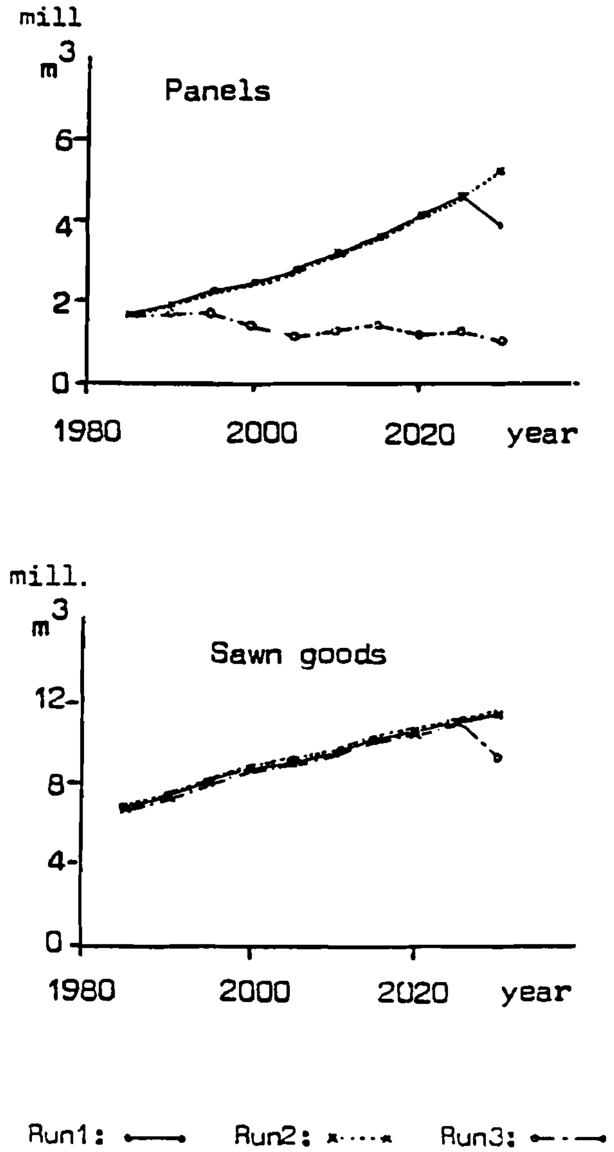


Figure 2. Annual production of sawn wood and panels (in millions of m^3 per year).

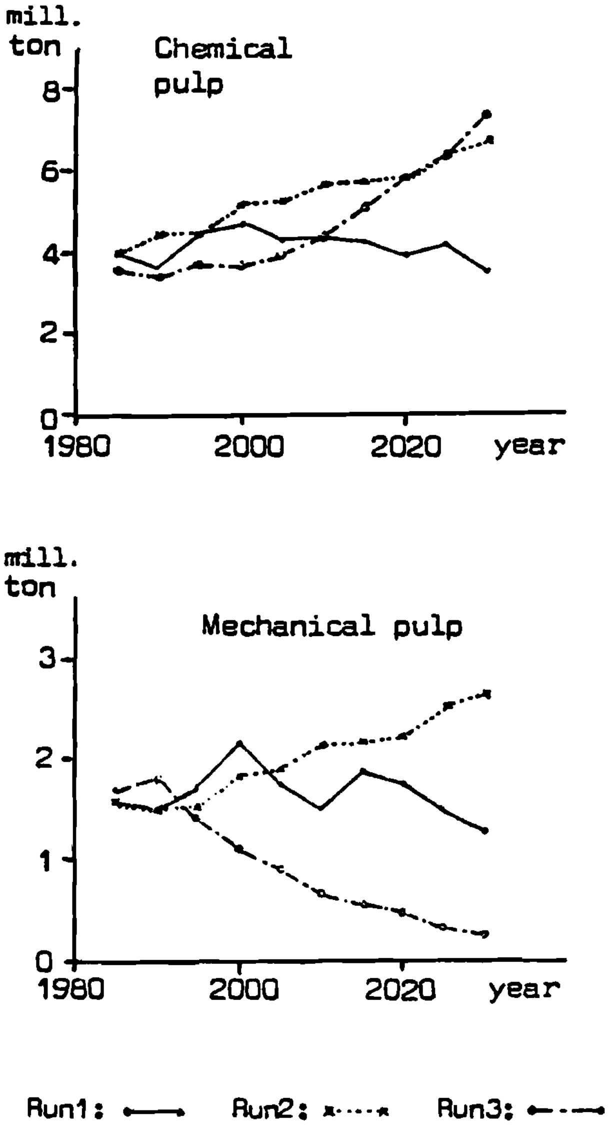


Figure 3. Annual production of pulp (in millions of ton per year).

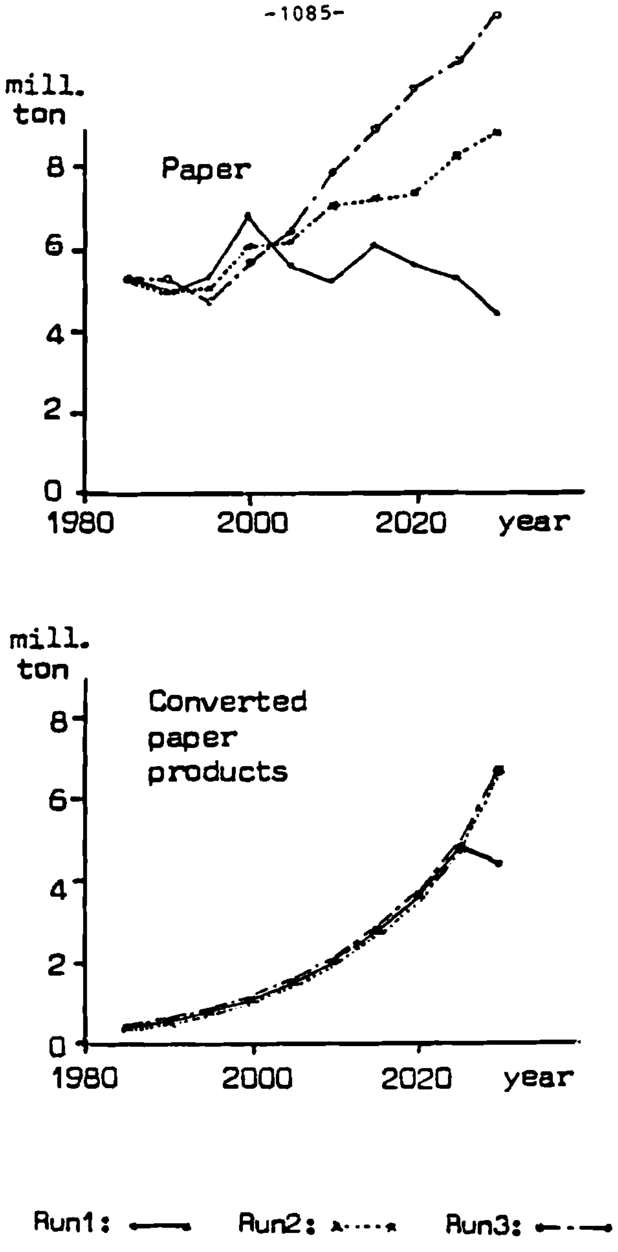


Figure 4. Annual production of paper and converted paper products (in millions of ton per year)

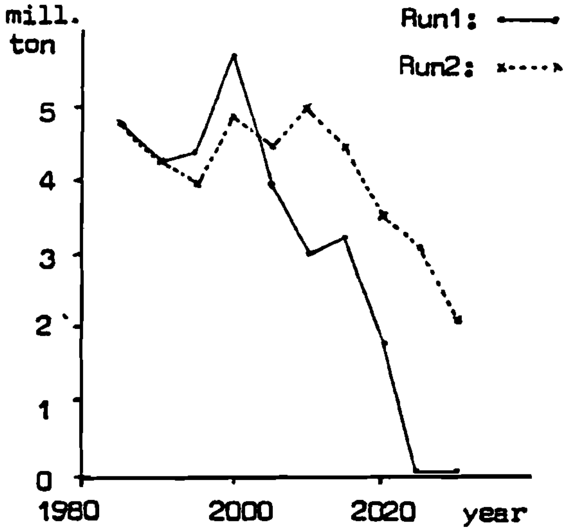


Figure 5. Paper export (in millions of ton per year)

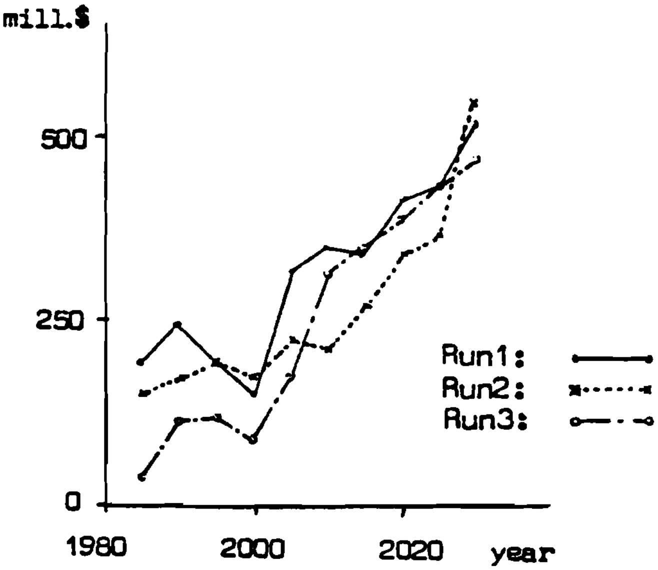


Figure 6. Industrial profit (in millions of dollars per year).

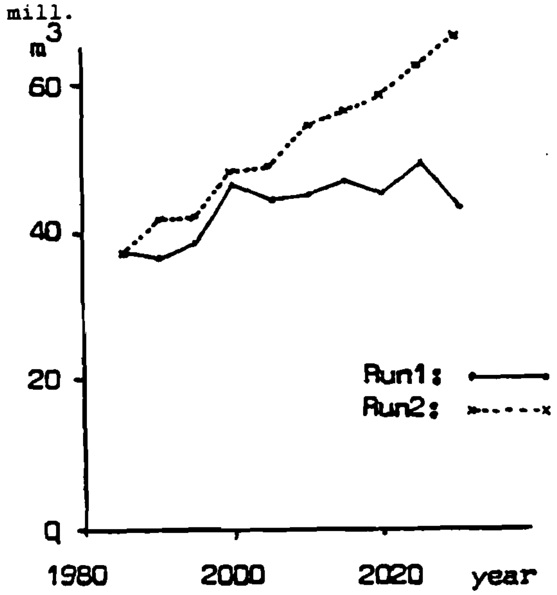


Figure 7. Industrial use of round wood (in millions of m^3 per year).

The use of wood has been shown in Figure 7. At the beginning the industrial use of wood increases from about 40 million m^3/year to the level of 45 million m^3/year and stays rather steadily there. According to Figure 6, the industrial profit increases from the annual level of .2 billion dollars towards the end of the planning horizon to around .5 billion dollars per year.

For the second run we have chosen the discounted sum of the increments of the forest sector to gross national product as an objective function. The results have been illustrated using dotted lines in the same Figures 2 through 7.

Compared with the previous case, there is no significant difference in the production of sawn goods, panels and converted paper products for which export demand again limits the production. However, there is a significant difference in pulp and paper production. Pulp (both mechanical and chemical) is now produced to satisfy fully the demand for export. Paper production is now steadily increasing from 5 million ton/year to nearly 9 million ton/year. Paper export is still declining again due to increasing use for the converting processes of paper products. Therefore, the export demand for paper is not fully exploited.

The bottleneck for paper production now is the biological capacity of the forests to supply wood. The use of round wood increases from about 40 million m^3/year to the level of 65 million m^3/year. The increase in the yield of the forests may be explained by the change in the age structure of the forests during the planning horizon. Such change over the period 1980-2010 has been illustrated in Figure 8.

We notice a significant difference in the wood use between these first two runs. We may conclude that in the first run (the profit maximization) the national wood resources are being used in an inefficient way; i.e., under the assumed price and cost structure the poor profitability of the forest industry results in an investment behavior which does not make full use of the forest resources.

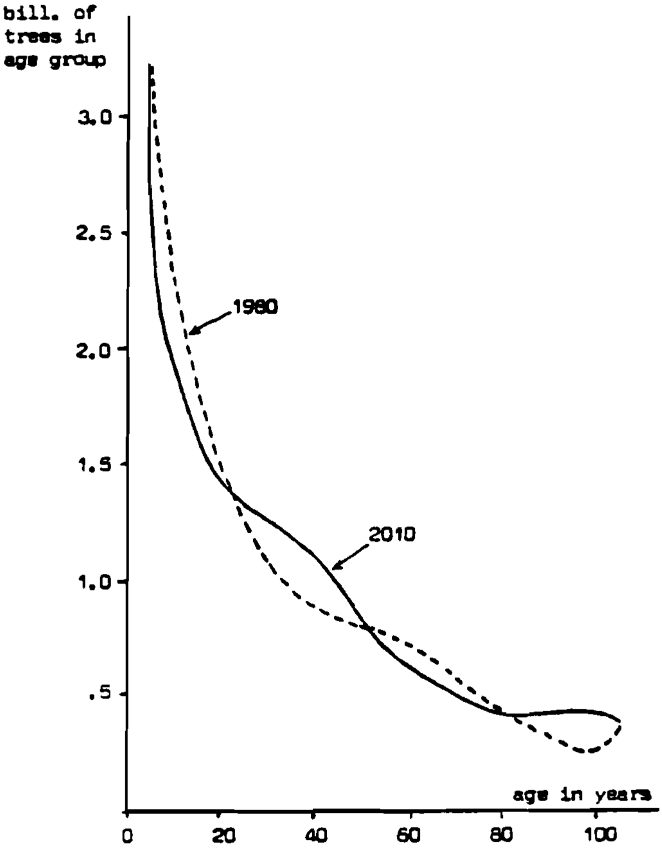


Figure 8. Age distribution of trees in 1980 and in 2010 according to Run2.

The third run is the same as the first one except that the larger version of the model was used and pulp import was allowed to be used in paper mills. The production of sawn goods and converted paper products, as described by broken lines in Figure 2, still meet the export demand. However, panel production is declining and it falls well below the level of the previous runs. The reason is that panel production is now considered as a separate financial unit which cannot afford to keep up its production capacity. Thus, an increase in panels production appears to be possible only if it is supported from other product lines. Similarly, the use of spruce for mechanical pulp appears unprofitable so that its production is declining. Production of Si-pulp (for which spruce pulpwood is used) grows steadily from 5 million ton/year to about 10 million ton/year. No spruce is used for Sa-pulp but both the use of pine and birch for Sa-pulp increase over time so that the total production of chemical pulp increases from about 3.5 million ton/year to the level of 7 million ton/year during the planning horizon. Thus chemical pulp production somewhat exceeds the amount produced in the first run.

Paper production in this third run exceeds the level obtained in both previous runs. The reason is that imported pulp is now allowed to be used in paper mills. (Note that in the second run, the raw wood supply was the limiting factor for paper production.) As a consequence, total paper production increased from 5 million ton/year to above 11 million ton/year. The share of newsprint is about one fifth and the share of printing paper one quarter. Only paperboard production appears to decline.

From the production curves of the primary uses of wood, i.e., sawn goods, panels and pulp, we may conclude (comparing with the second run) that wood resources are again being used inefficiently. It appears that, under the assumed price and cost structure, fiber (pulp in particular) import to be used as raw material in paper mills is more profitable than the use of domestic wood raw material.

6. SUMMARY AND POSSIBLE FURTHER RESEARCH

We have formulated a dynamic linear programming model of a forest sector. Such a model may be used for studying long-range development alternatives of forestry and forest based industries at a national and regional level. Our model comprises of two subsystems, the forestry and industrial subsystem, which are linked to each other through the raw wood supply from forestry to the industries. We may also single out static temporal submodels of forestry and industries for each interval (e.g., for each five year period) considered for the planning horizon. The dynamic model then comprises of these static submodels which are coupled with each other through inventory-type of variables; i.e., through state variables.

The forestry submodel describes the development of the volume and the age distribution of different tree species within the nation or its subregions. Among others, we account for the land available for timber production and the labor available for harvesting and planting activities. Also ecological constraints, such as preserving land as a watershed may be taken into account.

In the industrial submodel we consider various production activities, such as saw milling, panel production, pulp and paper milling, as well as further processing of primary products. For a single product, alternative production activities employing, for instance, different technologies, may be included. Thus, the production process is described by a small Leontief model with substitution. For the end product demand an exogenously given upper limit is assumed. Some products, such as pulp, may also be imported into the forest sector for further processing. Besides biological supply of wood and demand for wood based products, production is restricted through labor availability, production capacity, and financial resources. Availability of different types of labor (by region) is assumed to be given. The development of different types of production capacity depends on the initial situation in the country and on the investments which are endogeneous decisions in the model. The production activities are grouped into financial units to which the respective production capacities belong. The investments are made within the financial resources of such units. External financing is made available to each unit up to a limit which is determined by the realization value of that unit. Income tax is assumed proportional to the net income of each financial unit.

The structure of the integrated forestry-forest industry model is given in the canonical form of dynamic linear programs for which special solution techniques may be employed. (See, for instance, Kallio and Orchard-Hays 1979, Propoi and Krivonozhko 1978). Objectives related to gross national product, employment and profit for industry as well as for forestry have been formulated. Terminal conditions (i.e., values for the state variables at the end of the planning horizon) have been proposed to be determined through an optimal solution of a stationary model for the forest sector.

Two verisons of the Finnish forest sector model have been implemented for the interactive mathematical programming system called SESAME (Orchard-Hays 1978). Both versions are ten period models with each period five years in length. In neither case has the country been divided into subregions. The main difference between these versions are in the number of production activities and in the number of financial units. No distinction has been made between the tree species in the smaller version whereas pine, spruce, and birch are considered explicitly in the larger one. The complete model amounts to 520 rows and 612 columns in the smaller case, and to 2320 rows and 3188 columns for the larger model.

A few numerical runs have been presented to illustrate possible use of the model. Both the discounted industrial profit and the discounted increment to the GNP were used as objective functions. The results obtained illustrate a case where the internal wood price and wage structure results in a rather poor profitability for the forest industries. This in turn amounts to an investment behavior which provides insufficient capacity for making full use of the wood resources.

However, because of somewhat hypothetical data used for some key parameters, no conclusions based on these runs should be made on the Finnish case.

The purpose of this work has been the formulation, implementation and validation of the Finnish forest sector model. Natural continuation of this research is to use the model for studying some important aspects in the forest sector. For instance, the influence of alternative scenarios of the energy price and the world market prices for wood products would be of interest. Furthermore, the studies could concentrate on employment and wage rate questions, on labor availability restrictions and productivity, on new technology for harvesting and wood processing, on the influence of inflation and alternative taxation schemes, on land use between forestry and agriculture, on site improvement, on ecological constraints, on the use of wood as a source of energy, etc. Given the required data, such studies can be carried out relatively easily.

Further research requiring a larger modeling effort may concentrate on regional economic aspects, on linking the forest sector model for consistency to the national economic model, and on studying the inherent group decision problem for controlling the development of the forest sector. The first of these three topics requires a complete revision of our model generating program and, of course, the regionalized data. The second task may be carried out either by building in the model a simple input-output model for the whole economy where the non-forest sectors are aggregated up to ten sectors. Alternatively, our current model may be linked for consistency to an existing national economic model. The group decision problem has been proposed to be analyzed, for instance, using a multicriteria optimization approach (Kallio, Lewandowski, and Orchard-Hays forthcoming) which is based on the use of reference point optimization (Wierzbicki 1979).

APPENDIX: NOTATION

Indices

a, a'	age group of trees (range 1,..., N)
d	type of forest land
g	type of resource for forestry activities
h	harvesting activity
i	production activity (of the forest industries)
j	industrial product
k	timber assortment
n	planting activity
s	tree species
t	time period (range 1,..., T)

State and control variables

$b(t)$	stockholders equity at the beginning of period t
$b^0 = b(0)$	initial level of stockholders equity
$c(t)$	cash (and receivables)at the beginning of period t
$c^0 = c(0)$	initial amount of cash
c^*	terminal requirement for cash
$e(t) = \{e_j(t)\}$	export (and sales outside the forest sector) of forest products during period t

$\ell(t)$	beginning balance of external financing for period t
$\ell^0 = \ell(0)$	initial balance of external financing
ℓ^*	terminal requirement for external financing
$\ell^+(t)$	drawings of debt during period t
$\ell^-(t)$	repayments made during period t
$m(t) = \{m_j(t)\}$	import of forest products during period t
$p^+(t)$	profits of period t
$p^-(t)$	(financial) losses of period t
$q(t)$	production capacity at the beginning of period t
$q^0 = q(0)$	initial level of production capacity
q^*	terminal requirement for production capacity
$\bar{q}(t)$	fixed assets at the beginning of period t
$\bar{q}^0 = \bar{q}(0)$	initial value of fixed assets
$\bar{q}^*$	terminal requirement for fixed assets
$r(t) = \{r_k(t)\}$	timber assortments inventory at the beginning of period t
$r^0 = r(0)$	initial level of timber assortments inventory
r^*	terminal requirement for timber assortments inventory
$u^-(t) = \{u_h^-(t)\}$	level of harvesting activities during period t
u^-	level of harvesting in a stationary solution
$u^+(t) = \{u_n^+(t)\}$	level of planting activities during period t
u^+	level of planting in a stationary solution
$v(t)$	level of investments (in physical units) during t
$w(t) = \{w_s(t)\} = \{w_{sa}(t)\}$	number of trees at the beginning of of period t
$w^0 = w(0)$	initial number of trees
w^*	terminal requirement for the number of trees
w	number of trees in a stationary solution

$x(t)$	level of production activities during period t
$X(t)$	state vector at the beginning of period t
$X^0 = X(0)$	initial state
X^*	requirement for terminal state
$y(t) = \{y_k(t)\}$	supply of timber assortments during period t
$Y(t)$	level of control activities during period t
$z^+(t)$	import of timber assortments during period t
$z^-(t)$	export of timber assortments during period t

<u>Parameters</u>

$\alpha^s_{aa'}(t)$	ratio of trees of species s and in age group a that will proceed to age group a' during period t
$\alpha,\ \alpha^s$	matrices of coefficients $\alpha^s_{aa'}(t)$
$\beta(t)$	discounting factor
$\gamma(t)$	objective function coefficients for the state vector X(t)
$\Gamma(t)$	coefficient matrix for the state vector X(t) in the state equation
δ	physical depreciation rates
$\overline{\delta}(t)$	financial depreciation rates
Δ	age interval in an age group of trees (e.g., five years)
$\epsilon(t)$	interest rates for external financing
$\psi(t)$	right hand side vector of constraints for period t
$\Phi(t)$	coefficient matrix for the state vector X(t) in constraints for period t
η	matrix relating planting activities to the increase in the number of trees
$\lambda(t)$	objective function coefficients for the control vector Y(t)
$\Lambda(t)$	coefficient matrix for the control vector Y(t) in the state equation
ω	matrix relating harvesting activities to the decrease in the number of trees
$\Omega(t)$	coefficient matrix for the control vector Y(t) in constraints for period t
$\rho(t)$	labor requirement for different production activities
$\tau(t)$	tax factors for the industries during period t

$\mu(t)$	upper limit to external financing as a percentage of total assets less short term liabilities
$\xi(t)$	right hand side vector for the state equation of period t
$A(t)$	input-output matrix for the forest industries
$B(t)$	stock issued during period t
$C(t)$	direct unit production costs
$D(t)$	exogeneously given costs
$E(t)$	upper bound on demand of forest products
$F(t)$	fixed costs (per unit of production capacity)
$G = (G^s_{ad})$	land requirement of the species in various age groups
$H(t)$	land available for forests
I	identity matrix
$J^-(t)$	objective function coefficients for harvesting activities (an example)
$J^+(t)$	objective function coefficients for planting activities (an example)
$K(t)$	investment costs per capacity unit
$L(t)$	labor available for forest industries
$M(t)$	upper limit on import of forest products
N	number of age groups for trees
$P(t)$	prices of forest products
$Q(t)$	matrix of capacity requirements for production activities
$R(t) = \{R_g(t)\}$	resources available for forestry activities
$R^+(t) = \{R^+_{gn}(t)\}$	resource usage of planting activities
$R^-(t) = \{R^-_{gh}(t)\}$	resource usage of harvesting activities
$S(t)$	matrix transforming the trees harvested into volumes of timber assortments
T	number of time periods
U	usage of timber assortments by various production activities

REFERENCES

Barros, O., and A. Weintraub (1979) Planning for Vertically Integrated Forest Industry. Presented at Tenth International Symposium on Mathematical Programming, Montreal.

Dantzig, G. (1963) Linear Programming and Extensions. Princeton, N.J.: Princeton Univ. Press.

Dantzig, G. (1974) Determining Optimal Policies for Ecosystems. Technical Report 74-11. Stanford, California: Department of Operations Research, Stanford University.

Jackson, B. (1974) Forest Products in the United Kingdom Economy. Ph.D. thesis, Department of Forestry, Oxford University.

Jegr, K., E. Miller, and K. Thompson (1978) PAPRISIM 1 - A Dynamic Model of the Canadian Pulp and Paper Industry. Pointe Clair, P.Q.: Pulp and Paper Research Institute of Canada.

Kallio, M., W. Orchard-Hays, and A. Propoi (1979) Linking of Optimization Models. WP-79-83. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Kallio, M., and W. Orchard-Hays (1979) Experiments with the Reduced Gradient Mehtod for Linear Programming. WP-79-84. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Kallio, M., A. Lewandowski, and W. Orchard-Hays (1980) Application of a Multicriteria Optimization Method Using Reference Objectives. Laxenburg, Austria: International Institute for Applied Systems Analysis. Forthcoming.

Kilkki, P., K. Kuusela, and M. Siitonen (1977) Timber production programs for the forestry board districts of southern Finland. Folia Forestalia 307, The Finnish Research Institute (in Finnish).

Newnham, R. (1975) LOGPLAN, A Model for Planning Logging Operations. Information Report FMR-77. Ottawa: Forest Management Institute, Canadian Forestry Service.

Navon, Daniel I. (1971) Timber RAM--A Long Range Planning Method for Commercial Timberlands Under Multiple-Use Management. USDA Forest Service Research Paper PSW-70. Berkeley, California: Pacific Southwest Forest and Range Experiment Station.

Näslund, B. (1969) Optimal rotation and thinning. Forest Service 15.

Orchard-Hays, W. (1978) Anatomy of a mathematical programming system. Design and Implementation of Optimization Software, edited by H. Greenberg. Netherlands: Sijthoff and Noordhoff.

Propoi, A., and V. Krivonozhko (1978) The Simplex Method for Dynamic Linear Programs. RR-78-14. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Randers, J. (1976) A System Dynamic Study of the Transition From Ample to Scarce Wood Resources. Hanover, N.H.: Publications of Resource Policy Center, Dartmouth College.

Rorres, C. (1978) A linear programming approach to the optimal sustainable harvesting of a forest. Journal of Environmental Management 6.

Seppälä, H., J. Kuuluvainen, and R. Seppälä. The Finnish Forest Sector at the Turning Point (in Finnish). Forthcoming.

Steuer, R. and A. Schuler (1978) An interactive multiple-objective linear programming approach to a problem in forest management. Operations Research 26.

Wardle, P. (1965) Forest management and operational research: a linear programming study. Management Science 11.

Ware, G., and J. Clutter (1977) A mathematical programming system for the management of industrial forests. Forest Science, 17.

Weintraub, A., and D. Navon (1976) A forest management planning model integrating silvicultural and transportation activities. Management Science 22.

Wierzbicki, A. (1979) A Methodological Guide to Multiobjective Optimization. WP-79-122. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Williams, D. (1976) Integrating Stand and Forest Models for Decision Analysis. Ph.D. thesis, Department of Forestry, University of British Columbia.

Yearbook of Forest Statistics 1977/1978. Official Statistics of Finland XVII A:10, The Finnish Forest Research Institute.

OPERATIONAL USE OF MULTIPERIOD LP MODELS FOR PLANNING AND SCHEDULING

A.T. Langeveld

Koninklijke/Shell-Laboratorium, Amsterdam
(Shell Research B.V.)

In this paper we concentrate on the operational use of large multiperiod LP models for the planning and scheduling of plant operations, and the problems it poses for the methodology of large-scale linear programming. A number of requirements from the operational environment are listed. On the basis of an example (refinery planning/scheduling) it is shown that the structure of multiperiod models can be employed to reduce computation times. It is argued that modern electronic equipment can facilitate the input and output of large LP models, especially for non-LP specialists. An area of algorithmic research is indicated.

Operational environment

Linear programming can be a very helpful tool in the planning and scheduling of plant operations. Multiperiod LP models can be constructed to express the dynamic relationships between the operations of the various processing plants. In actual operations these models will have to be used on a day-to-day basis by people who are not familiar with the intricacies of large-scale LP. In such an environment special capabilities are required from the LP plant of the planning/scheduling systems. Here we list a number of these operational requirements:

- input and output must be understandable to the planner/scheduler: it must be possible for him to check easily the input data and judge the LP output, preferably in his own language and terminology;
- there is a need to adapt the LP outcome to satisfy operational requirements that can not be specified within the LP framework;

- successive LP runs are usually related to each other: either the new run is a modification of the previous run; or the new run covers a time interval largely overlapping the time interval of the previous run (moving time frame);
- the turn-around time of the LP must be short: in day-to-day use answers should be available within an hour;
- cost must be low;
- the system will have to be run on different computers.

Structure of multiperiod LP models

Efficient solution of large models requires an analysis of their characteristics. We illustrate such an analysis for a typical multiperiod refinery scheduling model.

For the daily scheduling of refinery operations use is made of refinery models describing the processing and intermediate storage of hydrocarbons ("materials"). Processing units transform hydrocarbon streams into other hydrocarbon streams through a number of "modes of operation" (see Waasdorp and Van Nes, 1975), each mode being specified by its maximum throughput and the yield distribution. The scheduler should determine for each day and each processing unit which mode is run. A severe restriction in the operation of the processing units is the limited tankage for intermediate hydrocarbon streams. Variables in the models are the daily throughputs of the various modes of the processing units; constraints describe the tankage limitations for each material and the throughput restrictions for each mode and each processing unit. The resulting multiperiod LP model then has the following structure:

$$\left.\begin{array}{lcllc} l_1 & < & Ax_1 & < & u_1 \\ l_2 & < & Ax_1 + Ax_2 & < & u_2 \\ l_3 & < & Ax_1 + Ax_2 + Ax_3 & < & u_3 \\ \vdots & & & & \vdots \\ l_N & < & Ax_1 + Ax_2 + Ax_3 + \ldots + Ax_N & < & u_N \end{array}\right\} \text{material tankage constraints}$$

$$\left.\begin{array}{llllll}
L_1 < & Bx_1 & & & & < U_1 \\
L_2 < & & Bx_2 & & & < U_2 \\
L_3 < & & & Bx_3 & & < U_3 \\
\vdots & & & & & \vdots \\
L_N < & & & & Bx_N & < U_N
\end{array}\right\} \begin{array}{c}\text{unit throughput}\\ \text{constraints}\end{array}$$

In this formulation x_t represents the vector of variables in time period t; the matrix A describes the production capabilities of the modes. Usually, the objectives of such models relate to cumulative production: maximize or minimize $\sum_{t=1}^{N} C^T x_t$, where C is a known vector which is constant for each time period; C^T denotes the transposed of C.

In such models one can introduce cumulative variables:

$$y_t = \sum_{j=1}^{t} x_j$$

(defined in the usual way: componentwise). The model structure then becomes:

$$\left.\begin{array}{llllll}
l_1 < & Ay_1 & & & & < u_1 \\
l_2 < & & Ay_2 & & & < u_2 \\
l_3 < & & & Ay_3 & & < u_3 \\
\vdots & & & & & \vdots \\
l_N < & & & & Ay_n & < u_N
\end{array}\right\} \begin{array}{c}\text{material tankage}\\ \text{constraints}\end{array}$$

$$\left.\begin{array}{llllll}
L_1 < & By_1 & & & & < U_1 \\
L_2 < & -By_1 + By_2 & & & & < U_2 \\
L_3 < & & -By_2 + By_3 & & & < U_3 \\
\vdots & & & & & \vdots \\
L_N < & & & & -By_{N-1} + By_N & < U_N
\end{array}\right\} \begin{array}{c}\text{unit throughput}\\ \text{constraints}\end{array}$$

Such a model might be significantly quicker to solve, because the density of the matrix of the cumulative model is usually less than the density of the "standard" formulation. Typical examples are shown in Table I, where we have collected some of our computational experiments. It shows that the density of the cumulative model is indeed less than

TABLE I

COMPARISON OF MULTIPERIOD MODELS WITH NON-CUMULATIVE AND CUMULATIVE VARIABLES
(USE WAS MADE OF MPSX/370 ON IBM 370/168)

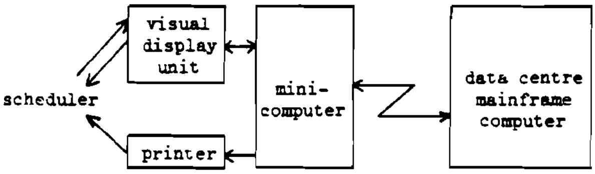
Model size			Standard formulation			Cumulative formulation		
Number of periods	Rows	Columns	Density (%)	Iterations	Solution time (minutes)	Density (%)	Iterations	Solution time (minutes)
10	838	330	2.04	940	0.81	0.37	201	0.23
20	1708	660	1.85	273	2.15	0.18	623	0.98
30	2578	990	1.79	243	5.73	0.12	784	1.67
40	3448	1320	1.76	294	12.42	0.09	876	2.56
13	2459	3042	0.34	380	2.47	0.12	449	1.70

that of the standard formulation, resulting in a reduction of the computation time; surprisingly, the number of iterations was often higher for the cumulative models.

Another characteristic of such a multiperiod LP model is that there exist many optimal solutions: an exchange of values of non-cumulative variables between two periods does not change the value of the objective function (due to the nature of this function) and is often allowed by the constraints, especially when the time periods are adjacent. This observation plays a crucial role in the adaptation of the LP outcomes, as we shall see later.

Use of modern equipment

To satisfy the first operational requirement for an LP-based scheduling system (user-understandable I/O) we have built a minicomputer system to handle the input and the output of the LP. This system has the following communication structure:



The scheduler specifies the data to the minicomputer using the visual display unit, either by filling in forms on the display or by calling data already stored in the minicomputer. The information on the screen relates to the scheduler's language, without LP jargon. The schedule can easily check the data, if he finds it necessary, using VDU or printer. The minicomputer then constructs the A and B matrices, the right and left hand sides of the LP matrix and the objectives (if any). Via telephone lines this information is sent to a data centre computer (the minicomputer acts as a remote batch terminal), where the LP matrix is constructed and the LP run is carried out. The results are sent back to the minicomputer and are translated into the terminology of the scheduler and presented on his VDU or printer.

The minicomputer system can thus be seen as a tailor-made matrix generator/report writer. We have found this system very convenient for the interactive process of specifying LP input data for multiperiod scheduling models.

Adaptation of LP results

Usually the LP models do not describe all the operational requirements of the refinery. For instance, one would like to keep the number of mode switches on a processing unit low; or a particular operation could better start during the day than in the night. Such requirements are not easy to model within the LP framework.

Fortunately, as explained earlier, these multiperiod LP models contain many optimal solutions, of which the LP code will only present a few basic optimal solutions. In our system the scheduler can use an optimal LP solution as a start; within the set of optimal LP solutions he can try to find another optimal solution which he thinks most appropriate in view of the operational requirements. On the minicomputer we have developed algorithms to allow the scheduler to adapt the LP outcome without becoming unfeasible and also retaining optimality. The final schedule will not necessarily be a basic solution; on the contrary, in view of the uncertainties of the future, the scheduler will prefer schedules than can withstand the inevitable changes in circumstances as much as possible*.

In the full cycle of input specification, LP run, output checking, and adaptation of LP results the last activity takes most of the time (50 %).

Algorithm research

The above approach has a number of practical difficulties:

- the cost of running large LP's during the day in a data centre is high: one has to run on high priority or obtain a reasonable turn-around time and therefore the highest tariffs apply;

* Work has been done at our laboratories on the flexibility properties of solutions of LP models. This work is reported in a Ph.D. Thesis (Van der Vet, 1980).

- low-priority runs are usually carried out during the night; if a run fails (e.g. due to wrong input) one would lose a full 24 hours: turnaround becomes too long;
- since data centres have many customers, it will happen that turnaround times exceed the required times despite the high priority;
- when LP-based systems as described above are to be used by different refineries, then different data centres will be used and different mainframe computers will have to be accessed. So far we have experience with IBM computers using MPSX as LP package, and with Univac computers using FMPS.

The question arose, of course, whether it would be possible to solve the multiperiod LP on the on-site minicomputer. In the current state of minicomputer technology standard LP codes (based on the simplex method) are available which are capable of solving problems up to 800 variables/ constraints*. It is surprising that there is still a need for LP codes and algorithms for solving relatively big LP problems on relatively small computers: 20 years after the publication of decomposition methods the same problem area that generated these methods still exists despite the developments in computer technology.

One of the consequences of the day-to-day use of these models is that most runs are related to each other (slightly different data, moving timeframe). At each new LP run the scheduler faces the situation in which he has available a satisfying schedule (from the operational point of view), which is slightly unfeasible for his new LP run. Use of LP brings him back to some undesirable basic solution for which he has again to spend a lot of time to bring it up to operational standards. Therefore, we believe that the scientific world should pay attention to the following area of algorithmic research: it would be useful to have algorithms that accept a slightly unfeasible nonbasic starting point and find an optimal and feasible solution in the "vicinity" of the starting point.

* Recently we became aware of a program called LAMPS, developed by J. Forrest for 32-bit minicomputers, which is claimed to solve LP problems with several thousands of variables and constraints within two hours.

We have started work on approximating (iterative) techniques for solving LP problems (Agmon; Motzkin and Schoenberg; Oettli). Our first results, even on small problems, were rather disappointing: convergence is very slow. However, using simple extrapolation techniques we speeded up convergence by a factor of ten, but a lot of work has as yet to be done before this method can be of practical value. Similarly, we have experimented with Khachian's method (1979) and found similar convergence characteristics.

Conclusions

A class of large-scale linear programming models are multiperiod models for the planning and scheduling of plant operations. Such multiperiod models have special characteristics which can be employed in their solution. We have shown that model formulation (cumulative versus non-cumulative variables) plays an important role in the solution efficiency. Further work could be done on special algorithms to employ the multiperiod structure.

For operational use of large-scale LP models it is of vital importance that easy means exist to speficy and check the LP input and to judge the LP output. Instead of the existing type of matrix generators/ report writers use could be made of present-day electronic equipment such as visual display units, minicomputers and data communication links. In our example of multiperiod LP models for refinery scheduling we have shown that this route is certainly viable.

Equally important in the operational use of large-scale LP is the fact that the LP result may not be the desired answer to the practical question. The LP solution is very often only a starting point for further manipulation, and therefore the LP step needs to be reliable and to take a minor portion of the time for the total activity.

We have indicated that there is still a need for algorithms and codes to solve LP problems on relatively small computers. Such codes should be as machine-independent as possible. In the light of the ever-increasing power of computers one has to be prepared for a situation in which a particular application now running on a small dedicated computer will eventually be switched to a more powerful dedicated computer in which standard LP techniques can be applied. This indicates that the input and output of LP codes need to be standardized, even for special codes on small computers.

As multiperiod planning models are often run on a regular basis with a shifting timeframe and slightly modified data (as more precise data on the future become available) there is a need to use the lastly obtained adapted LP solution as the basis for the new run. Currently, there are hardly any techniques to deal with situations in which one would like to find a non-basic optimal feasible solution close to an arbitrary slightly unfeasible starting point.

Our work on approximating (iterative) LP techniques (Agmon; Motzkin and Schoenberg; Oettli; Khachian) shows that a lot of work has as yet to be done before such methods become of practical value.

June 1980

PB.80.065
HH/TV

References

1. G.G. Waasdorp and A.G. van Nes, "Shell system aids plant operation", Oil and Gas Journal, 73 (1975) no. 9, pp. 124-126.
2. R.P. van der Vet, "Flexible solutions to systems of linear equalities and inequalities", Thesis, 1980, Technical University Eindhoven, Netherlands.
3. S. Agmon, "The relaxation method for linear inequalities", Canadian Journal of Mathematics, 6 (1954) pp. 382-392.
4. T.S. Motzkin and I.J. Schoenberg, "The relaxation method for linear inequalities", Canadian Journal of Mathematics, 6 (1954) pp. 393-404.
5. W. Oettli, "An iterative method, having linear rate of convergence for solving a pair of dual linear programs", Mathematical Programming, 3 (1972) pp. 302-311.
6. L.G. Khachian, "A polynomial algorithm in linear programming", Doklady Akademii Nauk SSSR, 244 (1979) pp. 1093-1096.

A COLUMN GENERATION/NESTED DECOMPOSITION ALGORITHM FOR DYNAMIC INPUT/OUTPUT MODELS*

Yves Smeers

CORE
Université Catholique de Louvain
Louvain-la-Neuve

We consider dynamic input/output models for which it is desired to include a refined representation of some of the sectors. A general modeling approach is proposed based on process representations developed in the field of energy flow. Nested decomposition is then used as a first step to generate problems of smaller size. It is then shown how the process representation adopted in the model allows one to use column generation to reduce again the size of the problems generated by nested decomposition.

*Research supported by the Belgian Department of Science Policy under contract E/I/3.

INTRODUCTION

We consider the following dynamic input output model

Program P

$$\text{Max} \sum_{t=1}^{T} U(d_t), \tag{1.1}$$

s.t. $$AX_t + BY_t + d_t = 0 \tag{1.2}$$

$$0 \leq X_t \leq Z_t \tag{1.3}$$

$$Z_{t+1} = \Lambda Z_t + Y_{t+1}. \tag{1.4}$$

where - $\sum_{t=1}^{T} U(d_t)$ is a utility function with respect to t and U is a nondecreasing function of d_t,

- A is a matrix of intersectoral technical coefficients,
- B is an investment coefficient matrix,
- X_t and Y_t are respectively the activity and investment levels of the different sectors of the economy in period t,
- Z_t is the capital stock vector of the various sectors of the economy in period t.

Models of this type are well known and usually take much more complicated forms than the one considered here. In this paper we take up the particular case where it is desired to detail in the model the representation of several sectors of the economy. This problem is commonly encountered in environmental and energy planning problems [1] [4] [11] where structural changes are expected in various sectors of the economy. Expanding the representation of some sectors of the economy usually leads to rather complex and large models which may be difficult to solve. It is the purpose of this paper to propose a systematic modeling approach for that problem as well as a special purpose algorithm for handling the resulting model. The discussion is presented on a model derived from the simple input/output model (P); the reader can easily convince himself that the procedure remains valid for more complex models including various constraints such as import export balance, employment objectives, saving formation ... .

In the following we shall assume that the set of sectors of the economy is partitionned in two subsets E and NE, where E designates the set of sectors for which we want to adopt a more refined technological description and NE those for which we accept the input output representation. This partitioning was already introduced and exploited by several authors before ([2] [9]) in the context of energy modeling. We shall denote by Z^{NE}, X^{NE} and Y^{NE} the vectors formed by the NE components of Z, X and Y respectively. A detailed representation of the E sectors will usually require the introduction of additional goods compared to the input output representation. We shall assume in the following that this extension has been made. We then define

$$A^{\cdot,NE} = \begin{pmatrix} A^{NE,NE} \\ A^{E,NE} \end{pmatrix} \text{ and } B^{\cdot,NE} = \begin{pmatrix} B^{NE,NE} \\ B^{E,NE} \end{pmatrix}$$

as the matrices of the vectors associated with X^{NE} and Y^{NE} respectively. The submatrices $A^{NE,NE}$ and $B^{NE,NE}$ are directly extracted from A and B respectively; the matrices $A^{E,NE}$ and $B^{E,NE}$ are obtained by expanding the corresponding submatrices of A and B to take into account the additional goods introduced in the model when disaggregating the E sectors. This expansion is illustrated on figure 1.

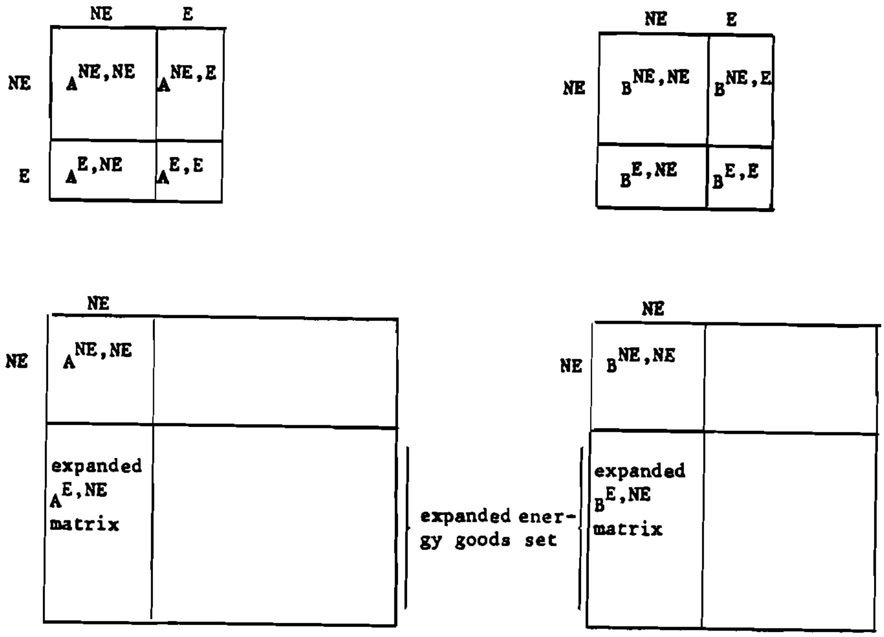


Fig. 1 : Decomposition and expansion of the A and B matrices

In the rest of this paper we shall adopt the following representation of the E subsystems. To each E sector i we associate the set K_i of the equipments constituting the sector. By definition we shall say that an equipment is characterized by a unique capacity variable. Plants that do not satisfy that condition will have to be disaggregated in such a way that the assumption is verified. An example of this situation is given by pumping storage in power generation which needs to be disaggregated in two equipments : the reservoir and the reversible pumps.

Let $S_{ik}(Z_{ik})$ denote the production set corresponding to a capacity Z_{ik} of the equipment k in sector i. We shall assume $S_{ik}(Z_{ik})$ to be described by a set of linear inequalities of the form

$$S_{ik}(Z_{ik}) \equiv \{\xi_{ik} \mid G_{ik}\, \xi_{ik} < H_{ik}\, Z_{ik}\}, \tag{1.6}$$

where G_{ik} and H_{ik} are respectively a matrix and a vector and ξ_{ik} is the vector of goods (E and NE) produced and consumed by the equipments. In order to simplify the notation, we assume that the non-negativity constraints on the ξ have been included in the definition of $S_{ik}(Z_{ik})$.

Using this notation we shall define the following expanded form of the dynamic I/O model.

<u>Program P^c</u>

$$\text{Max} \sum_{t=1}^{T} U(d_t), \tag{1.7}$$

s.t. $$A^{\cdot,NE} X_t^{NE} + \sum_{ik} \xi_{tik} + B^{\cdot,NE} Y_t^{NE} + \sum_{ik} B_{ik}\, Y_{tik} + d_t = 0 \tag{1.8}$$

$$\xi_{tik} \in S_{ik}(Z_{tik}),\ k \in K_i \text{ and } i \in E \tag{1.9}$$

$$0 < X_t^{NE} < Z_t^{NE} \tag{1.10}$$

$$Z_{t+1} = \Lambda Z_t + Y_{t+1}, \tag{1.11}$$

where - B_{ik} is the vector of goods (E and NE) consumed by an investment of a unit capacity in plant k of sector i and Y_{tik} denotes the quantity invested in that plant in period t.

- Λ is a matrix which takes into account the obsolescence of the capital stock of the NE sectors and the technical service life of the E sector equipments.

It may be that for some applications the size of the problem is sufficiently large to preclude the treatment of the problem by a straight simplex algorithm. In the following we present an algorithm for dealing with this situation.

2. PRELIMINARY REMARKS : A REMINDER OF NESTED DECOMPOSITION

We consider the dynamic problem P^c defined precedingly. As in many planning models, the variables of the problem can be categorized in two groups namely operations and capacity variables. Operations variables relative to a period t only appear in constraints (1.8) (1.9) and (1.10) which involve variables of that period. Capacity variables appear among other things in constraint (1.11) which involve variables of the two successive periods. The set of non zero elements of the constraint matrix will thus exhibit a classical staircase structure. Denoting by Ω_t the set represented by the constraints (1.8) to (1.10) and by x_t the vector $(X_t^{NE}, (\xi_{tik};$ $k \in K_i, i \in E), d_t)$. One can write the problem in the form

Program SC

$$\text{Max} \sum_{t=1}^{T} V(x_t) \tag{2.1}$$

$$- Y_1 + Z_1 = Z_0 \tag{2.2}$$

$$- \Lambda Z_{t-1} - Y_t + Z_t = \mathcal{O},\ t = 2, \ldots, T \tag{2.3}$$

$$(Y_t, Z_t, x_t) \in \Omega_t, \tag{2.4}$$

where $V(x_t) = U(d_t)$; $t = 1, \ldots, T$.

Several algorithms based on the decomposition principle or on particular block factorizations of the basis have been proposed to handle staircase problems. Among these approaches, the nested decomposition algorithm is certainly the one which has received the most systematic attention : it has been described in several publications [5] [6] as well as implemented and tested on various dynamic linear problems; it has also been extended by O'Neill [8] to non-linear problems.

The principle of the nested decomposition algorithm is to replace the solution of a large problem such as SC by the repeated solution of a set of T smaller problems. A set of T problems is constructed by the algorithm at each major iteration or cycle. We shall denote these cycles by an upper index κ. The problems of the set are of the following form :

<u>Problem</u> <u>SP_T^κ</u>

$$z_t^\kappa = \text{Max } V(x_T) + P_T^\kappa \lambda_T, \tag{2.5}$$

s.t. $$Q_T^\kappa \lambda_T - Y_T + Z_T = 0 \tag{2.6}$$

$$(Y_T, Z_T, x_T) \in \Omega_T \tag{2.7}$$

$$e\lambda_T = 1, \; \lambda_T > 0. \tag{2.8}$$

<u>Problems</u> $\underline{SP_t^K}$: (defined for $t = 2, \ldots, T-1$)

$$z_t^K = \max V(x_t) + P_t^K \lambda_t + \Pi_{t+1}^K \Lambda Z_t, \tag{2.9}$$

s.t. $$Q_t^K \lambda_t - Y_t + Z_t = 0 \tag{2.10}$$

$$(Y_t, Z_t, x_t) \in \Omega_t \tag{2.11}$$

$$e\lambda_t = 1, \lambda_t \geq 0. \tag{2.12}$$

<u>Problem</u> $\underline{SP_1^K}$

$$z_1^K = \max V(x_1) + \Pi_2^K \Lambda Z_1, \tag{2.13}$$

s.t. $$- Y_1 + Z_1 = Z_0 \tag{2.14}$$

$$(Y_1, Z_1, x_1) \in \Omega_1. \tag{2.15}$$

The justification of the procedure is given in [5] and [8] and will not be repeated here. For the clarity of the presentation, we shall however give an economic interpretation of some of the symbols appearing in the statement of the models and in particular of Π_t^K and Q_t^K. Strictly speaking Π_T^K and Π_t^K, $t = T-1, \ldots, 2$, are the vectors of the dual variables relative to the constraints (2.6) and (2.10) of the program SP_T^K and SP_t^K respectively : for each cycle, Π_{t+1}^K is obtained from SP_{t+1}^K and used to construct SP_t^K : this operation is performed for t varying from T-1 to 1. Intuitively a component of Π_t^K is the value for the rest of the horizon of a unitary capacity of the equipment involved in the equation corresponding to that component in the constraint (2.3). The idea of program SP_t^K

is thus to choose tentative production, consumption and investment vectors in period t, taking into account the value for the rest of the horizon of the capital stock forwarded to futrue periods. Q_t^κ and P_t^κ have a somewhat different interpretation : for every cycle κ the problem SP_t^κ determines a vector of capital stock for the E and NE sectors in period t and an evaluation of the utility that can be attained up to that period. More precisely the vector $q_{t+1}^{\kappa+1}$ is the vector $-\Lambda Z_t$ where Z_t is the optimal Z vector of problem SP_t^κ; similarly $p_{t+1}^{\kappa+1}$ is the optimal objective function value of SP_t^κ. It is worthwile to note for the sequel of the paper that, because Λ and Z are by nature nonnegative, q is a non positive vector. It is seen that each problem SP_t^κ determines its optimal capital mix by combining different capital structures inherited from the past with new investments.

We shall show in the following sections that nested decomposition, when applied to problem P^c can be combined with column generation to produce subproblems that have a number of constraints equal to the total number of goods produced and consumed in the economy plus one, thus allowing one to eliminate the technological constraints describing the equipments and the capital good conservation (1.9) to (1.11). In order to proceed toward that discussion, we first introduce some additional notions.

Let $S_{ik}(1)$ be the production set of a unit capacity of equipment (i,k); we shall assume that this set is bounded and contains the origin : it is clear that this assumption is not really restrictive in practical case.

As will be seen later, the extreme points of $S_{ik}(1)$ will play a role somewhat analogous to the columns of the input output matrix. For this reason, we shall write these extreme points, using a similar notation and define

$$\{A_{ik\ell} \mid \ell \in L_{ik}\} \tag{2.16}$$

as the set of extreme points of $S_{ik}(1)$.

We assume that we are dealing with problems for which extreme points of the production sets are easy to obtain. [10] shows that it is indeed the case for energy models where the extreme points of the $S_{ik}(1)$ can always be obtained explicitly by a one pass algorithm. The following additivity property of the production set will be useful later.

<u>*Lemma 1*</u> *: Any production set characterized by a unique capacity variable and satisfying the definition expressed in the relation (1.6) satisfies* $S(Z^1) + S(Z^2) = S(Z^1 + Z^2)$

<u>Proof</u> : the proof of the $\subset$ relation is obvious. In order to prove the $\supset$ we consider a point ξ belonging to $S(Z^1 + Z^2)$.

Defining

$$\xi^1 = \xi\frac{Z^1}{Z^1 + Z^2},$$

$$\xi^2 = \xi\frac{Z^2}{Z^1 + Z^2}.$$

It is clear that ξ^1 and ξ^2 belong respectively to $S(Z^1)$ and $S(Z^2)$ which proves the lemma.

□

Before going into the application of the nested decomposition approach to problem P^c, it may be useful to say a few words about the solution procedures that can be applied to problem SP_t^K. Various methods can be contemplated for solving SP_t^K; in this paper we shall assume that we use a reduced gradient type approach. We shall not elaborate here on the relative merits of that type of algorithm compared to other methods. The use of reduced gradient algorithm for solving SP_t^K is mainly justified in our context by the fact that it is compatible with a column generation procedure which is the procedure that we shall use later, to show how each problem SP_t^K can be transformed into a new problem with fewer constraints and a large number of columns which are only known implicitly. Since it is clear that we do not want to enumerate all those columns they will have to be generated only when required. This is performed naturally in a simplex type approach when computing the reduced cost of maximal value over the set of those unknown columns.

3. A COMBINATION OF NESTED DECOMPOSITION AND COLUMN GENERATION

According to the discussion of the preceding section, one can write the subproblem generated by the nested decomposition approach applied to P^c as follows :

Problem SP_t^K

$$z_t^K = \max U(d_t) + \Pi_{t+1}^K \Lambda Z_t + \gamma_t^K \lambda_t^K, \qquad (3.1)$$

$$\text{s.t.} \quad A^{\cdot,NE} X_t^{NE} + \sum_{ik} \xi_{tik} + B^{\cdot,NE} Y_t^{NE} + \sum_{ik} B_{ik} Y_{tik} + d_t = 0 \qquad (3.2)$$

$$\xi_{tik} \in S_{ik}(Z_{tik}),\ k \in K_i,\ i \in E \tag{3.3}$$

$$0 \leq X_t^{NE} \leq Z_t^{NE} \tag{3.4}$$

$$Q_t^{\kappa}\,\lambda_t - Y_t + Z_t = 0 \tag{3.5}$$

$$e\lambda_t = 1,\ \lambda_t \geq 0. \tag{3.6}$$

We shall now indicate how a column generation procedure can be applied to this problem in order to transform SP_t^{κ} into a new problem that only contains a number of constraints equal to the number of goods + 1.

In order to simplify the notation, we shall drop in the rest of the presentation all indices κ and t. In particular, the multipliers Π_{t+1}^{κ} and Π_t^{κ} will be denoted as $\bar{\Pi}$ and Π respectively. Let us assume that Q contains exactly K columns and let Q^p denote one of these : the subvector of Q^p corresponding to the NE goods will be noted as $Q^{p,NE}$, while the component of Q^p corresponding to an equipment k of the energy sector i will be noted Q_{ik}^p. The proposed procedure is based on the fact that the ξ_{tik} belonging to $S_{ik}(Z_{ik})$ can be represented very easily as convex combination of vectors that are quite easy to obtain. Briefly speaking the sets $S_{ik}(Z_{ik})$ can be replaced by sums of the type

$$S_{ik}\left(- \sum_{p=1}^{K} Q_{ik}^p\,\lambda_p\right) + S_{ik}(Y_{ik}), \tag{3.7}$$

where the extreme points of each set in the sum are easily to obtain. In order to discuss this systematically, we define the production set associated with the capital stock Q^p, let

$$D^P \equiv \{\eta^P \mid \eta^P = A^{\cdot,NE} X^{NE} + \sum_{ik} \xi_{ik};\ 0 \leqslant X^{NE} \leqslant -Q^{P,NE};$$

$$\xi_{ik} \in S_{ik}(-Q^P_{ik});\ k \in K_i,\ i \in E\}. \tag{3.8}$$

Program SP^K_t considers convex combinations of different capital stock vectors $- Q^P$: we first show that the same convex combinations allows one to mix the production sets D^P. This is stated in the following lemma.

Lemma 2 : One has for all vectors $\lambda \geqslant 0$

$$\sum_{p=1}^{K} D^P\lambda_p \equiv \left\{\eta \mid \eta = A^{\cdot,NE} X^{NE} + \sum_{ik} \xi_{ik};\ \xi_{ik} \in S_{ik}\Big((-Q\lambda)_{ik}\Big);\right.$$

$$\left. 0 \leqslant X^{NE} \leqslant \Big((-Q\lambda)^{NE}\Big)\right\}. \tag{3.9}$$

Proof : Let η be an element of $\sum_{p=1}^{K} D^P\lambda_p$. One can write

$$\eta = \sum_{p=1}^{K} \eta^P \lambda_p \text{ with } \eta_p \in D^P.$$

By definition of η^P, there exists $X^{P,NE}$ and ξ^P_{ik} such that

$$\eta^P = A^{\cdot,NE} X^{P,NE} + \sum_{ik} \xi^P_{ik},$$

with $0 \leqslant X^{P,NE} \leqslant - Q^{P,NE}$ and $\xi^P_{ik} \in S_{ik}(-Q^P_{ik})$, $k \in K_i$, $i \in E$.

Combining these relations, one can write

$$\eta = A^{\cdot,NE} \sum_{p=1}^{K} X^{P,NE} \lambda_p + \sum_{ik} \sum_{p=1}^{K} \xi^P_{ik} \lambda_p,$$

with $0 \leqslant \sum_{p=1}^{K} X^{P,NE} \lambda_p \leqslant -\sum_{p=1}^{K} Q^{P,NE} \lambda_p$

and $\sum_{p=1}^{K} \xi^P_{ik} \lambda_p \in \sum_{p=1}^{K} S_{ik}(-Q^P_{ik} \lambda_p) = S_{ik}\left(-\sum_{p=1}^{K} Q^P_{ik} \lambda_p\right)$

To prove the converse, we consider η equal to

$$A^{\cdot,NE} X^{NE} + \sum_{ik} \xi_{ik},$$

for some vector X^{NE} and ξ such that

$$0 < X^{NE} < (-Q\lambda)^{NE},$$

$$\xi_{ik} \in S_{ik}\left((-Q\lambda)_{ik}\right).$$

Because of lemma 1, one can find $X^{P,NE}$ and ξ^{P}_{ik} such that

$$0 < X^{P,NE} < -Q^{P,NE},$$

$$\xi^{P}_{ik} \in S_{ik}(-Q^{P}_{ik}),$$

$$X^{NE} = \sum_{p=1}^{K} X^{P,NE} \lambda_{p},$$

$$\xi_{ik} = \sum_{p=1}^{K} \xi^{P}_{ik} \lambda_{p}.$$

The proof follows then trivially.

□

In order to proceed toward a new problem equivalent to SP we first introduce a few additional notation. We first consider the capital vector Q^P and its associated production sets D^P. The operations of an economy of capital structure Q^P can be completely described by the extreme points of D^P. Since these points will roughly play the same role as columns of the A matrix we shall denote then using a similar notation $A_{D^P \ell}$ where ℓ is a current index taking its values in a set L_{D^P}. We also denote $X_{D^P \ell}$ to be the activity level of $A_{D^P \ell}$ in the economy. Consider now the vector Y^{NE} of new capacities in the non energy sectors. We define V^{NE} to be the vector of these new capacities which remains idle during the current period and $X^{+,NE}$ the amount of that new capacity already operated during the period. To $X^{+,NE}$ is thus associated a contribution

$$(A^{\cdot,NE} + B^{\cdot,NE})\, X^{+,NE},$$

to the bill of goods, while V^{NE} only contributes for $B^{\cdot,NE}\, V^{NE}$.

Finally, we crider the new capacities of the energy section. As already introduced before $A_{ik\ell}$ designates an extreme point of $S_{ik}(1)$. We let $X_{ik\ell}$ denote the activity level associated with $A_{ik\ell}$ and V_{ik} the newly invested capacity that remains idle during the period. The contribution to the bill of goods due to the new capacity is then

$$\sum_{ik} \left[\sum_{\ell \in L_{ik}} (A_{ik\ell} + B_{ik}) X_{ik\ell} + B_{ik}\, V_{ik} \right]$$

and the newly invested capacity

$$\sum_{\ell \in L_{ik}} X_{ik\ell} + V_{ik}$$

Using this notation, we can then introduce new equivalent problem SP^r as follows.

Let $A_{D^p\ell}$, $\ell \in L_{D^p}$ the extreme points of D^p, one then introduces the problem SP^r which shall be proved to be a substitute for SP in the decomposition approach.

<u>Program SP^r</u>

$$\text{Max } U(d) + (\bar{\Pi}\Lambda)^{NE} (X^{+,NE} + V^{NE}) + \sum_{ik} (\bar{\Pi}\Lambda)_{ik} (X_{ik\ell} + V_{ik})$$

$$+ \sum_{p=1} \sum_{\ell \in L_{D^p}} (P-\bar{\Pi}Q)_p \, X_{D^p\ell}, \qquad (3.10)$$

$$\text{s.t.} \quad (A^{\cdot,NE} + B^{\cdot,NE})X^{+,NE} + B^{\cdot,NE} V^{NE} + \sum_{ik} \sum_{\ell \in L_{ik}} (A_{ik\ell} + B_{ik})X_{ik\ell} + \sum_{ik} B_{ik} V_{ik} + \sum_{p=1} \sum_{\ell \in L_{D^p}} A_{D^p\ell} X_{D^p\ell} + d = 0 \quad (3.11)$$

$$\sum_{p=1}^{K} \sum_{\ell \in L_{D^p}} X_{D^p\ell} = 1, \quad (3.12)$$

$$X_{D^p\ell} \geq 0, \ \ell \in L_{D^p}, \ p = 1, \ldots, K. \quad (3.13)$$

It is clear that the number of constraints of SP^r is equal to the number of goods plus one. In the following, we shall denote by ρ and σ respectively the multipliers associated with the constraints (3.11) and (3.12) respectively.

Before stating any equivalence property between SP and SP^r, it is necessary to indicate exactly what it meant by that notion in the context of the nested decomposition approach.

It has been recalled at the beginning of this section that the implementation of the nested decomposition algorithm requires at every cycle and for each problem SP_t^κ the vector Π_{t+1}^κ of multipliers associated to the constraints (3.5) of SP_{t+1}^κ. In order to define $SP_t^{\kappa+1}$ it is also necessary to know the capital stock vector at cycle κ generated by the problem SP_t^κ. More precisely it can be shown on the basis of [8] that the following elements and conditions are required :

A. The optimal z_t^κ, d_t^κ and λ_t^κ of each problem SP_t^κ. These elements are used to construct p_t^κ;

B. The optimal z_t^κ and λ_t^κ of each problem SP_t^κ. This vector allows one to construct q_t^κ;

C. The multipliers Π^{κ}_{t} and σ^{κ}_{t} associated with the constraints (3.5) and (3.6). These multipliers satisfy the following optimality conditions :

C.1. $\sigma^{\kappa}_{t} - \Pi^{\kappa}_{t}(Z_t - Y_t) - P^{\kappa}_{t}\lambda^{\kappa}_{t} = 0$ (see relation (12) in [7]).

C.2. $P^{\kappa}_{t+1} + \Pi^{q}_{t+1}\Lambda Z^{\kappa}_{t} < \sigma^{q}_{t+1}$, $t = 1, \ldots, T-1$ et $q > \kappa + 1$ (see relation (13) in [7]).

Both C.1. and C.2. are used to prove convergence of the method [8]. We shall not discuss here the convergence proof but simply say a few words about these relations. Condition C.1. states the optimality conditions for the variables in problem SP^{κ}_{t}. Similarly condition C.2. states that the reduced cost of a variable κ must be non positive at the optimum for all subsequent subproblems SP^{q}_{t+1}. This clearly implies that all generated proposals q_t are kept in program SP_t once they have been generated.

4. RECONSTITUTION OF THE OPTIMAL VARIABLES OF THE ORIGINAL PROBLEM

The following proposition indicates how the optimal variables of the original problem SP can be recovered from the optimal primal variables of SP^r.

Proposition 1 : The following relations between SP *and* SP^r *hold at the optimum*

$$\sum_{\ell \in L_{D^p}} X_{D^p \ell} = \lambda_p, \quad p = 1, \ldots, K, \tag{4.1}$$

$$\sum_{\ell \in L_{ik}} X_{ik\ell} + V_{ik} = Y_{ik}, \quad i \in E,\ k \in K_i \tag{4.2}$$

$$X^{+,NE} + V^{NE} = Y^{NE} \qquad (4.3)$$

$$Z = Y - Q\lambda \qquad (4.4)$$

<u>Proof</u> : The proof is obtained by a succession of transformations of the problem SP. These transformations are based on the additivity of the production sets on lemma 2. We first write SP as

$$\text{Max } U(d) + \overline{\Pi}\Lambda - (\overline{\Pi}\Lambda Q)\ \lambda + P\lambda,$$

$$\text{s.t.} \quad A^{\cdot,NE}\, X^{+,NE} + B^{\cdot,NE}\, Y^{NE} + \sum_{ik} \xi_{ik} + \sum_{ik} B_{ik}\, Y_{ik} + \sum_{p=1}^{K} \xi^{p}\, \lambda_{p} + d = 0,$$

$$\xi_{ik} \in S_{ik}(Y_{ik}),\ i \in E,\ k \in K_{i}\ ;\ 0 \leq X^{+,NE} \leq Y^{NE},$$

$$\xi^{p} \in D^{p},\ p = 1, \ldots, K$$

$$e\lambda = 1,\ \lambda \geq 0.$$

Introducing the extreme points of S_{ik} and D^{p} one gets successively

$$\xi_{ik} = \sum_{\ell \in L_{ik}} A_{ik\ell}\, X_{ik\ell} \text{ with } \sum_{\ell \in L_{ik}} X_{ik\ell} + V_{ik} = Y_{ik}$$

$$\text{and} \quad \xi^{p} = \sum_{\ell \in L_{D^{p}}} A_{D^{p}\ell}\, X_{D^{p}\ell} \text{ with } \sum_{\ell \in L^{p}} X_{D^{p}\ell} = 1 \text{ and } X_{D^{p}\ell} \geq 0,$$

where V_{ik} represents the unused capacity invested in ik in the current period. This variable does not have to be introduced if {o} is an extreme point of $S_{ik}(1)$: indeed in that case the role of V_{ik} can be taken over by the $X_{ik\ell}$ corresponding to that extreme point. After introducing a variable V^{NE} to represent the unused capacity invested in the NE sectors during the current period and substituting these expressions in the program just obtained one arrives at SP^{r}. The expressions for λ_{p}, Y_{ik}, Y^{NE} and Z can then be derived from these transformations.

□

The derivation of the optimal dual variables associated with the capacity constraints (3.5) is not as clear. The following intuitive reasoning leads to expressions for these variables that will be justified in the next section.

It is known that dual variables at the optimum represent the derivative of the optimal objective function with respect to the right-hand side of the constraints. In order to evaluate the derivative we consider a small increase ε_{ik} of some capital stock ik. The corresponding capital stock balance equation will read

$$Z_{ik} - Y_{ik} + (Q\lambda)_{ik} = \varepsilon_{ik} \tag{4.5}$$

The resulting increase of the objective function value is twofold. A first contribution to this increase is the value of the additional capital stock that will be forwarded to future periods : this contribution is equal to $(\overline{\Pi}\Lambda)_{ik}\ \varepsilon_{ik}$. A second contribution arises from the enlarged possibilities of the economy due to the additional capital stock. In order to evaluate this second element, we consider the extreme point $A_{ik\ell(\rho)}$ of $S_{ik}(1)$ satisfying

$$- \rho A_{ik\ell(\rho)} = \max\{- \rho\xi \mid \xi \in S_{ik}(1)\}.$$

It is clear that the improvement of the objective function due to the capital stock increase ξ_{ik} will be

$$- \rho A_{ik\ell(\rho)}\ \varepsilon_{ik},$$

if this term is positive and zero otherwise.

Intuitively one can then propose for the values of dual variables Π

$$\Pi_{ik} = (\bar{\Pi}\Lambda)_{ik} + \max\,[-\,\rho\, A_{ik\ell(\rho)}\ ;\ 0] \qquad (4.6)$$

$$\Pi^{NE} = (\bar{\Pi}\Lambda)^{NE} + \max\,[-\,\rho\, A^{NE}\ ;\ 0] \qquad (4.7)$$

The following section shows that these expressions fulfill the conditions C_1 and C_2 defined before.

5. *THE OPTIMAL DUAL VARIABLES Π*

Consider an extreme point $A_{D^P\ell}$ of D^P. Because of the definition of D^P one can write (see section 6 lemma 3)

$$A_{D^P\ell} = -\sum_{i\in NE} A^{\cdot,i}\,\delta(i,\ell,D^P)\,Q_i^P - \sum_{ik} A_{ik}(A_{D^P\ell})\,Q_{ik}^P \qquad (5.1)$$

where - $\delta(i,\ell,\,D^P)$ is equal to zero if the NE vector i, $A^{\cdot,i}$ appears with activity level zero in the extreme point $A_{D^P\ell}$;

- $A_{ik}(A_{D^P\ell})$ designates the extreme point of S_{ik} contributing to the extreme point $A_{D^P\ell}$ of D^P.

In the following we shall deal with points of D^P that satisfy the same type of expression (5.1) without being extreme point of D^P. Since there are only a finite number of these points we shall denote them as A_{D^Pn} , n being the current index identifying each of these points; we can write, using notation similar to the one appearing in (5.1)

$$A_{D^Pn} = -\sum_{i\in NE} A^{\cdot,i}\,\delta(i,n,D^P)\,Q_i^P - \sum_{ik} A_{ik}(A_{D^Pn})\,Q_{ik}^P\,. \qquad (5.2)$$

Suppose now that the problem SP^r is solved using a revised simplex method and let ρ and σ be the dual variables associated with the constraints (3.11) and (3.12) respectively at some iteration. The reduced cost of the variable $X_{D^P\ell}$ is then

$$(P - \overline{\Pi}\Lambda Q)_p - \rho A_{D^P\ell} - \sigma, \tag{5.3}$$

that we shall designate by $RC(A_{D^P\ell})$ in the following. Consider now a point A_{D^Pn} as defined before. One shall define for each A_{D^Pn} an expression $RC(A_{D^Pn})$ of the same algebraic form as (5.3). Since A_{D^Pn} is not a column of SP^r, $RC(A_{D^Pn})$ is no longer a reduced cost. It is introduced here for future use in the proofs.

We can now state the following propositions

Proposition 2 : *For each* p, *there is at most one vector* $X_{D^P\ell}$ *in the basis*.

Proof : Suppose not and let ℓ' and ℓ'' be two extreme points of D^P belonging to the basis; one considers the vector A_{D^Pn} obtained as follows.

For $i \in NE$ let

$$-\rho A^{\cdot,i}\ \delta(i,n,D^P) = \max\Big((-\rho A^{\cdot,i}\ \delta(i,\ell',D^P)\ ;$$

$$-\rho A^{\cdot,i}\ \delta(i,\ell'',D^P)\Big).$$

For every ik, $i \in E$, $k \in K_i$, let

$$-\rho A_{ik}(A_{D^Pn}) = \max\Big(-\rho A_{ik}(A_{D^P\ell'})\ ;\ -\rho A_{ik}(A_{D^P\ell''})\Big).$$

It is clear that the vector A_{D^Pn} belongs to D^P. Moreover since $A_{D^P\ell'}$ and $A_{D^P\ell''}$ are in the basis, one has

$$RC(A_{D^P\ell'}) = RC(A_{D^P\ell''}) = 0,$$

and hence neglecting the case of degeneracy and taking into account the nonpositivity of Q^P

$$RC(A_{D^Pn}) > 0.$$

Writing $A_{D^p_n}$ as a convex combination of extreme points of D^p, one obtains

$$A_{D^p_n} = \sum_{\ell \in L} \nu_\ell \, A_{D^p_\ell}$$

with

$$\sum_{\ell \in L} \nu_\ell = 1 \text{ and } \nu_\ell \geq 0 \text{ for } \ell \in L$$

which implies

$$RC(A_{D^p_n}) = \sum_{\ell \in L} \nu_\ell \, RC(A_{D^p_\ell})$$

and hence, there exit extreme points of D^p with a positive reduced cost. This contradicts the optimality of SP^r.

□

Proposition 3 : If $X_{D^p_\ell} > 0$ *in the optimal solution, then one has*

$$- \rho \, A^{\cdot,i} \, \delta(i,\ell,D^p) \geq 0 \qquad i \in NE$$

$$- \rho \, A_{ik}(A_{D^p_\ell}) \geq 0 \qquad i \in E,\ k \in K_i$$

Proof : We first show that ρ is nonnegative. To see this we assume that the equality in relation (3.11) is replaced by an inequality $\leq$; because each function U is nondecreasing in d, these inequalities will be tight at the optimum and hence the problem with the inequality sign is equivalent to the original program SP^r. ρ is then nonnegative because it is the dual variable vector of a set of inequality constraints.

Suppose now that the proposition is not true and let $\bar{i}$, $\bar{k}$ be such that

$$- \rho \, A_{\bar{i}\bar{k}}(A_{D^p_\ell}) < 0$$

We let $A_{D^p_n}$ be the point of D^p obtained by replacing $A_{\bar{i}\bar{k}}(A_{D^p_\ell})$ by the zero vector in the expression of $A_{D^p_n}$. It is clear that $A_{D^p_n}$ belongs to D^p. Moreover $RC(A_{D^p_n})$ is positive. A contradiction can then be obtained as in the preceding proposition.

Before stating the next proposition we recall that $A_{ik\ell(\rho)}$ designates the extreme point of S_{ik} that maximises $-\rho\xi$ on $S_{ik}(1)$.

□

Proposition 4 : If $X_{D^p_\ell} > 0$ *in the optimal solution, then one has*

$$- \rho A^{\cdot,i}\, \delta(i,\ell,D^p) = \max\left(- \rho A^{\cdot,i}\ ;\ 0\right) \quad \text{for } i \in NE;$$

$$A_{ik}(A_{D^p_\ell}) = A_{ik\ell(\rho)} \quad \text{for } k \in K_i \text{ and } i \in E.$$

Proof : The first relation is obtained directly from the preceding lemma. In order to prove the second relation, we suppose that

$$A_{ik}(A_{D^p_\ell}) \neq A_{ik\ell(\rho)}$$

and define $A_{D^p_n}$ as the vector obtained by replacing $A_{ik}(A_{D^p_\ell})$ by $A_{ik\ell(\rho)}$ in the expression $A_{D^p_\ell}$. Because of the definition of $\ell(\rho)$ one has

$$-\rho A_{ik\ell(\rho)} > -\rho A_{ik}(A_{D^p_\ell})$$

and hence

$$RC(A_{D^p_n}) > 0,$$

which leads again to the same type of contradiction as precedingly..

□

As a corollary of these propositions it is possible to prove that if λ_p defined by relation (4.1) is positive, then one has

$$\left(P - \Pi Q\right)_p - \sigma = 0,$$

which is part of the property C1 mentioned at the end of section 3. This is shown in the following proposition.

Proposition 5 : Let λ_p *be such that*

$$\lambda_p = \sum_{\ell \in L_{D^p}} X_{D^p\ell} > 0$$

and Π *be the vector defined in relations (4.6) and (4.7). Then*

$$P_p - \Pi Q^p - \sigma = 0.$$

Proof : Consider the basic variable $X_{D^p\ell}$. One has

$$RC(A_{D^p\ell}) = 0$$

and hence

$$P_p - (\overline{\Pi}\Lambda Q)_p - \rho\, A_{D^p\ell} - \sigma = 0.$$

Because of the preceding proposition one has

$$\rho\, A_{D^p\ell} = \sum_{i \in NE} \max\left(-\rho\, A^{\cdot,\cdot,i}\ ;\ 0\right) Q_i^p + \sum_{\substack{i \in E\\ k \in K_i}} \max\left(-\rho\, A_{ik\ell(\rho)}\ ;\ 0\right) Q_{ik}^p$$

and hence $RC(A_{D^p\ell})$ can be written as

$$P_p - \left\{\sum_{i\ NE} \left((\overline{\Pi}\Lambda)_i + \max(-\rho\, A^{\cdot,\cdot,i}\ ;\ 0)\right)\right\} Q_i^p$$

$$-\sum_{\substack{i \in E\\ k \in K_i}} \left((\overline{\Pi}\Lambda)_{ik} + \max(-\rho\, A_{ik\ell(\rho)}\ ;\ 0)\right) Q_{ik}^p - \sigma = P_p - \Pi Q^p - \sigma = 0.$$

□

We now consider the optimality conditions for the λ_p that are equal to zero namely those for which $\sum_{\ell \in L_{D^p}} X_{D^p\ell} = 0.$

One can write this following proposition

Proposition 6 : If $X_{D^P\ell} = 0$ *for all* $\ell \in L_{D^P}$

$$(P - \overline{\Pi}\Lambda Q)_p - \sum_{i\in NE} \max(-\rho\cdot A^{\cdot\cdot,i};\ 0)Q_i^p$$

$$- \sum_{ik} \max(-\rho A_{ik\ell(\rho)}\ ;\ 0)Q_{ik}^p - \sigma \leq 0$$

Proof : Suppose not and let A_{D^Pn} be the point defined as in (5.2) where

$\delta(i,n,D^P) = 0$ if $\max\ (-\rho A^{\cdot\cdot,i}\ ;\ 0) = 0;$

$\delta(i,n,D^P) = 1$ if $\max\ (-\rho A^{\cdot\cdot,i}\ ;\ 0) > 0;$

$A_{ik}(A_{D^Pn}) = 0$ if $\max\ (-\rho A_{ik\ell(\rho)}\ ;\ 0) = 0;$

$A_{ik}(A_{D^Pn}) = A_{ik\ell(\rho)}$ if $\max\ (-\rho A_{ik\ell(\rho)}\ ;\ 0) > 0.$

One has, because we have assumed the proposition to be false that

$$RC(A_{D^Pn}) - RC(A_{D^P\ell}) > 0$$

which leads to a contradiction as in proposition 2.

□

Corollary : If $\lambda_p = 0$*, then* $P_p - \Pi Q^P - \sigma \leq 0$

Proof : This follows directly from the preceding proposition.

We can now show the validity of the expressions (4.6) and (4.7) defined in the preceding section.

Proposition 7 : (4.6) and (4.7) satisfy the condition C stated in section 4.

Proof : Because of the preceding proposition we have that $\lambda_p > 0$ implies $P_p - \Pi Q^P - \sigma = 0$ and $\lambda_p = 0$ implies $P^P - \Pi Q^P - \sigma \leq 0,$

and hence

$$P\lambda - \Pi Q\lambda - \sigma = 0.$$

Since

$$Z - Y + Q\lambda = 0,$$

one can write

$$P\lambda + \Pi(Z-Y) - \sigma = 0,$$

which is property C1.

In order to prove C2 note that one has at the optimum of SP_{t+1}

$$P^{\kappa}_{t+1} - \Pi^{q}_{t+1}\, Q^{\kappa}_{t+1} - \sigma^{q}_{t+1} < 0 \text{ for } q > \kappa + 1.$$

C2 follows then trivially from the fact that

$$- Q^{\kappa}_{t+1}.$$

□

6. GENERATION OF EXTREME POINTS

In this section we briefly discuss the modeling approach underlying program P^{c} and show how it naturally allows one to find the extreme points required by the algorithmic framework presented in this paper. The approach was introduced in the context of energy modeling but can easily be extended to other field.

In energy flow models ([3] [7]) the different energy production and consumer sectors are represented as a graph. Following this description, we shall assume each process of an E sector to be represented as in figure 2.

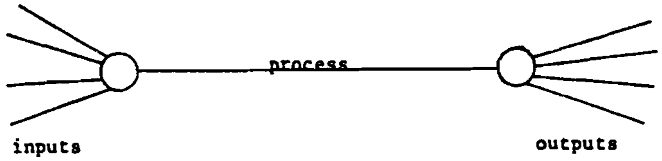


Fig. 2 : Energy flow representation of a process

In this representation, an arc is associated to each process. Various inputs are consumed by the process which also produces some outputs; bounds are imposed on the inputs and outputs indicating that they cannot be consumed or produced in any proportion.

Let ξ^I and ξ^0 be respectively the vectors of goods consumed and produced by the process. Because of technological constraints the inputs and outputs must usually remain within certain maximal and minimal proportions. The set of constraints describing the process is then as follows :

- a first constraint expresses a conservation principle (material or energy)

$$a^0 \xi^0 - a^I \xi^I = 0 \qquad (6.1)$$

- a second set of constraints expresses maximal and minimal proportions on the input and output of the system : they can be stated as

$$\underline{b}_i^0 a^0 \xi_i^0 < \xi_i^0 < \bar{b}_i^0 a^0 \xi^0 \text{ for all output i,} \qquad (6.2)$$

$$\underline{b}_i^I a^I \xi^I < \xi_i^I < \bar{b}_i^I a^I \xi^I \text{ for all input i;} \qquad (6.3)$$

- the last constraint expresses the capacity limitation of the equipment. If we assume a unit capacity we can write this constraint as

$$0 < a^0 \xi^0 < 1 \qquad (6.4)$$

Let us assume first that the equipment operates at a constant level throughout a period of the planning horizon and let ρ_ξ be the subvector of ρ consisting of the components of ρ related to the goods appearing in the operation of the equipment. Clearly the problem of evaluating a maximal reduced cost for a column associated to an equipment of this type can be formulated as

$$\text{Min } \rho_{\xi^I}\xi^I + \rho_{\xi^0}\xi^0, \qquad (6.5)$$

$$\text{s.t.} \quad a^0 \xi^0 - a^I \xi^I = 0, \qquad (6.6)$$

$$\underline{b}_i^0 \, a^0 \xi^0 < \xi_i^0 < \bar{b}_i^0 \, a^0 \xi^0 \text{ for all output } i, \qquad (6.7)$$

$$b_i^I \, a^I \xi^I < \xi_i^I < \bar{b}_i^I \, a^I \xi^I \text{ for all input } i, \qquad (6.8)$$

$$0 < a^0 \xi^0 < 1 \qquad (6.9)$$

We first note that an obvious extreme point of this production set is not to operate the plant at all; this corresponds to a zero objectif function value. If the plant is to be operated then it will be at full capacity, which implies

$$a^0 \xi^0 = a^I \xi^I = 1, \qquad (6.10)$$

and hence the preceding problem is reduced to the following set of two problems

$$\underset{\xi^I}{\text{Min}}\ \rho_I \xi \tag{6.11}$$

$$A^I\ \xi^I = 1 \tag{6.12}$$

$$\underline{b}_i^I < \xi_i^I < \overline{b}_i^I \text{ for all output i,} \tag{6.13}$$

and

$$\underset{\xi^0}{\text{Min}}\ \rho_0 \xi^0 \tag{6.14}$$

$$a^0\ \xi^0 = 1 \tag{6.15}$$

$$\underline{b}_i^0 < \xi_i^0 < \overline{b}_i^0 \text{ for all input i,} \tag{6.16}$$

for which an explicit solution can be found by some simple logic (knapsack problem in continuous variables).

Much more complex situations can be considered which include phenomena such as time varying operations and storage, while still allowing one to generate extreme points explicitly. A systematic discussion of the approach with examples taken from the energy sector is presented in [10].

As a final remark we indicate how extreme points of D^P can be obtained easily from the extreme points of the S_{ik}.

Lemma 3 : *An extreme point of* D^P *is a sum of extreme points of the* $S_{ik}\left(-Q_{ik}^P\right)$ *for* $k \in K_i$ *and* $i \in E$ *and of vectors* 0 *or* $-A^{\cdot,i}\ Q_i^{NE}$ *for* $i \in E$.

Proof : Since D^P is polyhedral, there exists for every extreme point ξ^* of D^P a vector ρ such that

$$\rho\xi^* = \max\{\rho\ \xi \mid \xi \in D^P\}.$$

The lemma follows directly from the definition of D^P.

□

CONCLUSIONS

Because of their size dynamic input output models may be difficult to extend so as to include a detailed representation of some of the sectors of the economy. In this paper, we propose a general formulation of those models that considers a detailed representation of some sectors. This representation is based on the assumption that the equipments of the sectors of interest are characterized by a single capacity variable and that their production set is simple enough so as to allow one to construct their extreme points easily. These assumptions are taken from the field of energy flow modeling where they are generally satisfied. A special purpose algorithm is proposed for the resulting model which takes advantages of the aforementioned representation of some of the equipments. The algorithm is a combination of nested decomposition and column generation. Nested decomposition is applied first on the dynamic model to transform it into a set of smaller subproblems. A further reduction of the number of constraints of each of the resulting subproblems is then obtained by eliminating the constraints describing the operation of the equipments satisfying the assumption.

REFERENCES

[1] Dantzig G.B., Connoly T.J. and S.C. Parikh, "Stanford PILOT Energy/Economy Model", Electric Power Research Institute Report E.A.-626 Palo Alto, Cal., 1978.

[2] Dantzig G.B., "On the Reduction of an Integrated Energy and Interindustry Model to a Smaller Linear Program" Technical Report, SOL 74-20, Department of Operations Research, Stanford, Cal, 1974.

[3] Finon D., "Un modèle énergétique pour la France", Paris, CNRS, 1976.

[4] Goreux, L.M. AND A.S. Manne , editors, *Multi-Level Planning : Case Studies in Mexico,* North Holland, 1973.

[5] Ho J.K. and A.S. Manne, "Nested Decomposition of Dynamic Models", *Mathematical Programming 6 (2)* 1974, 121-140.

[6] Ho J.K., "Nested Decomposition of Dynamic Energy Models", *Management Sciences, 23 (9),* 1977, 1023-1026.

[7] Hoffman K.C. and D.W. Jorgenson, "Economic and Technological Models for Energy Policy", *The Bell Journal of Economics 8(2),* 1977, 444-466.

[8] O'Neill R.P., "Nested Decomposition of Multistage Convex Programs", *SIAM Journal on Control and Optimisation, 14, (2),* 1976.

[9] Propoi A. and I. Zimin, "Energy Ressources : Economy Developments Models", Working Paper WP-79-2, IIASA Laxemburg, Austria.

[10] Smeers Y., "Construction et Décomposition de Modèles Globaux Energie/Economie", Mimeo, C.O.R.E., Université Catholique de Louvain, 1979.

[11] Thompson R.G., Calloway, J.A. and L.A. Nawalalic, editor, "The Cost of Energy and a Clean Environment", Gulf Publishing Company, Book division Houston Texas, 1978.

BIBLIOGRAPHY ON LARGE-SCALE SYSTEMS

PART I. GENERAL 1949–1966

PART II. CLASSIFIED 1949–1980

GENERAL BIBLIOGRAPHY ON LARGE-SCALE SYSTEMS 1949—1966

1. ABADIE, J.M., "Dual Decomposition Method for Linear Programs", Comp. Center Case Institute of Technology, July 1962.

2. ABADIE, J.M., "On Decomposition Principle", Operations Research Center, University of California, Berkeley, ORC 63-20, 1963.

3. ABADIE, J.M. and WILLIAMS, A.C., "Dual and Parametric Methods in Decomposition", in Recent Advances in Math. Prog., edited by R. Graves and P. Wolfe, McGraw-Hill, 1963.

4. ACZEL, M.A. and RUSSEL, A.H.,"New Methods of Solving Linear Programs", O.R. Qu. Vol. 8 No. 4, Dec. 1957.

5. ADIN, B. Thomas, "Optimizing a Multistage Production Process", O.R. Qu. Vol. 14, No. 2, June 1963.

6. AGGARWAL, S.P., "A Simplex Technique for a Particular Convex Programming Problem", Canadian Operational Research Journal, Vol. 4, No. 2 July 1966.

7. ALTMAN, M., "An Elimination Method for L.P. with Application to the Decomposition Problem", Bull. Acad. Polon. Sci. Ser. Sci. Math. Astron. Physics.

8. ALWAY, G.G., "A Triangularization Method for Computations in Linear Programming", Naval Research Logistics Quarterly, Vol. 9, pp. 163-180.

9. BAKES, M.D., "Solution of Special Linear Programming Problem with Additional Constraints", O.R. Qu. Vol. 17, No. 4, Dec. 1966.

10. BALAS, Egon, "An Infeasibility - Pricing Decomposition Method for Linear Program", July 1966, Operations Research 14 (1966) 843-873.

11. BALAS, Egon, :Solution of Large Scale Transportation Problems Through Aggregation", Operations Research, 13 (1965) 82-93.

12. BALINSKI, M.L., "On Some Decomposition Approaches in Linear Programming", and "Integer Programming", The University of Michigan Engineering Summer Conferences, 1966.

13. BARNETT, S., "Stability of the Solution to a Linear Programming Problem", O.R. Qu., Vol. 13, No. 3, September 1962.

14. BAUMOL, W.J. and FABIAN, T., "Decomposition, Pricing for Decentralization and External Economics", Management Science, Vol. 11 No. 1, September 1964.

15. BEALE, E.M.L., "Survey of Integer Programming", O.R. Qu. Vol. 16, No. 2, June 1965.

16. BEALE, E.M.L., "Decomposition and Partitioning Methods for Nonlinear Programming", in Non-Linear Programming, J. Abadie, Ed., North-holland publishing Company, also Wiley.

17. BEALE, E.M.L., "The Simplex Method Using Pseudo-Basic Variables for Structured Linear Programming Problems", from Recent Advances in Math. Prog., edited by R. Graves and P. Wolfe, McGraw-Hill, 1963.

18. BELL, E.J., "Primal-Dual Decomposition Programming", Unpublished Ph.D. Thesis, Industrial Engineering Department, University of California, Berkeley, 1964.

19. BELLAR, F.J., "Iterative Solution of Large-Scale Systems of Simultaneous Linear Equations", SIAM Journal, Vol. 9, No. 2, June 1961.

20. BENDERS, J.F., "Partitioning Procedures for Solving Mixed Variables Programming Problems", Num. Math. 4, 1962.

21. BENNETT, J.M., "An Approach to Some Structured Linear Programming Problems" Operations Research 14 (1966) 4 (July-August) pp. 636-645.

22. BESSIERE, F. et SAUTER, E., "Optimisation et Enviornment Economique: La Methode Des Modeles Flargis", Revue Francaise de Recherche Operationnelle No. 40, 1966.

23. BOOT, J.C.G., "On Trivial and Binding Constraints in Programming Problems", Management Sci. Vol. 8, 1962, pp. 419-441.

24. BRADLEY, S.P., "Solution Techniques for the Traffic Assignment Problem", ORC 65-35, University of California, Berkeley, 1965.

25. BRASILOW, C.B., LASDON, L.S., PEARENS, J.D., MACKO, O., TAKAHORA, Y., "Papers on Multilevel Control Systems", DTV 70-A-65, Case Institute of Technology, 1965.

26. CATCHPOLE, A.R., "The Application of Linear Programming to Integrated Supply Problems in the Oil Industry", O.R. Qu., Vol. 13, No. 2, June, 1962.

27. CHARNES, A. and COOPER, W.W., "Generalizations of the Warehousing Model", O.R. Qu. Vol. 6, No. 4, Dec. 1955.

28. CHARNES, A. and COOPER, W.W., "Management Models and Industrial Applications in Linear Programming", Management Science, Vol. 4, No. 1 October 1957, pp. 38-91.

29. CHURCHMAN, C.W., "On the Ethics of Large-Scale Systems, Part I", Internal Working Paper, No. 37, SSL, University of California, Berkeley, September 1965.

30. CRAVEN, B.D., "A Generalization of the Transportation Method of Linear Programming", O.R. Qu. Vol. 14, No. 2, June 1963.

31. CURTES, H.A., "Use of Decomposition Theory in the Solution of the State Assignment Problem of Sequential Machines", Journal of A.C.M., July 1963, p. 386.

32. DANTZIG, G.B., "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming", Econometrica, Vol. 23, No. 2 April, 1955.

33. DANTZIG, G.B., "Optimal Solution of a Dynamic Leontief Model with Substitution", Econometrica, Vol. 23, No. 3, July 1955.

34. DANTZIG, G.B., "Linear Programming Under Unvertainty", Management Science, Vol. 1 (1955) pp. 197-206.

35. DANTZIG, G.B., "On the Status of Multistage Linear Program", The RAND Corp. p. 1028, 20 Feb. 1957, Proc. International Statistical Institute, Stockholm, 1957.

36. DANTZIG, G.B., "On the Status of Multistage Linear Programming Problems", Management Science, Vol. 6, No. 1, October 1959, Also in, Mathematical Studies in Management Science, - Veinott.

37. DANTZIG, G.B., "Compact Basis Triangularization for the Simplex Method", from Recent Advances in Math. Prog. edited by R. Graves and P. Wolfe, McGraw-Hill, 1963.

38. DANTZIG, G.B., "Linear Programming and Extensions" Princeton University Press, 1963, 1966.

39. DANTZIG, G.B., "Large Scale System Optimization", ORC 65-9, University of California, Berkeley, 1965.

40. DANTZIG, G.B., "Operations Research in the World of Today and Tomorrow", Operations Research Center, 1965-67, University of California, Berkeley; also in Management Science, January 1965.

41. DANTZIG, G.B., "Linear Control Processes and Mathematical Programming, SIAM Journal, Vol. 4, No. 1, 1966.

42. DANTZIG, G.B., FULKERSON, D.R., and JOHNSON, S., "Solution of a Large Scale Travelling Salesman Problem", JORSA, Vol. 2, No. 4, November 1954, p. 393.

43. DANTZIG, G.B., HARVEY, R., McKNIGHT, R., "Updating the Product Form of the Inverse for the Revised Simplex Method", Operations Research Center 1964-33, University of California, Berkeley.

44. DANTZIG, G.B., and MADANSKY, A., "On the Solution of Two-Stage Linear Programs under Uncertainty", Proceedings, Fourth Symposium on Mathematical Statistics and Probability, Vol. 1, 1961, pp. 165-176.

45. DANTZIG, G.B., and HAYS, W. Orchard, "Alternative Algorithm for the Revised Simplex Method using Product Form for the Inverse." The RAND Corp. RM-1268, 1953.

46. DANTZIG, G.B., and VAN SLYKE, R.M., "Generalized Upper Bounded Techniques for Linear Programming, I, II", Operations Research Center, University of California, Berkeley, ORC 64-17,18; also in Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems, March 16-18, 1964, pp. 249-261, and Journal of Computer and System Sciences, issue 2 forthcoming.

47. DANTZIG, G.B., and WOLFE, P., "The Decomposition Algorithm for Linear Programming", Econometrica, Vol. 29, No. 4, October 1961, Operations Research, Vol. 8, No. 1, January, February, October 1960.

48. DENNIS, D.E. (abstract) "A Multi-Period Transportation Problem", Econometrica, Vol. 31, 1963, p. 595.

49. DZIELINSKI, P. and GOMORY, R.E. (abstract) "Lot Size Programming and the Decomposition Principle" Econometrica, Vol. 31, 1963, p. 595.

50. EL AGIZY, M., "Programming Under Uncertainty with Discrete D.F.". ORC 64-13, University of California, Berkeley, July 1964, Ph.D. Thesis.

51. ELMAGHRABY, S.E., "An Approach to L.P. under Uncertainty", JORSA Vol. 7, No. 2, March, April 1959, p. 208.

52. ELECTRICITE DE FRANCE, "Programmes Lineaires Method de Decomposition", Direction de Etudes et Recherches, Paris, June 13, 1961.

53. ELECTRICITE DE FRANCE, "Programmation Lineaire, Methode de Dantzig et Wolfe, Programme Experimental", Direction des Etudes et Recherches, Paris, May 28th, 1962.

54. FORD, Lester, Jr., FULKERSON, D.R., "Suggested Computation for Maximal Multi-Commodity Network Flows", the RAND Corp., R-1114, Management Science, Oct. 1958, Vol. 5, No. 1.

55. FRISCH, R., "Tentative Formulation of the Multiplex Method for the Case of a Large Number of Basic Variables", Institute of Economics, University of Oslo, March 1962.

56. FULKERSON, D.R., "A Feasibility Criterion for Staircase Transportation Problems and Application to a Scheduling Problem" The RAND Corp., Report P. 1188, October 1957.

57. FAURE, P. et HUARD, P., "Résolution de Programmes Mathématiques à Fonction non Lineaire par la Méthode due Gradient Réduit", No. 36, 1965, Revue Francaise de Recherche Operationnelle.

58. GALE, David, "On Optimal Development in a Multi-Sector Economy", Operations Research Center, 1966-11, University of California, Berkeley, April 1966.

59. GASS, Saul I., "The Dualplex Method for Large-Scale Linear Programs", Operations Research Center, 1966-15, University of California, Berkeley, June 1966, Ph.D. Thesis.

60. GAUTHIER, J.M., "Le principle de Decomposition de Dantzig et Wolfe", Groupe de Travail, Mathematiques de Programmes Economiques, March 13, 1961.

61. GEOFFRION, A.M., "Direct Reduction of Large Concave Programs" Working Paper III, WMSI, UCLA, December 1966.

62. GILMORE, P.C. and GOMORY, R.E., "The Theory and Computation of Knapsack Functions", Operations Research, Vol. 14, No. 6, Nov-Dec. 1966.

63. GOMORY, R.E., "Large and Non-Convex Problems in Linear Programming" RC-765, IBM, August, 1962; see also [64].

64. GOMORY, R.E., "Large and Non-Convex Problems in L.P.", Proc. Sympos. Appl. Math. 15 (1963) 125-139.

65. GOMORY, R.E., and HU, T.C., "An Application of Generalized Linear Programming to Network Flows", IBM Research Report (1960) 50 p., SIAM Journal Vol. 10, No. 2, June 1962.

66. GOULD, S., "A Method of Dealing with Certain Non-Linear Allocation Problems Using the Transportation Technique", Operations Research Qu. Vol. 10, No. 3, September 1959.

67. GRAVES, R.L., WOLFE, P. (eds.) Recent Advances in Mathematical Programming, (McGraw-Hill Book Co., New York, 1963, 347 pp.)

68. HADLEY, G., Linear Programming, p. 437-508.

69. HALEY, R.B., "A General Method of Solution for Special Structure Linear Programmes", O.R. Qu. Vol. 17, No. 1, March 1966.

70. HARVEY, R.P., "Decomposition Principle for Linear Programming", Int. Jour. Comp. Math. May 1964 (20-35).

71. HEESTERMAN, A.R.G., "Partitioning a Phased Linear Programming Problem", Central Plan Bureau, Stolkweg, Working paper, April 1962.

72. HEESTERMAN, A.R.G., "Special Simplex Algorithm for Multi-Sector Problems", Series A #68, University of Birmingham, 1965.

73. HEESTERMAN, A.R.G., SANDEA, J., "Special Simplex Algorithm for Linked Problems", Management Science, 11,3 (January 1965) 420-428.

74. HELLERMAN, Eli, "The Dantzig-Wolfe Decomposition Algorithm as Implemented on a Large-Scale (Systems Engineering) Computer", Presented at Modern Techniques in the Analysis of Large-Scale Engineering Systems, Nov. 1965.

75. HERSHKOWITZ, M., and NOBLE, S.B., "Finding the Inverse and Connections of a Type of Large Sparse Matrix", Naval Research Logistics Quarterly, Vol. 12, No. 1, pp. 119-133.

76. HITCHCOCK, D.F., and MacQUEEN, J.B., "On Computing the Expected Discounted Return in a Markov Chain", Working Paper No. 105, Western Management Science Institute, UCLA, August 1966.

77. HU, T.C., "Multi-Commodity Network Flow", IBM Watson Research Center, Research Report RC-865, January 1963, and Operations Research, Vol. 11, 1963, pp. 344-360.

78. KANTOROVITCH, L.V. "Mathematical Methods in Organization and Planning of Production", Leningrad 1939, translated in Management Science, Vol. 6, 1960, pp. 366-422.

79. KAUL, R.N., "An Extension of Generalized Upper-Bounded Techniques for Linear Programming", Operations Research Center, University of California 1965-27, Berkeley, August 1965.

80. KLEE, Victor, "A Class of Linear Programming Problems Requiring a Large Number of Iterations", Numerische Mathematik, 7,313-321 (1965).

81. KRON, G., "Piecewise Solution of Large Scale Systems", General Electric, July, 1957.

82. KRON, G., "Piecewise Optimization of Linear Programming", General Electric, December 1958.

83. KRON, G., DIAKOPTICS - The Piecewise Solution of Large-Scale Systems, Macdonald Publishers 2 Portman, St., London W1.

84. KUNZI, H.P., and TAN, S.T., "Lineare Optimierung Grozer Systeme", Springer-Verlag, Berlin, 1966.

85. LABRO, C., "Efficiency and Degrees of Decomposition", Working Paper No. 98, Centre for Research in Management Science, University of California, Berkeley, August 1964.

86. LANCZOS, C., "Iterative Solution of Large-Scale Linear Systems", SIAM Journal, Vol. 13, No. 1, March 1958.

87. LAND, A.H., "A Problem of Assignment with Inter-related Costs", O.R. Qu., Vol. 14, No. 2, June 1963.

88. MACGUIRE, C.B., "Some Extensions of the Dantzig-Wolfe Decomposition Scheme", Center for Research in Management Science, University of California, Berkeley, Working Paper, No. 66, March 1963.

89. MALINVAUD, E., "Decentralized Procedures for Planning", Technical Report No. 15, Center for Research in Management Science, University of California, Berkeley, 1963.

90. MARKOWITZ, "The Elimination Form of the Inverse and its Application to Linear Programming", The Rand Corp. p. 680, 1955.

91. MURTY, K.G., "Two-Stage Linear Programming Under Uncertainty: A Basic Property of the Optimal Solution", O.R. Center 1966-4, University of California, Berkeley, February, 1966.

92. NASLUND and WHINSTON, "Model for Multi-Period Decision Making Under Uncertainty", M.S. 8 (1962).

93. NEMHAUSER, G.L., "Decomposition of Linear Programming by Dynamic Programming", Naval Research Logistics Quarterly, Vol. 11, June-Septembar 1964, pp. 191-196.

94. PARIKH, S.C. and JEWELL, W.S., "Decomposition of Project Networks", Management Science, Vol. 11, No. 3, January, 1965.

95. PARIKH, S.C., :Linear Dynamic Decomposition Programming of Optimal Long Range Operations of a Multiple Multi-Purpose Reservoir System", Presented at Fourth International Conference on O.R. 1966.

96. PEARSON, J.D., "Duality and a Decomposition Technique", SIAM Journal, Vol. 4, No. 1, 1966.

97. RECH, Paul, "Optimization by Price Communication between Leontief Expressions", O.R. Center 1964-35, University of California, Berkeley, December 1964.

98. RECH, Paul, "Decomposition and Interconnected Systems in Mathematical Programming", O.R. Center 1965-31, University of California, Berkeley, September, 1965, Ph.D. Thesis.

99. RITTER, K., "A Decomposition Method for Linear Programming Problems with Coupling Constraints and Variables", Math. Research Center, The University of Wisconsin #739, April 1967.

100. ROBERTS, J.E., "A Method of Solving a Particular Type of Very Large Linear Programming Problem", Proceedings of the First Annual Conference, Canadian Operational Research Society, University of Toronto, May 7-8, 1959, pp. 25-26.

101. ROBERTS, J.E., "A Method of Solving a Particular Type of Very Large Linear Programming Problem", Canadian Operational Research Journal, Vol. 1, No. 1. December 1963.

102. ROSEN, J.B., "Partition Programming", Notices, American Math. Soc. Vol. 7, 1960, pp. 718-719.

103. ROSEN, J.B., "Primal Partition Programming for Block Diagonal Matrices", Computer Science Division, School of Humanities and Sciences, Stanford University, Technical Report No. 32, November 1963; Numerische Math. 6,3(1964), 250-264.

104. ROSEN, J.B. and ORNEA, J.C., "Solution of Nonlinear Programming Problems by Partitioning", Shell Development Company, Emeryville, California, p-1115, June 1962.

105. SAIGAL, Romesh, "Block - Triangularization of Multi-Stage Linear Programs", O.R. Center 1966-9, University of California, Berkeley, March, 1966.

106. SAIGAL and SAKAROVITCH, "Compact Basis Triangularization for the Block Angular Structures", ORC, University of California, Berkeley (66-1).

107. SANDERS, J.L., "A Non-linear Decomposition Principle", Operations Research, 13 (1965), 266-67.

108. SHETTY, C.M., "On Analyses of the Solution to a Linear Programming Problem", O.R. Qu. Vol. 12, No. 2, June 1961.

109. SINHA, "Stochastic Programming, ORC, University of California, Berkeley, 63-22.

110. SIMONNARD, "Programmation Lineaire", Dunod (Paris) 1962.

111. STEINBERG, N., "Le Problème de Transport Généralisé à n dimensions, avec Données Aliatoires", Revue Francaise de Recherche Operationnelle, No. 26, 1963.

112. TCHENG, Tse-Hao, "Scheduling of a Large Forestry-Cutting Problem by Linear Programming Decomposition", Ph.D. Thesis, University of Iowa, August 1966.

113. THOMPSON, P.M., "Editing Large Linear Programming Matrices",JORSA, Vol. 4, No. 1, March 1957, pp. 97-100.

114. VAN SLYKE, R., "Mathematical Programming and Optimal Control", 1964, Ph.D. Thesis, University of California, Berkeley.

115. VAN SLYKE, R.M. and WETS, R., "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming", O.R. Center 1966-17, University of California, Berkeley, June 1966.

116. VAN SLYKE, R. and WETS, R., "Programming Under Uncertainty and Stochastic Optimal Control", SIAM Journal, Vol. 4, No. 1, 1966.

117. VARAIYA, P.P., "Decomposition of Large-Scale Systems", SIAM Journal, Vol. 4, No. 1, 1966.

118. WAGNER, Harvey, "A Linear Programming Solution to Dynamic Leontief Type Models", Management Science, Vol. 3, No. 3, 1957, p. 234-254.

119. WETS, R., "Programming under Uncertainty", 1964, Ph.D. Thesis, University of California, Berkeley.

120. WILDE, D.J., "Production Planning of Large Systems", Chemical Engineering Progress, January 1963.

121. WILLIAMS, J.D., "The Method of Continuous Coefficients, Parts I and II", Report No. ECC 60.3, Socony, 1960.

122. WILLIAMS, A.C., "A Treatment of Transportation Problems by Decomposition", J. Soc. Indust. Appl. Math. Vol. 10, No. 1, March 1962, pp. 35 - 48.

123. WHALEN, "Linear, Programming, Optimal Control, IRE Trans on Auto Control, Vol. AC-7, #4 (July, 1962) 1962 Ph.D. Thesis, University of California, Berkeley.

124. WOLFE, P., "Accelerating the Cutting Plane Method for Non-linear Programming", J. Soc. Indust. Appl. Math., Vol. 9, No. 3, September 1961, pp. 481-488.

125. WOLFE, P., and DANTZIG, G.B., "Linear Programming in a Markov Chain", Operations Research 10 (1962), 702-710.

126. WOOD, Marshall, K. and DANTZIG, G.B., "The Programming of Interdependent Activities: General Discussion", Econometrica, Vol 17, No. 3 & 4, July-October 1949, pp 193-199. Also in Activity Analysis of Production and Allocation, Koopmans, ed., 1951-1, pp. 15-18, and following chapter by Dantzig.

127. ZSCHAU, E.V.W., "A Primal Decomposition Algorithm for Linear Programming", Graduate School of Business, Stanford University, January 1967.

128. ZADEH, L., "Note on Linear Programming and Optimal Control, IRE Trans on Auto. Control, Vol AC-7, #4, July 1962.

CLASSIFIED BIBLIOGRAPHY ON LARGE-SCALE SYSTEMS 1949—1980

GENERAL BOOKS

ABADIE, J. (ed.), Integer and Nonlinear Programming, North Holland, 1970.

BELLMAN, R., Dynamic Programming, Princeton University Press, 1957.

BROISE, P., HUARD, P. and SENTENAC, J., Decomposition des Programmes Mathematiques, Menographies de Recherche Operationelle, Dunod, 1968.

BRYSON, A.E. and HO, Y.C., Applied Optimal Control, Blaisdell, Waltham, Mass, 1969.

COTTLE, R.W. and KRARUP, J. (eds.), Optimization Methods for Resource Allocation, English Universities Press, 1974.

DANTZIG, G.B., Linear Programming and Extensions, Princeton University Press, 1963.

DANTZIG, G.B. and VEINOTT, A.F., Jr. (eds.), Mathematics of the Decision Sciences, Parts 1 and 2 (2 books), American Mathematical Society, 1968.

DANTZIG, G.B., HAX, A., POMEROY, R., SANDERSON, R. and VAN SLYKE, R., Natural Gas Transmission System Optimization, American Gas Assoc., 1970.

DRUD, A., Methods for Control of Complex Dynamic Systems: Illustrated by econometric models, No. 27, Tech. Univ. of Denmark, Lyngby, 1976.

ERLANDER, S., Optimizing Nonlinear Models of Certain Transportation and Inventory Systems, Acta Polytechnica Scandinavica, Mathematics and Computing Machinery Series, No. 16, 1968.

HADLEY, G., Linear Programming, Addison-Wesley, Palo Alto, 1962.

HIMMELBLAU, D.M. (ed.), Decomposition of Large-Scale Problems, North Holland/American Elsevier, 1973.

KRON, G., Diakoptics, MacDonald, 1963.

LASDON, L.S., Optimization Theory for Large Systems, MacMillan, New York, 1970.

LJUNG, B. and SELMER, J., "Samordnad Planeringi Decentraliserade Företag en Studie av Dantzig & Wolfe Dekompositionsalgoritm", Stockholm Universitet, 1975.

LUHKS, R.A., "Large Scale Linear Programming Problems: Analysis and Generation", Ph.D. Dissertation, Comp. Sci./O.R. Center, Southern Methodist University, December 1972.

PARKER, J.M., "Management of Large-Scale Production by Linear Programming", Ph.D. Thesis, University of Texas at Austin, May 1974.

REID, J.K. (ed.), Large Sparse Sets of Linear Equations, Academic Press, 1970.

ROSEN, J.B., MANGASARIAN, O.L. and RITTER, K. (eds.), Nonlinear Programming, Academic Press, 1970.

ROSE, D.J. and WILLOUGHBY, R.A. (eds.), Sparse Matrices and Their Applications, Plenum Press, 1972.

WILLOUGHBY, R.A. (ed.), Sparse Matrices, Proceedings of Symposium at IBM Watson Research Center, September 1968.

WISMER, D.A. (ed.), Optimization Methods for Large-Scale Systems with Applications, McGraw-Hill, 1971.

SURVEY ARTICLES

BEALE, E.M.L., "Advanced algorithmic features for general mathematical programming systems", in Integer and Nonlinear Programming, North Holland, 1970.

DANTZIG, G.B., "Large-Scale Linear Programming", Tech. Report No. 67-8, Dept. of O.R., Stanford University, November 1967.

DANTZIG, G.B., "On the Need for a Systems Optimization Laboratory", in Optimization Methods for Resource Allocation, Crane, Russak & Co., New York, pp. 3-22, 1974.

GEOFFRION, A.M., "Large-Scale Linear and Nonlinear Programming", in Optimization Methods for Large-Scale Systems, edited by D.A. Wismer, McGraw-Hill, pp. 47-74, 1971.

GEOFFRION, A.M., "Elements of Large-Scale Mathematical Programming, Part I: Concepts", Management Science, Vol. 16, No. 11, pp. 652-675, 1970.

KALLIO, M., "On Large-Scale Linear Programming", Ph.D. Dissertation, Graduate School of Business, Stanford University, 1975.

LASDON, L.S., "A Survey of Large Scale Mathematical Programming", Tech. Memo. No. 349, Dept. of Operations Research, Case Western University, December 1974.

SEN, D.K., "Contracted Description by Aggregation", Work Paper No. 72, Dept. of Management Sciences, University of Waterloo, Ontario.

TAN, G-E. and SEN, D.K., "Contracted Description: An Approach to Systems Analysis", Work Paper No. 71, Dept. of Management Sciences, University of Waterloo, Ontario.

WHITE, W.W., "A Status Report on Computing Algorithms for Mathematical Programming", ACM Computing Survey, Vol. 5, No. 3, September 1973.

Proceedings of IIASA Planning Conference on Optimization and Control of Complex Dynamic Systems, Laxenburg, Austria, 1973.

GUB, G-GUB AND THE DECOMPOSITION PRINCIPLE

DANTZIG, G.B. and VAN SLYKE, R.M., "Generalized Upper Bounding Techniques", *J. Computer System Science*, Vol. 1, pp. 213-226, 1967.

DANTZIG, G.B. and WOLFE, P., "The Decomposition Algorithm for Linear Programming", *Econometrica*, Vol. 29, No. 4, October 1961.

HAUK, R.F., "The Biplex Method for Vertically Couoled Block-Angular Linear Programs with Many Coupling Variables", United States Steel Corp., June 1972.

HIRSHFELD, D.S., *Generalized Upper Bounding, Theory, Applications, Performance*, Management Science Systems, Inc., August 1970.

KALLIO, M. and PORTEUS, E.L., "A Note on the Relation Between the Generalized-GUB Technique and Dantzig-Wolfe Decomposition", Research Paper No. 265, Graduate School of Business, Stanford University, July 1975.

KAUL, R.N., "Comment on Generalized Upper Bounded Techniques in Linear Programming", Report ORC 65-21, Operations Research Center, University of California, Berkeley, July 1965.

LAWRENCE, J.A., "Parametric Programming with Extensions to Large-Scale Algorithms", Report ORC 73-18, Operations Research Center, University of California, Berkeley, September 1973.

MACK, D.R., "The Decomposition Method of Solving Linear Programs", Eng. Admin. Consulting Service, General Electric Report TIS 67 ETE I.

MADSEN, O.B., "A Numerical Comparison of Different Decomposition Algorithms", Institute of Math. Stat. & O.R., Tech. Univ. of Denmark.

MADSEN, O.E.G., "The Connection Between Decomposition Algorithms and Optimal Degree of Decomposition", Institute of Math. Stat. & O.R., Tech. Univ. of Denmark.

ORCHARD-HAYS, W., "Practical Problems in L.P. Decomposition", in *Decomposition of Large-Scale Problems*, North Holland, 1973.

WEIL, R.L., Jr. and KETTLER, P.C., "Rearranging Matrices to Block-Angular for Decomposition (and Other) Algorithms", Center for Mathematical Studies in Business and Economics, Report 6949, November 1969.

WINKLER, C., "Basis Factorization for Block-Angular Linear Programs: Unified Theory of Partitioning and Decomposition Using the Simplex Method", Ph.D. Dissertation, Operations Research Department, Stanford University, December 1974.

WINKLER, C., "On a Generalized-GUB Basis Factorization Algorithm for Block-Angular Linear Problems with Coupling Constraints", IIASA Working Paper WP-74-49, September 1974.

Management Science Systems, "Introduction to GUB (Preliminary)'', May 1969.

VARIANTS ON GUB, G-GUB AND THE DECOMPOSITION PRINCIPLE

AGANAGIC, M.I., "A Dual Decomposition Algorithm", Research & Development Centre for Automatic Control, Sarajevo, Yugoslavia.

AHMADI, H.C., "Block Partitioning and Decomposition in Large Scale Linear Programs; Analysis of Coupling", National Meeting of O.R. Society of America, San Juan, October 1974.

Anonymous, "Linear Fractional Programming in the Dual Decomposition Method".

AONUMA, T., "Two-Level Approach to Dynamically Decomposing of a Linear Planning Model", Working Paper No. 26, Institute of Econ. Research, Kobe University of Commerce, Tarumi, Kobe, Japan.

BAZARAA, M.S. and JAMIE, J.G., "A New Decomposition Technique", School of Industrial and Systems Engineering, Georgia Inst. of Tech.

BENDERS, J.R., "Partitioning Procedure for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol. 4, pp. 238-252, 1962.

BEN ISRAEL, A. and ROBERS, P.D., "A Decomposition Method for Interval Linear Programming", Management Science, Vol. 16, No. 5, pp. 374-387, 1970.

BISSCHOP, J. and MEERAUS, A., "Matrix Augmentation and Partitioning in the Updating of the Basis Inverse", Mathematical Programming, Vol. 13, No. 3, pp. 241-254, 1977.

BRUCKNER, P., "Large Scale Linear Programming with Hidden Cyclic Structure", NATO Conference on Appl. of Opt. Methods for Large Scale Resource Alloc. Problems, Elsinore, Denmark, July 1971.

DANTZIG, G.B., "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming", Econometrica, Vol. 23, No. 2, pp. 174-183, 1955.

ELSINGA, J., "Modified Convex Partition Programming", private communication.

GEOFFRION, A.W., "Generalized Benders Decomposition", JOTA, Vol. 10, No. 4, pp. 237-260, 1972.

HAUCK, R.F., "The Polyplex Method", United States Steel Corp., January 1973.

HELLERMAN, E. and RARICK, D., "The Partitioned Preassigned Pivot Procedure (P^4)", in Sparse Matrices and Their Applications, edited by D.J. Rose and P.A. Willoughby, Plenum Press, New York, pp. 67-76, 1972.

HO, J.K., "Nested Decomposition for Cyclic Models, Part II. Implementation and Computational Experience", Brookhaven National Laboratory Report BNL 19884, March 1975.

HO, J.K., "Optimal Design of Multi-Stage Structure; A Nested Decomposition Algorithm", SOL Tech. Report 74-2, Dept. of O.R., Stanford University, March 1974.

KALLIO, M. and PORTEUS, E.L., "Triangular Factorization and Generalized Upper Bounding Techniques", Operations Research, Vol. 25, No. 1, pp. 89-99, 1977.

OHSE, D., "A Dual Decomposition Method for Block-Diagonal Linear Programs", Zeitschrift für Operations Research, Band 17, pp. 55-67, 1973.

LOUTE, E., "A Revised Simplex Method for Block Structured Linear Programs", Ph.D. Dissertation, Catholic University of Louvain, Belgium, 1975.

McBRIDE, R.D., "A Bump Triangular Dynamic Factorization Algorithm for the Simplex Method", Working Paper No. 19, School of Business, University of California, December 1977.

PEROLD, A.F. and DANTZIG, G.B., "A Basis Factorization Method for Block Triangular Linear Programs", SOL Tech. Report 78-26, Dept. of O.R., Stanford University, 1978.

REARDON, K.J., "A Decomposition Method for the Solution of Dual-Angular Integer Programs", SOL Tech. Report 74-10, Stanford University, August 1974.

SCHRAGE, L., "Implicit Representation of Variable Upper Bounds in Linear Programming", Math. Prog. Study, Vol. 4, pp. 118-132, 1975.

WEINTRAUB, A. and INGRAM, B., "An Heuristic Decomposition Algorithm, Preliminary Report", Op. Research Center, University of California, Berkeley.

BLOCK TRIANGULARITY

DANTZIG, G.B., "Upper Bounds, Secondary Constraints and Block Triangularity in Linear Programming", Econometrica, Vol. 23, No. 2, pp. 174-183, 1955.

GRAVES, G.W. and McBRIDE, R.D., "The Factorization Approach to Large-Scale Linear Programming", Working Paper No. 208, Western Management Science Institute, UCLA, August 1973.

LINEAR OPTIMAL CONTROL THEORY AND DYNAMIC SYSTEMS

ABADIE, J., "Application of the GRG Algorithm to Optimal Control Problems", in Integer and Nonlinear Programming, edited by J. Abadie, Amsterdam, 1970.

ABADIE, J. and BICHARA, M., "Resolution Numerique de Certains Problemes de Commande Optimale", *Revue Francaise d'Automatique, Info. & Recherche Operationelle*, 1973.

ARROW and KURTZ, *Methods of Optimization over Time*.

BERKOVITZ, L.D., "On Control Problems with Bounded State Variables", *J. Math. Anal. Appl.*, Vol. 5, pp. 488-501.

BRYSON, A.E., Jr., DENHAM, W.F. and DREYFUS, S.E., "Optimal Programming Problems with Inequality Constraints I: Necessary Conditions for Extremal Solutions", *AIAA Journal*, Vol. 1, No. 11, pp. 2544-2550, 1963.

CHANG, S.S.L., "Optimal Control in Bounded Phase Space", *Automatica*, Vol. 1, pp. 55-67, 1963.

DANTZIG, G.B., "Optimal Solution of a Dynamic Leontief Model with Substitution", *Econometrica*, July 1955.

DANTZIG, G.B., "Linear Control Processes and Mathematical Programming", *SIAM Journal of Control*, Series A, Vol. 4, No. 1, pp. 56-60, 1966.

DANTZIG, G.B., and WOLFE, P., "Linear Programming in a Markov Chain", *Operations Research*, Vol. 10, pp. 702-710, 1962.

DREWS, W.P., "A Simplex-Like Algorithm for Continuous Time Linear Opt. Control Problems", NATO Conf. on Appl. of Opt. Methods for Large Scale Resource Alloc. Problems, Elsinore, Denmark, July 1971.

DREWS, W.P., HARTBERGER, R.J. and SEGER, R.B., "On Continuous Mathematical Programming", in *Optimization Methods in Resource Allocation*, edited by R.W. Cottle and J. Krarup, Crane, Russak & Co., New York, 1974.

DREYFUS, S., "Variational Problems with State Variable Inequality Constraints", Ph.D. Dissertation, Harvard University, 1962.

DUBOVSKY, S.V., DYUKALOV, A.N., *et al.*, "On Construction of an Optimal Economic Plan", private communication.

DYUKALOV, A.N. and ILYUTOVICH, A. Ye., "Qualitative Features in Economic Dynamics Trajectory", private communication.

GAMKRELIDZE, R.V., "Optimal Processes with Restricted Phase Coordinates", *Izv. Akad. Nauka SSSR*, Ser. Mat. 24, pp. 315-356, 1960.

HARTBERGER, R.J., "'Representation' Extended to Continuous Time", NATO Conf. on Appl. of Opt. Methods for Large Scale Resource Allocation Problems, Elsinore, Denmark, July 1971.

HO, J.K. and MANNE, A.S., "Nested Decomposition for Dynamic Models", *Mathematical Programming*, Vol. 6, pp. 121-140, 1974.

JACOBSON, D.H. and LELE, M.M., "A Transformation Technique for Optimal Control Problems with a State Variable Inequality Constraint", *IEEE Trans. Automat. Control*, Vol. AC-14, pp. 457-464, 1969.

KOHLHAGEN, S.N., "Staircasing Dinamico", Memo. 72-3, 1972.

LASDON, L.S., MITTER, S.K. and WAREN, A.D., "The Conjugate Gradient Method for Optimal Control Problems", IEEE Trans. Automat. Control, Vol. AC-12, pp. 132-138, 1967.

MEHRA, R.K. and DAVIS, R.E., "A Generalized Gradient Method for Optimal Control Problems with Inequality Constraints and Singular Arcs", IEEE Trans. Automat. Control, Vol. AC-17, No. 1, pp. 69-79, 1972.

PEROLD, A.F., "Fundamentals of a Continuous Time Simplex Method", Ph.D. Dissertation and SOL Tech. Report 78-26, Dept. of O.R., Standford University, December 1978.

PONTRYAGIN, L.S., BOLTYANSKII, V.G., GAMKRELIDGE, R.V. and MISCHCHENKO, E.F., The Mathematical Theory of Optimal Procasses, Interscience Publishers, New York, 1962.

PREKOPA, A., "Optimal Control of a Storage Level Using Stochastic Programming", Problems of Control and Info. Theory.

PROPOI, A., "Problems of Dynamic Linear Programming", Research Memo. RM-76-78, IIASA, November 1976.

PROPOI, A. and WILLEKENS, F., "A Dynamic Linear Programming Approach to National Settlement System Planning", IIASA Conference, December 1976.

PROPOI, A.I., "Linear Dynamic Programming", IIASA Conference, November 1974.

ROBBINS, N.M., "A Generalized Legendre-Clebsch Condition for the Singular Cases of Optimal Control", IBM Journal, July 1967.

SPEYER, J.L. and BRYSON, A.E., Jr., "Optimal Programming Problems with a Bounded State Space", AIAA Journal, Vol. 6, No. 8, pp. 1488-1491, 1968.

WOLLMER, R.D., "A Substitute Inverse for the Basis of a Staircase Structure Linear Program", Mathematics of Operations Research, Vol. 2, No. 3, pp. 230-239, 1977.

NESTED DECOMPOSITION

DANTZIG, G.B., "Solving Staircase Linear Programs by a Nested Block Angular Method", Tech. Report No. 73-1, Dept. of O.R., Stanford University, January 1973.

HO, J.K., "Nested Decomposition of Large-Scale Linear Programs with the Staircase Structure", SOL Tech. Report 74-4, Dept. of O.R., Stanford University, May 1974.

HO, J.K., "Nested Decomposition of a Dynamic Energy Model", Brookhaven National Laboratory Report BNL-20289, June 1975.

HO, J.K. and MANNE, A.S., "Nested Decomposition for Dynamic Models", Mathematical Programming, Vol. 6, pp. 121-140, 1974.

COLUMN GENERATION, CONVEX PROGRAMS, STOCHASTIC PROGRAMMING, AND SOME NONLINEAR PROGRAMMING

ABADIE, J., "Optimization Problems with Coupled Blocks", Symposium for National Economy Planning, Novosibirsk, USSR, June 1970.

ARMACOST, R.L., "Computational Experience with Optimal Value Function and Lagrange Multiplier Sensitivity in NLP", T-335,6, School of Engineer. and Applied Science, Washington University, May 1976.

ARROW, K.J., "General Economic Equilibrium: Purpose, Analytic Techniques, Collective Choice, Project on Efficiency of Decision Making in Economic Systems", Tech. Report No. 2, Harvard University, January 1973.

AUSLENDER, A., "Resolution Numerique d'Inegalities Variationelles", RAIRO, Feannee, R-2, pp. 67-72, 1973.

BEALE, E.M.L. and TOMLIN, J.A., "Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables" in O.R. '69 Proc. 5th Int'l. Conf. of O.R., edited by J. Lawrence.

CASTELLANI, G. and GIANNESSI, F., "Decomposition of Mathematical Programs by Means of Theorems of Alternatives for Linear and Nonlinear Systems", 9th Int'l. Symp. on Math. Prog., Budapest, August 1976.

CHANDRASEKARAN, R., "Row Generalized Linear Programs", Tech. Memorandum No. 310, O.R. Dept., School of Management, Case Western Reserve University, July 1973.

COTTLE, R.W., "Computational Experience with Large-Scale Linear Complementarity Problems", SOL Tech. Report 74-13, Dept. of O.R., Stanford University, September 1974.

DANTZIG, G.B. and MADANSKY, A., "On the Solution of Two-Stage Linear Programs under Uncertainty", in Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, pp. 156-176, 1961.

DRUD, A., "Optimizing Large Econometric Models - A Computational Method Based on Reduced Gradients", IMSOR, Tech. Univ. of Denmark, Lyngby.

DRUD, A., "Simulation with Large Nonlinear Simultaneous Systems - A Computational Method", J. No. 2189/034284, IMSOR, Tech. Univ. of Denmark, Lyngby, October 1970.

DRUD, A., "Application of Sparse Matrix Techniques in Large-Scale Nonlinear Programming", J. No. 2184/041613, IMSOR, Tech. Univ. of Denmark, Lyngby, August 1976.

FROMOVITZ, S., CHAIHO, K. and MORTON, R.L., "A Decomposition Algorithm for Nonconvex Mathematical Programming Problems", ORSA - TIMS - AIIE 1972 Joint National Meeting, November 1972.

GEOFFRION, A., "Primal Resource-Directive Approaches for Optimizing Nonlinear Decomposition Systems", Operations Research, pp. 375-403, May-June 1970.

GRINOLD, R.C., "Steepest Ascent for Large Scale Linear Programs", SIAM Review, Vol. 14, No. 3, July 1972.

HANSOTIA, B.J., "Programming under Uncertainty", Working Paper 203, College of Bus. Ad., Bradley University, August 1976.

HO, J.K., "Column Generation in Decomposition Algorithms", Applied Math. Dept., Brookhaven Nat. Lab., October 1974.

HODGSON, T.J. and LOVELAND, C.S., "A Partial Lagrange Multiplier Approach to a Resource Constrained C.P.M. Problem", Research Report 76-11, Industrial Systems Eng. Dept., University of Florida, Gainesville, May 1976.

HOGAN, W.W., MARSTEN, R.F. and BLANKENSHIP, J.N., "Boxstep: A New Strategy for Large Scale Mathematical Programming", Discussion Paper No. 46, Center for Math. Studies in Econom. and Management Sc., March 1973.

HOLLOWAY, C.A., "A Generalized Approach to Dantzig-Wolfe Decomposition for Concave Programs", Operations Research, Vol. 21, Nol. 1, pp. 210-220, January 1973.

JEROSLOW, R.G., "Cutting Planes for Complementary Constraints", private communciation.

KEITMAN, D.J. and MAGNANTI, T.L., "On the Number of Latent Subsets of Intersecting Collections", OR 012-72, O.R. Center, MIT, October 1972.

LUCAS, R., "von Neumann-Morgenstern Solutions", Tech. Report No. 246, Dept. of O.R., Cornell University, January 1975.

MAGNANTI, T.L., SHAPIRO, J.F. and WAGNER, M.H., "Generalized Linear Programming Solves the Dual", OR 019-73, Sloan School of Management, MIT, September 1973.

OETTLI, W., "Eine Allgemeine, Symmetrische Formulierung des Dekompositionsprinzips für Duale Paare Nichtlinearer Minimax- und Maximumprobleme", Zeitschrift für Operations Research, Band 18/7.

OETTLI, W., "Eine Allgemeine, Symmetrische Formulierung des Dekompositionsprinzips zur Lösung Konvex-Konkaver Sattelpunktprobleme".

PELEG, B., "Mavinvaud Prices for Efficient Stochastic Production Plans", Research Memo. No. 93, Inst. of Math., Hebrew University of Jerusalem, April 1974.

PETERSON, E.L., "The Decomposition of Large (Generalized) Geometric Programming Problems by Tearing", Discussion Paper No. 13, Center for Math. Studies in Econ. and Management Science, September 1972.

SHOVEN, J.B., "A Proof of the Existence Theorem of a General Equilibrium with Ad Valorem Commodity Taxes", Journal of Economic Theory, Vol. 8, May 1974.

SHOVEN, B. and WHALLEY, J., "General Equilibrium with Taxes: A Computational Procedure and an Existence Proof", Review of Economic Studies, Vol. XL(4), pp. 475-489, October 1973.

TABAK, D., "Hierarchical Systems Optimization Study", Hartford Grad. Center, Rensselaer Polytechnic Institute of Connecticut, Conn., May 1973.

VAN SLYKE, R.M. and WETS, R.J.B., "L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming", Report ORC 66-17, Operations Research Center, University of California, Berkeley, July 1966.

WILSON, R., "Computing Equilibria of Two-Person Games from the Extensive Form", Work Paper No. 176, Stanford University, May 1970 (also SIAM J. Appl. Math., Vol. 21, Nol. 1, July 1971).

SPARSE MATRIX TECHNIQUES

BEALE, E.M.L., "Sparseness in Linear Programming", in Large Sparse Sets of Linear Equations, edited by J.K. Reid, Academic Press, New York, pp. 1-16, 1971.

DANTZIG, G.B., HARVEY, R.P., McKNIGHT, R.D. and SMITH, S.S., "Sparse Matrix Techniques in Two Mathematical Programming Codes", in Sparse Matrix Proceedings, IBM Research Center, 1969.

KOEHLER, G.J., WHINSTON, A.B. and WRIGHT, G.P., "Matrix Iterative Techniques in Large Scale Linear Programming", Kannert Grad. School of Ind. Admin., Purdue University.

FORREST, J.J.H. and TOMLIN, J.A., "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method", Math. Progr., Vol. 2, pp. 263-278, 1972.

GILL, P.E. and MURRAY, W., "A Numerically Stable Form of the Simplex Algorithm", Nat. Physical Lab., August 1970.

GRIGORIADIS, M.D., "Unified Pivoting Procedures for Large Structured Linear Systems", NATO Institute of Decomp., July 1972.

LOUTE, E.F., "A General Inverse Comactification Method for Large Scale Structured Linear Programs", Tech. Report No. 269, Dept. of Operations Research, Cornell University, Ithaca, N.Y., July 1975.

MAGNANTI, T.L., "Optimization for Systems", Tech. Report No. 119, O.R. Center, MIT, November 1975.

McBRIDE, R.D., "A Dynamic Factorization Algorithm", ORSA/TIMS Meeting, Las Vegas, November 1975.

SAUNDERS, M.A., "Large-Scale Linear Programming Using the Cholesky Factorization", Stan-CS-72-252, January 1972.

LARGE NETWORKS AND RELATED PROBLEMS

BALAS, E., "Solution of Large-Scale Transportation Problems through Aggregation", Operations Research, Vol. 13, pp. 82-93, 1965.

BANSAL, P.P. and JACOBSEN, S.E., "An Algorithm for Optimizing Network Flow Capacity under Economies of Scale", *Journal of Optimization Theory and Applications*, Vol. 15, No. 5, 1975.

CHEN, S. and SAIGAL, R., "A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints", Bell Laboratories, Holndel, N.J.

DANTZIG, G.B., "On the Shortest Route through a Network", *Management Science* , Vol. 6, pp. 187-190, 1960 (also in *Some Topics in Graph Theory*, edited by D.R. Fulkerson, MAA Studies No. 11, 1975).

DANTZIG, G.B. "All Shortest Routes in a Graph", Tech. Report 66-3, Dept. of O.R., Stanford University, November 1966 (also in *Theories des Graphes*, International Symposium, Rome, Italy, pp. 19-92, July 1966, published by Dunod, Paris).

DANTZIG, G.B., BLATTNER, W.O. and RAO, M.R., "All Shortest Routes from a Fixed Origin in a Graph", Tech. Report 66-2, Dept. of O.R., Stanford University, November 1966 (also in *Theorie des Graphes*, International Symposium, Rome, Italy, pp. 85-90, July 1966, published by Dunod, Paris).

DANTZIG, G.B., MAIER, S.F. and LANSDOWNE, Z.F., "The Application of Decomposition to Transportation Network Analysis", Central Analysis Corp., Palo Alto, March 1976.

DANTZIG, G.B. and RAMSER, J.H., "The Truck Dispatching Problem", *Management Science*, Vol. 6, pp. 80-91, 1959.

ESCHENBACH, T. and CARLSON, R.C., "The Capacitated Multi-Period Location-Allocation Problem", SOL Tech. Report 75-27, Stanford University, October 1975.

FORD, L.R. and FULKERSON, D.R., *Flows in Networks*, Princeton University Press, 1962.

FORD, L.R. and FULKERSON, D.R., "A Suggested Computation for Maximal Multi-Commodity Network Flows", *Management Science*, Vol. 5, No. 1, pp. 97-101, October 1958.

FULKERSON, D.R., "An Out-of-Kilter Method for Minimal Cost Flow Problems", *J. Soc. Ind. Appl. Math.*, Vol. 9, pp. 18-27, 1961.

FULKERSON, D.R., "Increasing the Capacity of a Network", *Management Science*, Vol. 5, pp. 472-482, 1959.

GLOVER, F. and KLINGMAN, D., "New Advances in the Solution of Large-Scale Network and Network-Related Problems", Business Research Division, Graduate School of Business Administration, University of Colorado.

GLOVER, F. and KLINGMAN, D. "Effective Treatment of Degeneracy in Large-Scale Assignment and Transshipment Problems", Research Report CCS 247, Center for Cybernetic Studies, September 1975.

GLOVER, F and KLINGMAN, D., "Important Practical Misconceptions of Optimizing Large Scale Assignment and Transportation Problems", Research Report CCS 195, Center for Cybernetic Studies, October 1974.

GLOVER, F and KLINGMAN, D., "A Practitioner's Guide to the State of Large Scale Network and Network Related Problems", AFIPS Conference Proceedings, Vol. 45, 1976.

GRIGORIADIS, M.D. and WHITE, W.W., "Computational Experience with a Multi-commodity Network Flow Algorithm", in Optimization Methods for Resource Allocation, edited by R. Cottle and J. Krarup, Crane, Russak & Co., New York, pp. 205-226, 1974.

GOMORY, E.E. and HU, T.C., "An Application of Generalized Linear Programming to Network Flows", J. Soc. Indust. Appl. Math., Vol. 10, No. 2, June 1962.

HARTMAN, J.K. and LASON, L.S., "A Generalized Bounding Algorithm for Multi-commodity Network Flow Problems", Networks, Vol. 1, pp. 333-354, 1972.

JORGENSON, N.O., "Some Aspects of the Urban Traffic Assignment Problem", ITTE Report, Berkeley, July 1963.

KLESSIG, R.W., "An Algorithm for Non-Linear Multi-commodity Flow Problems", Networks, Vol. 4, pp. 343-355, 1974.

LANSDOWNE, Z.F., DANTZIG, G.B., HARVEY, R.P. and McKNIGHT, R.D., "Development of an Algorithm to Solve Multi-Stage Games", Control Analysis Corporation (prepared for the Office of Studies and Analysis, U.S. Air Force), May 1973.

LANSDOWNE, Z.F., DANTZIG, G.B., HARVEY, R.P. and McKNIGHT, R.D., "Approximate Solutions of Multi-Stage Network Games", Control Analysis Corporation (prepared for the Office of Studies and Analysis, U.S. Air Force), August 1973.

LASDON, L.S. and TERJUNG, R., "An Efficient Algorithm for Multi-Item Scheduling", Operations Research, Vol. 19, No. 4, pp. 949-969, 1971.

MAIER, S.F., "A Compact Inverse Scheme Applied to a Multicommodity Network with Resource Constraints", in Optimization Methods for Resource Allocation, Crane, Russak & Co., New York, pp. 179-204, 1974.

MAIER, S.F., "The Multi-Commodity Flow Problem L-U Decomposition Approach", private communication, May 1969.

MAIER, S.F., "Solving the Traffic Assignment and Sub-Area Focusing Problems by Geographic Decomposition", GSBA Working Paper No. 155, Duke University, September 1975.

MIDLER, J.L.,"Investment in Network Expansion under Uncertainty", Transportation Research, Vol. 4, pp. 267-280, October 1970.

MOSHER, W.W., "A Capacity Restraint Algorithm for Assigning Flow to a Transportation Network", HRB Record, Vol. 6, 1963.

MURPHY, R.A. and ALLEN, R.E., "An Assignment Problem with Solution Dependent Costs", Management Systems Division, Procter & Gamble Co., Cincinnati, Ohio, 1976.

NGUYEN, S., "An Algorithm for the Traffic Assignment Problem", Transportation Science, Vol. 8, pp. 203-216, August 1974.

SWEENEY, D.J. and MURPHY, R.A., "A Decomposition Principle for Integer Programs", Procter & Gamble Co., Cincinnati, Ohio, 1976.

TOMLIN, J.A., "Minimum-Cost Multi-Commodity Network Flows", Operations Research, Vol. 14, pp. 45-51, 1966.

WILLIAMS, A.C., "A Treatment of Transportation Problems by Decomposition", J. Soc. Indust. Appl. Math., Vol. 1, pp. 35-48, March 1962.

WOLLMER, R.D., "Multi-commodity Networks with Resource Constraints: The Generalized Multi-commodity Flow Problem", Networks, Vol. 1, pp. 245-264, 1972.

ZIMIN, I., "Optimal Control Theory Problems with Network Constraints and Their Applications", IIASA Research Memo. RM-76-13, March 1976.

APPLICATIONS

AGUADO, E. and REMSON, I., "Linear Programming and Finite Differences in the Solution of Subsurface - Flow Problems: Preliminary Application", Dept. of Appl. Earth Sciences and Geology, Stanford University.

ARMSTRONG, R.D. and WILLIS, C.E., "Simultaneous Investment-Allocation. An Application of Generalized Benders Decomposition to Work Planning", Research Report CCS 243, Center for Cybernetic Studies, University of Texas, Austin, August 1975.

BALINSKI, M.L. and WOLFE, P., "On Benders' Decomposition and a Plant Location Problem", Mathematica Working Paper, ARO-27, 1963.

BALINTFY, J.L., "Large Scale Programming Properties of Menu Planning and Scheduling", Grad. School of Bus. Ad., University of Mass., July 1971.

BARRETT, E.B. and DEVICH, R.N., "Linear Programming Compensation for Space-Varient Image Degradation", SPIE/OSA, Vol. 74, 1976.

BUZBY, B.R., STONE, B.J. and TAYLOR, R.L., "Computational Experience with a Non-Linear Distribution Problem", Union Carbide, January 1965.

DANTZIG, G.B., "Application of the Simplex Method to a Transportation Problem", in Activity Analysis Production and Allocation, edited by T.C. Koopmans, John Wiley, New York, pp. 359-373, 1951.

DANTZIG, G.B., "A Machine Job Scheduling Model", Management Science, Vol. 6, pp. 191-196, 1960.

DANTZIG, G.B., BLATTNER, W.O. and RAO, M.R., "Finding a Cycle in a Graph with Minimum Cost to Time Ratio with Application to a Ship Routing Problem", Tech. Report 66-1, Dept. of O.R., Stanford University, November 1966 (also in Theories des Graphes, International Symposium, Rome, Italy, pp. 77-84, July 1966, published by Dunod, Paris).

DANTZIG, G.B. and FERGUSON, A.R., "The Allocation of Aircraft to Routes - An Example of Linear Programming under Uncertain Demand", Management Science, Vol. 3, pp. 45-73, 1956 (also in Analysis of Industrial Operations, edited by Bowman and Fetter, Richard D. Irwin, Homewood, Ill., 1959).

DANTZIG, G.B., HAX, A., POMEROY, R. SANDERSON, R. and VAN SLYKE, R., "Natural Gas Transmision System Optimization", American Gas Association, April 1970.

DANTZIG, G.B. and JOHNSON, D.L., "Maximum Payloads per Unit Time Delivered through an Air Network", Operations Research, Vol. 12, pp. 230-236, 1964.

DANTZIG, G.B. and FULKERSON, D.R., "Minimizing the Number of Tankers to Meet a Fixed Schedule", Naval Research Logistics Quarterly, Vol. 1, pp. 217-222, 1954.

GASS, S.I. and SISSON, R., "A Guide to Models in Governmental Planning and Operations", Environmental Protection Agency, August 1974.

GEOFFRION, A and GRAVES, G., "Multicommodity Distribution System Design by Benders Decomposition", Working Paper No. 209, Western Management Science Institute UCLA, August 1973.

GILMORE, P.C. and GOMERY, R.E., "A Linear Programming Approach to the Cutting Stock Problem", Operations Research, Vol. 9, pp. 849-859, 1961.

GILMORE, P.C. and GOMERY, R.E., "Multistage Cutting Stock Problems of Two and More Dimensions", Operations Research, pp. 94-120, 1964.

GLASSEY, C.R., "Dynamic Linear Programs for Production Scheduling", Operations Research, pp. 45-56, January-February, 1971.

GLOVER, F. and KLINGMAN, D., "Network Applications in Industry and Government", Center for Cybernetic Studies, University of Texas, December 1974.

GUBBY, H., "An Overview of Vehicular Scheduling Problems", Tech. Report 103, Operations Research Center, MIT, September 1974.

HAUCK, R.F. "Optimal Long Range Coal Resource Allocation", United States Steel Corporation, June 1971.

HAUCK, R.F., "New Optimal Shipment of Cement by Barge and Rail", Operations Research, July 1972.

HAVERLY, C.A., "Modern Practice in Facility Location", SIGMAP, No. 18, pp. 38-44, February 1975.

HIRCHE, J., "Über Zwei Modifikationen des Dekompositionsalgorithmus von Williams", M, H.6, pp. 123-129, Wiss. Z. Univ. Halle, 1970.

KULIKOWSKI, R. "Optimum Control of Environment Development System", Automatica, Vol. 9, pp. 357-365, 1973.

KLINGMAN, D., RANDOLPH, P.H. and FULLER, S.W., "A Cottonpickin' Cotton Ginning Problem", Research Report CS 175, Center for Cybernetic Studies, University of Texas, Austin, May 1974.

KLINGMAN, D., RANDOLPH, P.H. and FULLER, S.W., "A Cotton Ginning Problem", Operations Research, Vol. 24, No. 4, pp. 700-717, July 1976.

KUTCHER, G.P., "On Decomposing Price-Endogenous Models", January 1972.

LANSDOWNE, Z.F., "Formulation and Solution of the Routing and Loading Problem", Mathematica (prepared for the U.S. Army Programming and Planning Analysis Directorate), July 1970.

LASDON, L.S., "Generalized Upper Bounding Methods in Production Scheduling and Distribution", Tech. Report No. 234, O.R. Dept., Case Western Reserve University, June 1971.

LASDON, L.S., "Generalized Upper Bounding Methods in Production Scheduling and Distribution", in Optimization Methods for Resource Allocation, Crane, Russak & Co., New York, pp. 25-42, 1974.

LEVIN, A., "Some Fleet Routing and Scheduling Problems for Air Transportation Systems", Report R-68-5, Dept. of Aeronautics and Astronautics, MIT, January 1969.

MAIER, S.F. and VANDERWEIDE, J.H. "Capital Budgeting in the Decentralized Firm", Management Science, December 1976.

MANNE, A.S., "Electricity Investments under Uncertainty - Waiting for the Breeder", Stanford University, February 1973.

MOREY, R.C., "A Mathematical Model for Optimal Network Construction of Transportation and Other Service Systems", TN-RMR-35, Stanford Research Institute, June 1968.

NAZARETH, L. "A Resource Allocation Model Using Dantzig-Wolfe Decomposition and Dynamic Programming", Masters Thesis, Dept. of O.R., University of California, Berkeley.

OLIVER, R.M. and HOPKINS, D.S.P., "Instructional Costs of University Outputs", Report ORC 73-16, Operation Research Center, University of California, Berkeley, July 1973.

PARIKH, S.C., "Linear Dynamic Decomposition Programming of Optimal Large Range Operation of a Multiple Multi-Purpose Reservoir System", Report ORC 66-38, Operations Research Center, University of California, Berkeley, September 1966.

PARIKH, S.C. and JEWELL, W.S., "Decomposition of Project Networks", Management Science, Vol. 11, No. 3, June 1965.

PINGRY, D. and WHINSTON, A., "Planning for Pollution Control. A Critical Survey".

RUSSELL, C.S., SPOFFORD, W.O., Jr. and HAEFELE, E.T., "Environmental Quality Management in Metropolitan Areas", April 1972.

RUSSELL, C.S. and SPOFFORD, W.O., Jr., "A Quantitative Framework for Residual Management Decisions" in Environmental Quality Analysis: Theory & Method in the Social Sciences, edited by A.V. Kneese and B.T. Bower, 1972.

ROTHFORD, B., FRANK, H., ROSENBAUM, D.M., STEIGLITX, K. and KLEITMAN, D.J., "Optimal Design of Offshore Natural-Gas Pipeline Systems", Operations Research, Vol. 18, No. 6, pp. 992-1020, 1970.

SIMPSON, R.W., "Computerized Schedule Construction for an Airline Transportation System", Lab. Report FT-66-3, MIT, December 1966.

SPOFFORD, W.O., Jr., CLIFFORD, S.R. and KELLY, R.A., "Operational Problems in Large Scale Residuals Management Models", private communication, February 1973.

VENTURA, E., "Mixed Integer Telephone Capacity Model (with Basic Data)", private communication.

WILDE, D.J., "Production Planning of Large Systems", Chemical Engineering Progress, Vol. 59, No. 1, January 1963.

ZADEH, N., "Construction of Efficient Tree Networks: The Pipeline Problem", Networks, Vol. 3, pp. 1-31, 1973.

"Gestion Optimale des Reservoirs d'une Vallee Hydraulique", Electricite de France: Etudes Economiques Generales, M. 171, October 1973.

"Research on the Energy-Modeling Process, Research Proposal", National Bureau of Economic Research, Inc., Cambridge, Mass., November 1974.

KALLIAUER, A., "Compactification and Decomposition Methods for NLP Problems with Nested Structures in Hydro Power System Planning", Österreichische Elektrizitätswirtschaft AG, Vienna, Austria.

SOFTWARE DEVELOPMENT

ASAP, A Scientific Application Programmer, National CCS, Inc., Connecticut, October 1972.

BAYER, R and WITZGALL, C., "A Data Structure Calculus for Matrices", Information Sciences Report No. 20, Scientific Research Laboratories, May 1968.

BAYER, R. and WITZGALL, C., "Some Complete Calculi for Matrices", Comm. ACM, Vol. 13, No. 4, April 1970.

BEALE, E.M.L., "Fortran Programming Conventions".

BRADLEY, G.H., BROWN, G.G. and GRAVES, G.W., "Design and Implementation of Large Scale Primal Networks Transshipment Algorithms", Report No. NPS55 BZBW 76091, Naval Post Grad. School, September 1976.

BROWN, K., HILLSTROM, K., MINKOFF, M., NAZARETH, L., POOL, J. and SMITH, B., "A Minpack Progress Report", Technical Memo. No. 255, Argonne National Laboratory, March 1975.

DeBUCHET, J., The Development of a Large Scale Mathematical Programming System, Proceedings of 1968 ACM National Conference.

HARVEY, R.P., "DYGAM — A Computer System for the Solution of Dynamic Programs", Control Analysis Corporation (prepared for the International Institute for Applied Systems Analysis), August 1974.

KAILATH, T., "Some New Methods for Solving Linear Equations", Information Systems Lab., Dept. of Electrical Eng., Stanford University, May 1976.

NUNAMAKER, J.F., Jr., SWENSON, D.E. and WHINSTON, A.B., "Specifications for the Development of a Generalized Data Base Planning System", National Computer Conference, 1973.

ORCHARD-HAYS, W.M., Structure of Mathematical Programming Systems, Proceedings, 1968 ACM National Conference.

REYNOLDS, G.H., "An Algorithm for Compactifying Nonzero Entries in Large Scale Linear Programs", ORSA-TIMS-AIIE Joint National Meeting, Atlantic City, N.J., November 1972.

UPPLULURI, V.B.R. and KIRK, B.L., "Exlicit Inverses of Some Special Matrices (with a Few Computer Programs)", Oak Ridge National Laboratory, February 1976.

"XMP: A Structured System of Subroutines for Mathematical Programming (Research Proposal)", National Bureau of Economic Research, Cambridge, Mass., April 1975.

"Introduction to MPS III", National CSS, Inc., Connecticut.

LIST OF PARTICIPANTS

WORKSHOP ON LARGE-SCALE LINEAR PROGRAMMING AT IIASA

Dr. Phillip Abrahamson
Department of Industrial Engineering
and Operations Research
University of Massachusetts
Amherst, Massachusetts 09002
USA

Dr. Tatsuo Aonuma
Kobe University of Commerce
Department of Management Sciences
Tarumi 655, Kobe
Japan

Dr. Michael Bastian
Rheinisch–Westfälische Technische
Hochschule Aachen
Buechel 29–31
D-51 Aachen
FRG

Prof. Klaus Beer
Mathematical Programming Department
of the Technische Hochschule
Karl-Marx-Stadt
GDR

Dr. Johannes Bisschop
Development Research Center
The World Bank
1818 H Street
Washington, D.C. 20433
USA

Prof. Gerald G. Brown
Naval Postgraduate School
Operations Research Code 55BW
Monterey, CA 93940
USA

Dr. Vladimir A. Bulavskiy
Institute of Mathematics
630090 Novosibirsk
USSR

Prof. George B. Dantzig
Department of Operations Research
Stanford University
Stanford. CA 94305
USA

Prof. Marshall L. Fisher
Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104
USA

Prof. Robert Fourer
Northwestern University
Department of Industrial Engineering and
Management Sciences
Evanston, Illinois 60201
USA

Dr. Philippe Gille
CORE
Université Catholique de Louvain
34 Voie du Roman Pays
B-1348 Louvain-la-Neuve
Belgium

Dr. Zsolt Harnos
Bureau for Systems Analysis of the State
Office for Technical Development
P.O. Box 565
H-1374 Budapest
Hungary

Dr. James K. Ho
Applied Mathematics Department
Brookhaven National Laboratory
Associated Universities, Inc.
Upton, N.Y. 11973
USA

Prof. Pierre Huard
EDF Direction des Etudes et Recherches
1, Avenue du General de Gaulle
F-92141 Clamart
France

Dr. K.V. Kim
Central Economic Mathematical Institute
Academy of Sciences of the USSR
Moscow V-333
Vavilova 44/2
USSR

Ir. T. Knol
Department of Applied Mathematics
Technical High School of Twente
Postbus 217
NL-7500 AE Enschede
The Netherlands

Dr. Gerhard Knolmayer
Institut für Betriebswirtschaftslehre
Universität Kiel
Olshausen Straße 40–60
Kiel
FRG

Dr. A.T. Langeveld
Royal Shell Laboratory Amsterdam
Postbus 3003
NL-1003 AA Amsterdam
The Netherlands

Dr. Etienne Loute
CORE
Université Catholique de Louvain
34 Voie du Roman Pays
B-1348 Louvain-la-Neuve
Belgium

Prof. Laszlo Lovász
Bolyai Institute
Josef Attila University
Aradi Vertanuk tese 1
H-6720 Szeged
Hungary

Prof. Richard D. McBride
University of Southern California
School of Business
Finance and Business Economics
Department
Los Angeles, CA 90007
USA

Dr. Dietrich Ohse
Johann Wolfgang Goethe Universität
Senckenberganlage 31
D-6000 Frankfurt/Main
FRG

Dr. William Orchard-Hays
Department of Energy
12th and Pennsylvania Avenue, NW
Mail Stop 4530
Energy Information Administration
Washington, D.C. 20461
USA

Dr. André F. Perold
Harvard University
Graduate School of Business Administration
Soldiers Field
Boston, Massachusetts 02163
USA

Dr. Michael A. Saunders
Department of Operations Research
Stanford University
Stanford, CA 94305
USA

Prof. Yves Smeers
CORE
Université Catholique de Louvain
34 Voie du Roman Pays
B-1348 Louvain-la-Neuve
Belgium

Dr. Janos Stahl
Institute for Applied Computer Techniques
Pf. 227
H-1536 Budapest
Hungary

Dr. Beate Strazicky
Computer and Automation Institute
Hungarian Academy of Sciences
H-1111 Budapest XI
Hungary

Prof. Richard Sutherland
Mathematics Department
Dalhousie University
Nova Scotia
Canada B3H 4H8

Dr. Eugeniusz Toczyłowski
Institute of Automatic Control
Technical University of Warsaw
Nowowiejska 15/19
PL-00-665 Warsaw
Poland

Dr. Stanisław Walukiewicz
Systems Research Institute
Polish Academy of Sciences
Newelska 6
PL-01-447 Warsaw
Poland

IIASA Participants from the Systems and Decision Sciences Area

Michael Dempster
Yuri Ermoliev
Markku Kallio
Evgeni Nurminski
Andrzej Wierzbicki