ON A GENERALIZED GUB BASIS FACTORIZATION ALGORITHM FOR BLOCKANGULAR LINEAR PROBLEMS WITH COUPLING CONSTRAINTS

Carlos Winkler

September 1974 WP-74-49

On A Generalized GUB Basis Factorization Algorithm for Blockangular Linear Problems with Coupling Constraints

by

Carlos Winkler*

Introduction

An extension of the Generalized Upper Bounding Technique to blockangular linear problems with coupling constraints was proposed by Kaul [5] in 1965. Similar methods were developed independently by Müller-Mehrbach [7] and Bennett [1]. They differ from other partitioning and decomposition algorithms in that they are primal methods that follow the same path as the Primal Simplex Method, using essentially a different representation for the basis inverse. As has been pointed out by Grigoriadis [4], other partitioning methods for this same class of problems, like Rosen's Primal Partitioning Procedure [10] or Ohse's Dual Algorithm [8], can be viewed as using this same basis inverse representation with different pivot selection strategies.

In the following we review the basic basis factorization for blockangular linear problems with coupling variables and from its analysis we determine an efficient strategy for large problems, which for a large class of problems should reduce the computations in the backward and forward transformation

* The research for this paper was performed at the Systems Optimization Laboratory, Stanford University, Stanford, California.

and the average time spent in inversions and reinversions.

At the end we present some preliminary computational results which tend to confirm our theoretical conclusions.

Review of the Generalized GUB (G-GUB) Basis Factorization for Blockangular Linear Problems with Coupling Constraints*

Consider the following linear program

$$\begin{array}{lrcl} & \text{Max } z & & \\ \text{s.t.} & U_0 z + H_0 x_0 + H_1 x_1 + \cdots + H_k x_k & = & b_0 \\ & D_1 x_1 \qquad\qquad & = & b_1 \\ & \ddots \qquad & & \\ & D_k x_k & = & b_k \\ & x_0 \geq 0 \;,\; x_1 \geq 0 \;, \ldots ,\; x_k \geq 0 & & \end{array}$$

where $x_i \varepsilon R^{n_i}$, $b_i \varepsilon R^{m_i}$, D_i $m_i \times n_i$, H_i $m_0 \times n_i$, $v_0 \varepsilon R^{m_0}$, z scalar.

We assume the system has full rank. Then if B_T is any basis for P, it can be ordered so that it has the following structure

$$B_T = \begin{array}{|c|c|c|c|c|} \hline \tilde{H}_0 & \tilde{H}_1 & \tilde{H}_2 & \cdots & \tilde{H}_k \\ \hline & G_1 & & & \\ & & G_2 & & \\ & & & \ddots & \\ & & & & G_k \\ \hline \end{array}$$

---

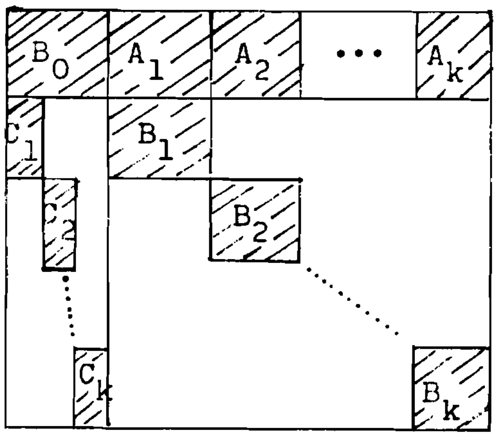
*Our presentation is on lines similar to Lasdon's [6].

Proposition 1: G_i, for $i = 1,\ldots,k$, can be partitioned, rearranging the columns if necessary, as $G_i = (B_i, C_i)$ where B_i is $m_i \times m_i$ and nonsingular.

Proof: Since B_T is nonsingular it has full row rank. This implies that G_i has full row rank for $i = 1,\ldots,k$. That is, G_i has rank m_i, and hence it contains at least one set of m_i linearly independent columns. Let B_i consist of such a set and C_i of the remaining columns in G_i (if any). ||

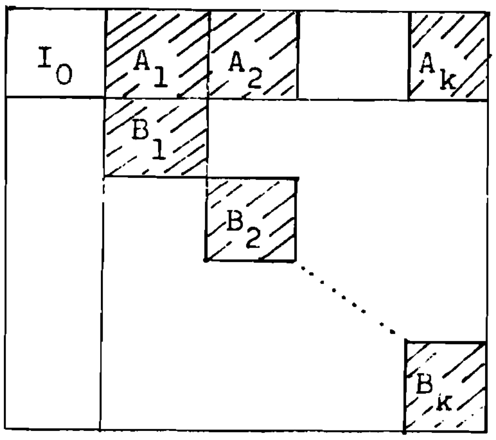
By proposition 1 we can rearrange the columns of B_T so that

B_T =



Let

B_N =



and

$$\hat{B}_i = \begin{bmatrix} I_0 & & & A_i & & & \\ & I_1 & & & & & \\ & & I_{i-1} & & & & \\ & & & B_i & & & \\ & & & & I_{i+1} & & \\ & & & & & \ddots & \\ & & & & & & I_k \end{bmatrix} \qquad i = 1,\dots,k$$

Proposition 2: $B_N = \prod_{i=1}^{k} \hat{B}_i$, where the product can be taken in any order.

Proof: Take an arbitrary order and carry out the matrix multiplications and verify that the product is equal to B_N. (Will not be done here.)

Corollary: B_N is nonsingular and $B_N^{-1} = \prod_{i=1}^{k} \hat{B}_i^{-1}$ where the product can be taken in any order.

Notice that

$$\hat{B}_i^{-1} \begin{pmatrix} I_0 & -A_iB_i^{-1} & \\ & B_i^{-1} & \\ & & I_k \end{pmatrix}$$

Proposition 3: If d is a column belonging to block i, i.e. $d^T = (d_0,0,\dots,0,d_i,0,\dots,0)$, then

$$\hat{d} = B_N^{-1}d = \hat{B}_i^{-1}d = (d_0 - A_iB_i^{-1}d_i,0,\dots,0,B_i^{-1}d_i,0,\dots,0)^T \quad .$$

In particular, if d is a column in block 0, then $\hat{d} = d$.

<u>Proof</u>: $\hat{d} = B_N^{-1}d$ is equivalent to $B_N\hat{d} = d$. Writing this out

$$\hat{d}_0 + \sum_{j=1}^{k} A_j\hat{d}_j = d_0$$

$$B_j\hat{d}_j = d_j \quad \forall_j$$

but for $j \neq i$, $d_j = 0 \Longrightarrow \hat{d}_j = 0$ since B_j is nonsingular. Hence the above equations reduce to

$$\hat{d}_0 + A_i\hat{d}_i = d_0$$

$$B_i\hat{d}_i = d_i \Longrightarrow \hat{d}_i = \hat{B}_i^{-1}d_i$$

and

$$\hat{d}_0 = d_0 - A_i\hat{d}_i = d_0 - A_iB_i^{-1}d_i$$

and

$$\hat{d} = \hat{B}_i^{-1}d \quad .$$

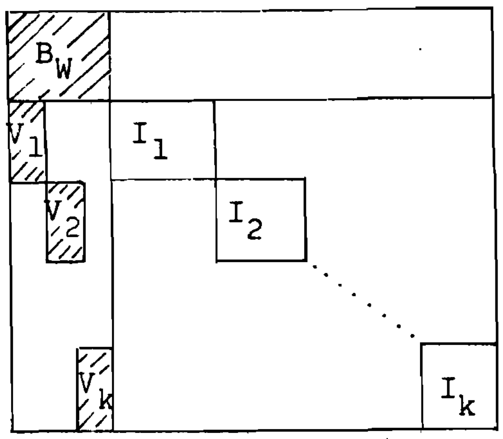
In particular if d belongs to block 0, then $d_i = 0$ and by the above relations $\hat{d}_i = 0$ and $\hat{d}_0 = d_0$ making $\hat{d} = d$ in this case. ||

We can now define $B_A = B_N^{-1}B_T$ and express the basis in factorized form as

$$B_T = B_NB_A \quad .$$

Proposition 4: B_A has the following structure:

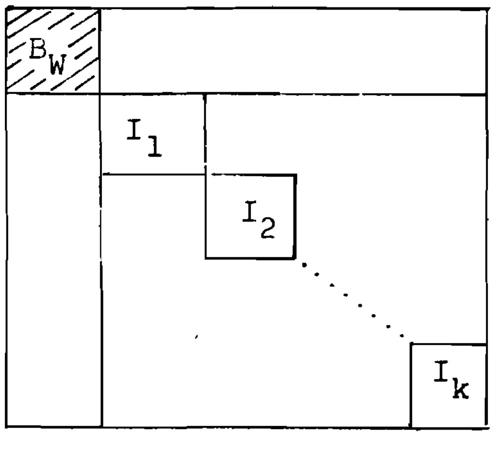
B_A =



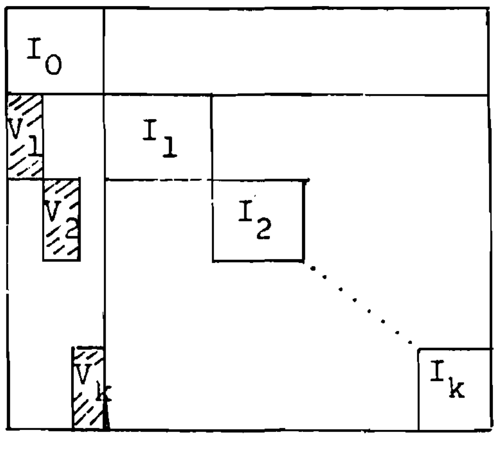
where B_W is $m_0 \times m_0$ and nonsingular and $V_i = B_i^{-1}C_i$.

Proof: The columns of B_A are the columns of B_T updated by B_N^{-1}, hence we obtain a unit vector in the positions where a column is both in B_T and B_N. On the others, i.e. the first m_0 columns, we get by proposition 3 the above structure with $B_i^{-1}C_i = V_i$. Further we can factorize B_A as

$B_A = \hat{B}_W\hat{V}$ =



*



and hence $B_T = B_N \hat{B}_W \hat{V}$ and $\det B_T = \det B_N \cdot \det \hat{B}_W \cdot \det \hat{V}$ and since $\det B_T \neq 0$, $\det B_N \neq 0$ and $\det \hat{V} = 1$, this implies $\det B_W = \det \hat{B}_W \neq 0$ and hence B_W is nonsingular.

As a result of the above we have that we can express the inverse of the basis in factorized form as

$$B_T^{-1} = \hat{V}^{-1} \hat{B}_W^{-1} B_N^{-1} = \hat{V}^{-1} \hat{B}_W^{-1} \prod_{i=1}^{k} \hat{B}_i^{-1} \quad .$$

Updating the Representation of the Inverse

Whenever we replace one column in the basis by another, we encounter three mutually exclusive possible updating situations:

1) The outgoing variable is in the working basis (WB) B_W.
2) The outgoing variable is in some block, say i, and v_r, the row of V_i corresponding to it is zero.
3) The outgoing variable is in some block, say i, and $v_r \neq 0$.

We proceed in each case in the following way:

Case 1: Here we replace one of the columns of

$$\begin{pmatrix} B_W \\ V \end{pmatrix}$$

by a new one, which has the form $\hat{d} = B_N^{-1} d = \hat{B}_i^{-1} d$ for some i. Hence we have to replace one column of B_W and update its inverse in the usual way, and replace the corresponding column in V. Notice that all the information needed had to be calculated for updating the incoming column and hence no calculations are necessary.

Case 2: In this case we replace the outgoing variable directly in its block inverse by the incoming variable. Notice that both have to belong to the same block, since otherwise we would have a pivot element equal to zero, since $\bar{d}_r = -v_r\bar{d}_0 = 0$ for this case. Then if E_N is the elementary matrix to update B_N^{-1}, we would have

$$\begin{pmatrix} B_W \\ V \end{pmatrix}^{new} = E_N \begin{pmatrix} B_W \\ V \end{pmatrix} = \begin{pmatrix} B_W \\ V \end{pmatrix}$$

since the pivot row $v_r = 0$. Hence B_W^{-1},V are unchanged. Only one block inverse is changed by one column.

Case 3: Since $v_r \neq 0$ we can pick a $v_{r_j} \neq 0$. Then the column corresponding to v_{r_j} can be exchanged with the outgoing variable since it has a nonnegative pivot element, giving a new B_N which differs from the old one by one column. This exchange will change both B_W and V. To find out how B_W^{-1} is affected let E be the simple permutation matrix ($E = E^{-1}$) such that $B_T^* = B_T E$, where the index * is used for the new representation. Then we have

$$\hat{B}_T^{*-1} = \hat{V}^{*-1}\hat{B}_W^{*-1}B_N^{*-1} = EB_T^{-1} = E\hat{V}^{-1}\hat{B}_W^{-1}B_N^{-1}$$

$$\hat{B}_W^{*-1} = \hat{V}^* E V^{-1}\hat{B}_W^{-1}B_N^{-1}B_N^*$$

then if $B_N^* = E_N^{-1}B_N^{-1}$ we have $E_N = B_N^{-1}B_N^*$ and replacing this and writing the above expression in partitioned form

$$\begin{pmatrix} B_W^{*-1} & \\ & I_S \end{pmatrix} = \begin{pmatrix} I_0 & \\ V^* & I_S \end{pmatrix}\begin{pmatrix} E_1 & E_2 \\ E_3 & E_4 \end{pmatrix}\begin{pmatrix} I_0 & \\ -V & I_S \end{pmatrix}\begin{pmatrix} B_W^{-1} & \\ & I_S \end{pmatrix}\begin{pmatrix} E_N^1 & E_N^2 \\ E_N^3 & E_N^4 \end{pmatrix}$$

since E_N is an elementary column matrix with pivot row in some block, $E_N^1 = I_0$ and $E_N^3 = 0$. Hence

$$B_W^{*-1} = (I_0, 0)\begin{pmatrix} E_1 & E_2 \\ E_3 & E_4 \end{pmatrix}\begin{pmatrix} I_0 & \\ -V & I_S \end{pmatrix}\begin{pmatrix} B_W^{-1} & \\ & I_S \end{pmatrix}\begin{pmatrix} I \\ 0 \end{pmatrix}$$

$$= (E_1, E_2)\begin{pmatrix} I_0 & \\ -V & I_S \end{pmatrix}\begin{pmatrix} B_W^{-1} \\ 0 \end{pmatrix} = (E_1, E_2)\begin{pmatrix} B_W^{-1} \\ -VB_W^{-1} \end{pmatrix}$$

$$B_W^{*-1} = (E_1 - E_2V)\, B_W^{-1} \quad .$$

But since E is a permutation matrix for columns j and r, E_1 is an identity except for the j-th column which is 0, and E_2 is 0, except for its component corresponding to the j-th row and column r, which is 1. Hence

$$(E_1 - E_2V) = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \boxed{\quad -v_r \quad} & & \\ & & & 1 & \\ & & & & \ddots \quad 1 \end{bmatrix} = E_R \quad ,$$

an elementary row matrix which has $-v_r$ on its j-th row. Now we have to update V. Notice that only V_i needs to be updated, and only for columns j for which $v_{r_j} \neq 0$. The update of the W B corresponds to premultiplying by an elementary column matrix. After this we are back to Case 1.

In conclusion:

<u>Case 1</u>: Replace one column in working basis by another and the same in V. No calculations.

<u>Case 2</u>: Replace one column by another in some block inverse. No calculations.

<u>Case 3</u>: Exchange outgoing variable to working basis. Add row eta to representation of working basis inverse. Update columns of V_i for which $v_{r_j} \neq 0$. Now proceed as in Case 1.

<u>Using the G-GUB Basis Factorization in the Simplex Method</u>

In a simplex method a representation of the inverse is needed for calculating:

1) $\Pi B_T = c$, i.e. $\Pi = c B_T^{-1}$ (BTRAN)

2) $B_T d = \hat{d}$, i.e. $\hat{d} = B_T^{-1} d$ (FTRAN) .

We now explore the implications of the G-GUB basis factorization on this operation. For the following we will use

$$\tilde{V}_i = \begin{array}{|c|c|c|}\hline 0 & V_i & 0 \\ \hline \end{array} \qquad m_i \times m_0$$

1) BTRAN

Writing $\Pi = cB_T^{-1} = c\hat{V}^{-1}\hat{B}_W^{-1}B_N^{-1}$ in expanded form

$$(\Pi_0,\Pi_1,\dots,\Pi_k) = (c_0,c_1,\dots,c_k)\begin{pmatrix} I_0 & & & \\ -\tilde{V}_1 & I_1 & & \\ \vdots & & \ddots & \\ -\tilde{V}_k & & & I_k \end{pmatrix}\begin{pmatrix} B_W^{-1} & & & \\ & I_1 & & \\ & & \ddots & \\ & & & I_k \end{pmatrix}\cdot B_N^{-1}$$

$$(\Pi_0,\Pi_1,\dots,\Pi_k) = ([c_0 - \sum_{i=1}^{k} c_i\tilde{V}_i]B_W^{-1},c_1,\dots,c_k)\ B_N^{-1}$$

$$\Longrightarrow (\Pi_0,\Pi_1,\dots,\Pi_k)\ B_N = ([c_0 - \sum_{i=1}^{k} c_i\tilde{V}_i]B_W^{-1},c_1,\dots,c_k)$$

hence, because of the structure of B_N, we have that

$$\Pi_0 = (c_0 - \sum_{i=1}^{k} c_i\tilde{V}_i)\ B_W^{-1}$$

and

$$\Pi_0 A_i + \Pi_i B_i = c_i \quad , \qquad \text{for } i = 1,\dots,k$$

or

$$\Pi_i = (c_i - \Pi_0 A_i)\ B_i^{-1}$$

or

$$(\Pi_0,0,\dots,0,\Pi_i,0,\dots,0) = (\Pi_0,\dots,0,c_i,0,\dots,0)\ \hat{B}_i^{-1} \quad .$$

From these formulas we can obtain some useful information both for Primal and for Dual strategies.

A) <u>Primal Strategies</u>

A1) Whenever $c_i = 0$, $i = 1,\ldots,k$, i.e. when all sub-blocks are feasible, $\tilde{V}^{-1}$ is not used in BTRAN.

A2) In addition, if we use partial pricing to coincide with some block (or blocks), we need only to calculate the Π_i's for that block (or blocks).

Hence, if we first make all blocks feasible and then use partial pricing, we need only B_W^{-1} and one $\hat{B}_i^{-1}$ in BTRAN, with considerable potential savings in computations and Input-Output.

B) <u>Dual Strategies</u>

Here $c = U_R$ the r-th unit vector. If $c = U_R = (U_{R_0}, U_{R_1}, \ldots, U_{R_k})$ Then according to which of the three update situations we are (recall that they only depend on the position of the out-going variable) we have:

<u>Case 1</u>: $\Pi_0 = U_{R_0} B_W^{-1}$, all other Π_i's as before and we have to calculate them all.

<u>Case 2</u>: $U_{R_i} \neq 0$, $U_{R_j} = 0$ $\forall_j = i$, $v_r = 0$. Then

$$U_{R_0} - \sum_{j=1}^{k} U_{R_j} \tilde{V}_j = 0 \Longrightarrow \Pi_0 = 0 \Longrightarrow \Pi_j = 0 \quad \forall j \neq i.$$

Hence only have to calculate Π_i.

<u>Case 3</u>: As (2) with $v_r \neq 0$. Then $\Pi_0 = -v_r B_W^{-1}$. All other Π_i's as before and we have to calculate them all.

Hence, for Dual strategies we can only achieve savings in BTRAN if we are lucky enough to fall in update situation 2.

2) FTRAN

Here we have

$$\hat{d} = B_T^{-1}d = \hat{V}^{-1}\hat{B}_W^{-1}B_N^{-1}d$$

$$= \hat{V}^{-1}\hat{B}_W^{-1}\hat{B}_i^{-1}d$$

if d belongs to block i ($\hat{d} = \hat{V}^{-1}\hat{B}_W^{-1}d$ for $i = 0$), by proposition 3. That is, we need only one of the block inverses, the working basis inverse, and the V_i matrices. For some Dual methods it is not necessary to completely update the incoming vector, and then it is not necessary to use the V_i matrices. On the other hand it becomes necessary to use them when determining if the subproblem variables satisfy the nonnegativity constraints so that this benefit is partially offset.

Considerations in Selecting A Simplex Strategy

All Dual (and parametric) strategies for blockangular linear problems with coupling constraints go out from the fact that if all subproblems are solved first, then the resulting solution is dual feasible for the overall problem. Hence the first step is to solve the subproblems up to optimality. On the other hand, in order to make full use of the reductions in computations and data transfer during BTRAN for Primal strategies, it is necessary to have all subproblems feasible.

Alternatively we could also in this case solve them up to optimality, on the heuristics that this way we would later approach feasibility in the common rows "from above" (maximizing case) and hence could expect to hit the feasible region at a higher value of the objective function, and have fewer iterations in Phase II. Thus we would have a Unified Modified Phase I Procedure for both Primal and Dual strategies (See Orchard-Hays [9]).

Primal strategies would have the following advantages in this phase:

a) Since they do not require a dual feasible overall solution, we could stop before reaching optimality, saving iterations.

b) For the same reason, they do not require any special tricks if the subproblems are unbounded or if $H_0 \neq 0$.

After this phase, Primal strategies would have the advantage of a much reduced BTRAN if we make use of partial pricing. Since for large problems partial pricing is desirable to reduce overall computation time, the added benefits of reducing the amount of data and of computations required in the backward transformation make it even more attractive here.

Dual strategies would present a slight advantage in the Forward transformation, but nothing so dramatic as to offset the reduction in BTRAN for Primal strategies, especially so if we consider that usually time spent in BTRAN is much larger than that in FTRAN (See de Buchet [2]). The only case where there is a reduction in BTRAN for Dual strategies is in the case of a type 2 basis update. But in this case B_W does not change, and hence if we use the same c vector (true in Phase II or in Phase I if we interpret our objective as the minimization of the sum of artificials) then Π_0 will not change on this iteration for a Primal strategy and we need not recompute it, with still further savings on BTRAN for the next iteration.

All the above considerations make an algorithm using the Unified Modified Phase I Procedure, followed by a Primal strategy using partial pricing, look very attractive. If the number of iterations after the Modified Phase I Procedure is not vastly larger than for a Dual method, it should perform better. By not requiring us to solve problems up to optimality we have an added degree of freedom.

In the following figures 1-a) and 1-b) we present a basic flow-sheet of the resulting algorithm.

Other Considerations

Notice that the method does not depend on any particular representation for the inverses of the basis involved. Hence we can always use the most appropriate. LU factorization followed by the Forrest and Tomlin updating procedure [3] has proven highly successful. It can be used directly for each block inverse. The author has worked out an adaptation for this procedure for the working basis, which will be reported elsewhere.

Figure 1-a

G-GUB Modified Phase I

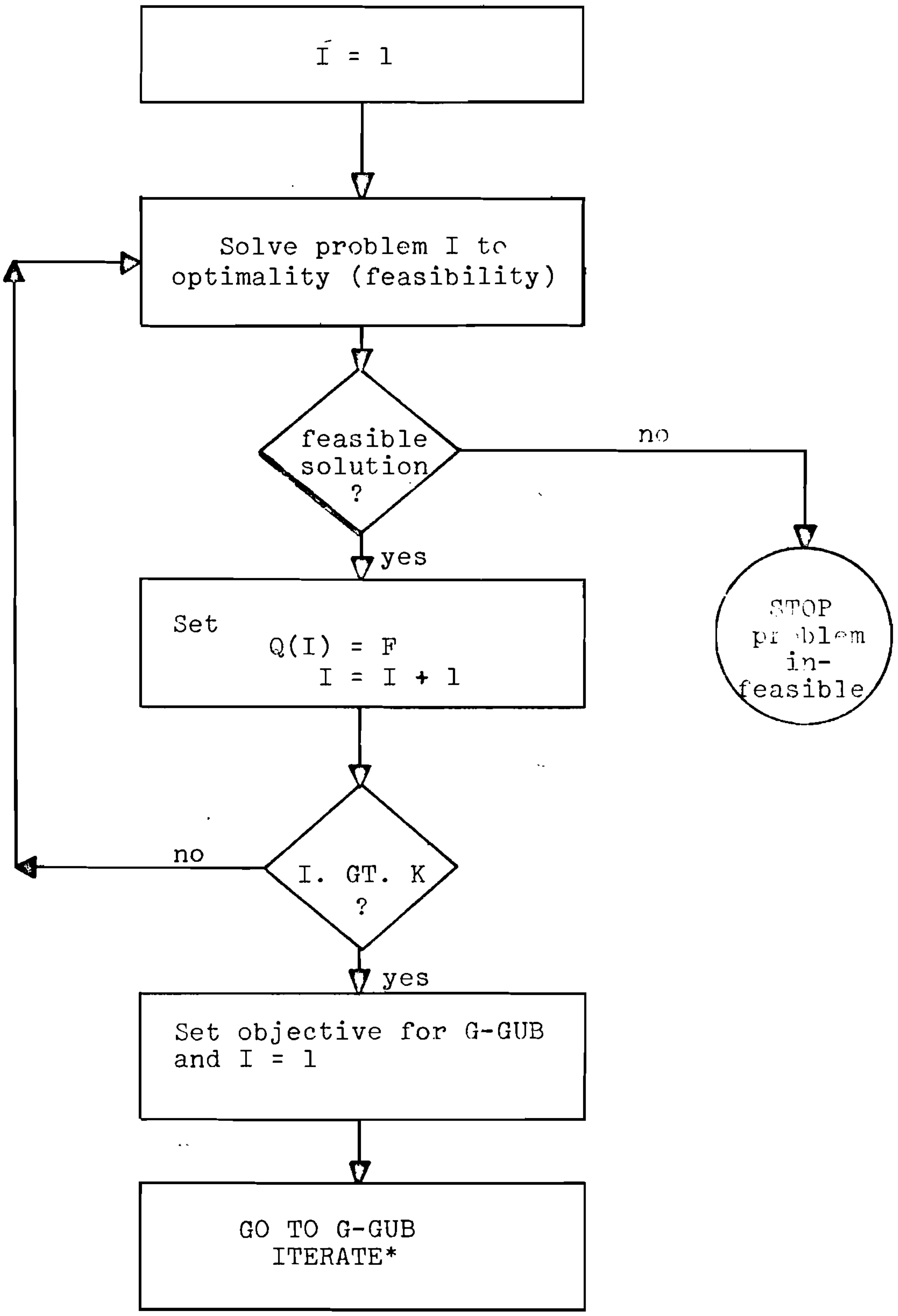
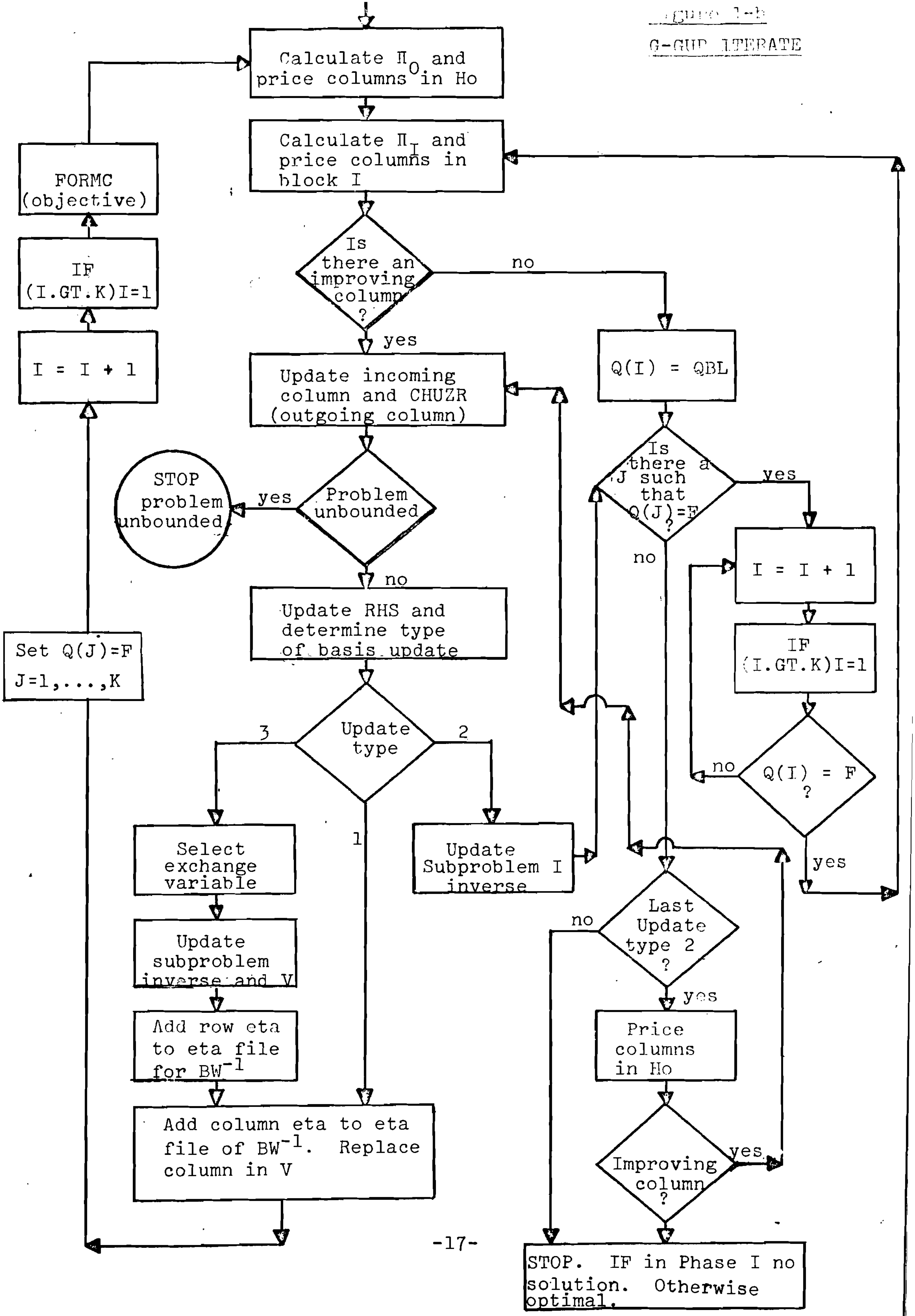


*Figure 1-b

Figure 1-b

G-GUP ITERATE



Calculate Π_0 and price columns in Ho
Calculate Π_I and price columns in block I
Is there an improving column ?
no
yes
Update incoming column and CHUZR (outgoing column)
Problem unbounded
yes
STOP problem unbounded
no
Update RHS and determine type of basis update
Update type
3
2
1
Select exchange variable
Update subproblem inverse and V
Add row eta to eta file for BW^{-1}
Add column eta to eta file of BW^{-1}. Replace column in V
Update Subproblem I inverse
Q(I) = QBL
Is there a J such that Q(J)=F ?
yes
no
I = I + 1
IF (I.GT.K)I=1
Q(I) = F ?
no
yes
Last Update type 2 ?
no
yes
Price columns in Ho
Improving column ?
yes
STOP. IF in Phase I no solution. Otherwise optimal.
FORMC (objective)
IF (I.GT.K)I=1
I = I + 1
Set Q(J)=F J=1,...,K

Preliminary Experimental Results

FORTRAN Code

An all in core FORTRAN LP code, LPM1, written by J.A. Tomlin in Stanford, was adapted to the G-GUB algorithm.

LPM1 stores the matrix packed by columns in a vector of nonzeroes, a vector of row indices and a vector giving for each column a pointer indicating the position of its first nonzero entry in the previous vectors. Similarly for the eta file. Invert uses an L-U factorization, with product form updates.

G-GUB was conceived as an out of core code, where at each time the data for one block, matrix and eta file, is held in core in the form required by LPM1, while in the meantime the data for all other blocks is kept on disk. The working basis inverse, H_0 and V are stored in the same way as a block 0. Whenever we need the data for another block, the one in core is written out to disk (unless it has not changed) and the new one is read in. Due to the packing scheme used by LPM1, which was designed as an in core code, the I-O operations for G-GUB are somehow inefficient. This was thought to be not too serious for an experimental code like G-GUB, since the time spent on I-O could be measured, which allows us to make comparisons on computation times alone, or to have an estimate of the effect of these inefficiencies. Besides, it was felt that the advantages of adapting and existing LP code instead of writing a new one outweighted these other considerations.

Other computational characteristics of G-GUB are:

1) Block inverses were inverted whenever N1 new etas had been added to it since its last inversion. The same for the working basis. (For tests N1 = 30.)

2) Every N2 iterations the solution was recomputed by solving $X_B = \hat{V}^{-1}\hat{B}_W^{-1}B_N^{-1}b$. A first step for this is to calculate for each i = 1,...,k, $\beta_i = B_i^{-1}b_i$. When doing this the accuracy of the computed β_i was checked. If the maximum row error exceeded a tolerance, the corresponding block basis was reinverted even though it was not necessary by the criteria in (1). At the same time the corresponding $V_i = B_i^{-1}C_i$ was recomputed using the new representation for the inverse. After this step the working basis was reinverted with its recomputed columns. The accuracy of the thus recomputed X_B was always found to be good.

3) All computations were performed on an IBM 360/91 at the Stanford Linear Acceleration Center (SLAC). The computing times reported are CPU seconds.

Description of Problems

Three problems were available. They are described in Table 1.

Table 1. Description of Problems

Problem	FIXMAR	FORESTRY	DINAMICO
Total number of rows	325	404	417
Total number of columns	452	603	527
Total density	1.8%	1.6%	1.8%
Number of blocks	4	6	3
Common rows	18	11	56
Block 1 rows, column, density	92/114/6%	73/103/6.1%	117/177/3.9%
Block 2	73/94/5.4%	47/71/12.3%	108/164/4.3%
Block 3	57/125/3.5%	69/109/8.9%	136/192/4.5%
Block 4	85/118/8.7%	72/134/5.7%	
Block 5		63/89/12.1%	
Block 6		69/97/7.4%	

First Runs

The first experiments, with an early version of the code, were done to compare solution times of G-GUB and MPS-360. The results with the two problems available at that time are given in Table 2.

Table 2. Solution times using G-GUB and MPS-360

Code / Problem	G-GUB making first blocks feasible	G-GUB making first blocks optimal	MPS-360
FIXMAR	22	21	36
DINAMICO	126	113	112

<u>Second Runs</u>

The above times for G-GUB were considered encouraging. It was felt that for later tests LPM1 should be used as the standard LP since then the times would not be affected by differences in the codes and would be directly comparable. Besides, if G-GUB performed better, it was important to determine to what degree this was due to the Modified Phase I Procedure, to the G-GUB basis factorization or to the partial pricing strategy used in conjunction with this latter one.

Therefore some slight modifications were introduced to the code, which allowed us to test different options. These options were

A) <u>Basis representation</u> using G-GUB factorization or the standard LPM1 LU factorization with product form updates for the basis of the whole problem.

B) <u>Modified Phase I</u>: Here we considered three options (1) solving blocks to optimality, (2) making blocks feasible or (3) use standard Phase I without treating blocks first.

C) Pricing: (1) Partial pricing by block or (2) total pricing at each iteration.

By a combination of these options the following strategies could be tested:

Strategy	Basis Representation	Modified Phase I	Pricing
-1	G-GUB	feasibility	block
0	G-GUB	optimality	block
1	G-GUB	optimality	total
2	G-GUB	no	block
3	G-GUB	no	total
5	LPM1	no	total
5a	LPM1	no	block

Using problem FIXMAR these strategies were compared. The results are given in Table 3.

Table 3. Comparison of strategies on problem FIXMAR

Strategy	-1	0	1	2	3	5	5a
Total CPU sec	22.12	21.04	25.45	38.52	38.70	35.02	47.65
Core used	156k	156k	156k	156k	156k	200k	200k
I/O CPU sec	4.30	2.98	4.35	10.99	10.53	--	--
Total-I/O CPU sec	17.82	18.06	21.10	27.53	28.17	35.02	47.65
Comp. time G-GUB iteration	3.35	3.46	4.42	3.40	4.41	--	--
Comp. time LMP1 iteration	--	--	--	--	--	5.15	5.30

Third Run

By this time strategies 0 and 5 were compared on problem FORESTRY. The results are given in Table 4.

Table 4. Comparison of strategies 0 and 5 on problem FORESTRY

Strategy	0	5
Total CPU sec	60.52	91.22
Core used	158k	270k
I/O CPU sec	11.82	--
Total - I/O CPU sec	48.70	91.22
Comp. time/ G-GUB iter.	9.30	--
Comp. time/ LMP1 iter.	--	10.1

Analysis of Results

1) The G-GUB algorithm can produce substantial reductions in overall computation time for blockangular linear problems with coupling constraints, with respect to comparable general LP's, as can be seen by comparing the total solution times for problems FIXMAR and FORESTRY using strategies 0 (G-GUB algorithm) and 5 (general LP).

2) If FIXMAR is any indication then each one of the three options in the G-GUB algorithm helps in reducing the overall solution time. The best results are obtained when all three are in effect, where in this case we get a reduction by approximately a factor of 2.

3) Notice that strategies 1 and 3 differ only in that 3 makes use of the Modified Phase I Procedure and this gives about a 25% reduction in computation times. This would mean, if it were true in general for blockangular problems with coupling constraints, that general LP's could be made more efficient for this type of problem by using this strategy.

4) The mean time per iteration was noticed to increase with the number of block vectors in the working basis. This relationship is plotted on Fig. 2 for DINAMICO, for which the effect is more pronounced due to the large number of common rows. Notice that the mean time per iteration with 45 block vectors in the working basis is about three times that with 0.

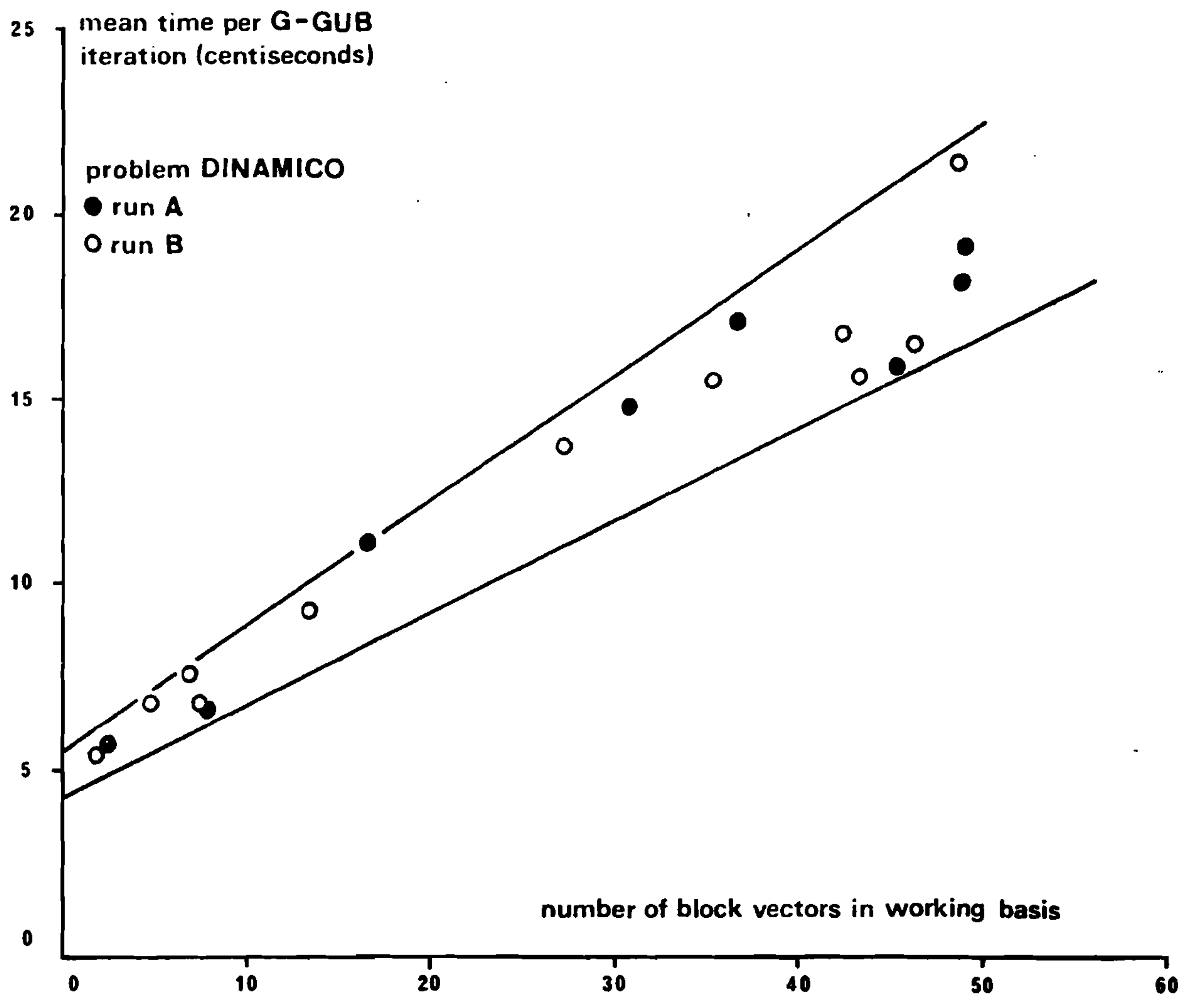
This effect is due mainly to longer transformation times, especially in FTRAN, as the number of non zeroes in the WB and in the V matrices increases linearly with the number of block vectors in the working basis, and to an increase in the frequency of the more expensive type 3 updates. (The more block vectors in BW the smaller the probability of $V_r = 0$).

This suggests a strategy modification to reduce the mean time per iteration. At the end of Phase I, all block variables in the working basis are treated as parameters fixed at their current value. Thus there are no block vectors in the working basis and V = 0 and we get faster iterations because of the reduced transformation time. When the number of block vectors in the working basis has again increased to a level similar to that at the end of Phase I, the variables treated as parameters are considered as candidates and their values modified in the direction to improve the objective function until they reach their bounds or displace a basic variable.

Remarks

I wish to thank J.A. Tomlin for his comments and advice and G.B. Dantzig for encouraging this work at the Systems Optimization Laboratory at Stanford.

FIGURE 2: MEAN TIME PER G-GUB ITERATION VS. NUMBER OF BLOCK VECTORS IN WORKING BASIS.



REFERENCES

[1] Bennett, J.M. An Approach to Some Structured Linear Programming Problems, Operations Research, 14, 1. 1966.

[2] de Buchet, J. How to Take into Account the Low Density of Matrices to Design a Mathematical Programming Package. In Large Sparse Sets of Linear Equations. (J.K. Reid, ed.), Academic Press, 1971.

[3] Forrest, J.J.H. and Tomlin, J.A. Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method, Math. Prog 2, 1972.

[4] Grigoriadis, M.D. Unified Pivoting Procedures for Large Structured Linear Systems and Programs. In Decomposition of Large-Scale Problems, (D.M. Himmelblau, ed.), North-Holland, 1973.

[5] Kaul, R.N. An Extension of Generalized Upper Bounding Techniques for Linear Programming, ORC-62-27, Operations Research Center, University of California, Berkeley, 1965.

[6] Lasdon, L.S. Optimization Theory for Large Systems, Macmillan, 1970.

[7] Müller-Mehrbach, H. Das Verfahren der Directen Dekomposition in der Linearen Planungsrechnung. Ablauf und Planungs-Forschung, 6. 1965.

[8] Ohse, D. A Dual Decomposition Method for Block Diagonal Linear Programs, Zeitschrift für Operations Research, 17. 1973.

[9] Orchard-Hays, W. Practical Problems in LP Decomposition. In Decomposition of Large-Scale Problems, (D.M. Himmelblau, ed.), North-Holland, 1973.

[10] Rosen, J.B. Primal Partition Programming for Block Diagonal Matrices, Numerische Mathematik, 6, 1964.