Working Paper

Pulsar Algorithms: A Class of Coarse-Grain Parallel Nonlinear Optimization Algorithms

Jerzy Sobczyk, Andrzej P. Wierzbicki

WP-94-53
August 1994

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 71521 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

Pulsar Algorithms: A Class of Coarse-Grain Parallel Nonlinear Optimization Algorithms

Jerzy Sobczyk, Andrzej P. Wierzbicki

WP-94-53
August 1994

International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 71521 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

Foreword

Computations that are performed either on parallel processors or distributed on a network of computers are becoming standard options technologically, while we are not yet fully prepared to take appropriate advantage of the available parallel or distributed computing power. This stimulates much research on parallel computation, usually motivated by using the parallel computing power to solve bigger and more complicated problems in shorter time.

The authors of this paper address a different objective: how to use the parallel computing power for obtaining more robust solutions to problems that are not necessarily bigger, but might be difficult to solve because of other reasons such as nonlinearity, bad conditioning, etc.

For this purpose, a new class of pulsating or "pulsar" algorithms has been developed by the authors of this paper who worked on these algorithms and prepared this paper in the cooperation with the Methodology of Decision Analysis Project as a part of the IIASA Contracted Study Agreement project activities on *Methodology and Techniques of Decision Analysis.* These results were presented also during a workshop on *Decomposition and Parallel Computing Techniques for Large Scale Systems* organized by the Optimization under Uncertainty Project. Thus, this paper represents results of a cooperation which has not only international, but also inter-project character.

Abstract

Parallel architectures of modern computers formed of processors with high computing power motivate the search for new approaches to basic computational algorithms. Another motivating force for parallelization of algorithms has been the need to solve very large scale or complex problems. However, the complexity of a mathematical programming problem is not necessarily due to its scale or dimension; thus, we should search also for new parallel computation approaches to problems that might have a moderate size but are difficult for other reasons. One of such approaches might be coarse-grained parallelization based on a parametric imbedding of an algorithm and on an allocation of resulting algorithmic phases and variants to many processors with suitable coordination of data obtained that way. Each processor performs then a phase of the algorithm - a substantial computational task which mitigates the problems related to data transmission and coordination. The paper presents a class of such coarse-grained parallel algorithms for unconstrained nonlinear optimization, called pulsar algorithms since the approximations of an optimal solution alternatively increase and reduce their spread in subsequent iterations. The main algorithmic phase of an algorithm of this class might be either a directional search or a restricted step determination in a trust region method. This class is exemplified by a modified, parallel Newton-type algorithm and a parallel rank-one variable metric algorithm. In the latter case, a consistent approximation of the inverse of the hessian matrix based on parallel produced data is available at each iteration, while the known deficiencies of a rank-one variable metric are suppressed by a parallel implementation. Additionally, pulsar algorithms might use a parametric imbedding into a family of regularized problems in order to counteract possible effects of ill-conditioning. Such parallel algorithms result not only in an increased speed of solving a problem but also in an increased robustness with respect to various sources of complexity of the problem. Necessary theoretical foundations, outlines of various variants of parallel algorithms and the results of preliminary tests are presented.

Contents

Pulsar Algorithms: A Class of Coarse-Grain Parallel Nonlinear Optimization Algorithms *

Jerzy Sobczyk,**Andrzej P. Wierzbicki***

1 Parametric Imbedding Approach to Parallel Optimization Algorithms

While there exist diversified approaches to parallel computations in optimization problems, see e.g. [Bertsekas et al. (1989)], [Grauer et al. (1991)], a new parametric imbedding approach to the parallelization of optimization algorithms for linear programming problems has been proposed in [Wierzbicki (1993)]. This approach does not aim at solving problems of larger scale, but at using the parallel computing power in order to more reliably and faster solve problems of moderate scale that might be difficult e.g. because of their ill-conditioning. This approach might be also extended to any other optimization algorithms, in particular, to nonlinear programming problems; such extension is the subject of this paper. The essential aspects of this parallelization approach are as follows:

1. An optimization problem and/or an optimization algorithm is *parametric imbedded* into a family of problems or algorithms (in [Wierzbicki (1993)] an imbedding into a family of multi-objective linear programming problems has been used, but any other imbedding might be also applied).

2. The optimization algorithm (not necessarily the optimization problem - see also [Ruszczyński (1989)]) is then *decomposed into algorithmic phases or tasks* that might be executed parallel for various parameter values.

3. The data obtained at the end of an algorithmic phase or task with various parameter values is used to *coordinate, accelerate and make more robust the process of finding the solution* of the original optimization problem.

Clearly, these aspects of the parallelization approach are very general and the main difficulty relates to a skillful choice of their particular features. Some of such features are as follows:

A. The resulting parallelization should be *coarse-grained*: each parallel working processor executes an algorithmic phase which is a considerable computational task. Many known approaches to parallel computations concentrate on fine-grain parallelization, e.g.

*The research reported in this paper was partly supported by the grant No. 3 0218 91 01 of the Committee for Scientific Research of Poland and partly by the grant under The Program for Research Collaboration between Norway and Eastern Europe Nations 1993 from the Norwegian Research Council.

**Institute of Automatic Control, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland.

on a best way of parallel execution of vector and matrix operations - see e.g. [Nogi (1986)]. Even in this case, it is known that the parallelization should not be too fine-grained in order e.g. not to loose the pipelining efficiency in vector computations, see e.g. [Carey (1989)]. Until recently, when the available processors had rather limited computational power and the research on parallel computing assumed *massively parallel computer architectures* (i.e. thousands of processors with small computing power), the concentration on rather fine-grained parallelization was well substantiated. However, modern processors have today computing power comparable to supercomputers from yesterday and the corresponding computer architectures are rather *scalably parallel* (i.e. consist of a variable number of very powerful processors). Therefore, it seems reasonable to concentrate on coarse-grain parallelization.

The coarse-grain parallelization justifies another focus on the organization of parallel computations: while the problems of *data transmission* might be negligible, the issues of *coordination of computations* might be even more challenging than in fine-grain parallelization. In particular, parallel computing tasks might be asynchronous, adaptively dynamic coordinated i.e. started and finished depending on the data obtained from all parallel tasks.

B. While the need of solving problems of larger scale is a legitimate motivation of parallelizing computations, it is not the only one possible. If the dimension of the original problem is moderate but this problem is difficult to solve because of other reasons, then we might try to *solve the problem parallel for many parametric instances*: due to a parametric imbedding, various problems similar to the original one might be solved together. This clearly increases the total computational effort and, at the same time, improves the robustness of the solution in respect to parametric variations, computational inaccuracies and bad numerical conditioning of the problem; such an improvement is well-known and typical for parametric imbedding approaches. The essential feature and difficulty of the parallelization approach is, however, to devise such decomposition and coordination of the algorithm that *only the algorithmic tasks, not the entire computations are executed for many instances of the problem*, while the choice of decomposition and coordination scheme results in a trade-off between the robustness of the solution and the acceleration of computations.

Suppose P parallel working processors are used: if we just generate the same number P of parametric instances of the problem to be solved and solve them without decomposition on each of the processors, then we only increase the robustness of the solution at the cost of P-times multiplication of the total computational effort. If we, however, decompose the solution algorithm into algorithmic tasks and evaluate the data obtained after each task is executed, we can use the data for two purposes: *to increase robustness and to shorten computational time.* The total computational effort will be greater than on a single processor, but *we have anyway P times increased, parallel computing power and the question is how to reasonably use it.* If the purposes of increasing robustness and shortening computational time are deemed equally important, we might postulate that the *computational time should be shortened $\sqrt{P}$ times while another $\sqrt{P}$ times of computational power increase should be used for improving the robustness of the solution.* In other words, we should be satisfied with a speedup of $\sqrt{P}$ and the so-called efficiency of $\frac{1}{\sqrt{P}}$ as long as the remaining computational power is actually not lost but utilized to increase the robustness of the parallel algorithm.

However, this '$\sqrt{P}$ postulate' is only an approximate goal. Generally, the number of parametric problem instances or tasks might be different, say M, than the number of available processors, while we can often assume $M \geq P$ for a scalably parallel architec-

ture. In order to balance processors loads, we can use the same processor several times for various parametric instances or tasks, evaluate the data obtained after each task is executed and dynamic adaptively adjust the number of necessary instances M depending on the specifics of problem solved and on the results obtained.

These general features of the parallelization approach might be further specified for the case of optimization algorithms. The number M of parametric instances or tasks can be then related to the dimension n (and/or to the number m of constraints) in the optimization problem. In some of such problems, in particular - large-scale linear optimization problems, the numbers n and m might reach hundreds of thousands; the issue is then how to reasonably limit the number of algorithmic phases and processors used. We concentrate here, however, on optimization problems of moderate size where the difficulty of solution is related to other features than the dimension; in particular, nonlinear optimization problems might be difficult even if their dimension does not exceed several hundreds.

Such optimization problems of moderate size but with strong nonlinearities and complex structure have numerous applications in science and engineering; they are frequently encountered in e.g. in control engineering. In this paper, we concentrate on the question how to use the general parallelization approach outlined above for such type of nonlinear optimization problems. Moreover, we start with the basic unconstrained optimization algorithms for differentiable functions, since they are used also as essential parts in more complicated cases of problems with constraints and/or with nondifferentiabilities. In sequential computations, three classes of unconstrained differentiable optimization algorithms are considered basic:

a) Several variants of *conjugate directions algorithms.* Conjugate direction algorithms are often used for solving large systems of linear equations; in this application, they were fine-grain parallelized in various ways, see e.g. [Bertsekas et al. (1989)]. For a fully parallel, coarse- grained version of such algorithms we would need a method of producing a set of many conjugate directions in one parallel sweep; although there are some recent attempts to develop such a method, see e.g. [Chronopulos et al. (1989)], these attempts relate again to linear equation systems or quadratic optimization problems. The question, whether conjugate direction algorithms can be essentially, coarse-grain parallelized in a general case of nonlinear optimization or whether they are sequential in their nature, seems to be open yet. We shall not address conjugate direction algorithms in this paper, although many of the ideas presented here are applicable also for them.

b) Various variants of *variable metric (quasi-Newton) algorithms* that use approximations of the (inverse of) hessian matrix of second-order derivatives based on data concerning gradient increments. The most popular variants of such algorithms (BFGS and other rank-two algorithms) are related to the idea of conjugate directions and might present difficulties in consistent parallelization. However, we shall use in this paper the fact (see also Lootsma in [Grauer et.al. 1991]) that another variant, the *rank-one variable metric method can be made consistently parallel* due to its specific properties, while its deficiencies can be overcome in parallel implementation.

c) Various variants of *Newton or Newton-type algorithms* using directly computed second-order derivatives. We should note that a revival of such techniques can be observed recently, most probably because symbolic differentiation helps to prevent errors in programming derivatives (e.g. for a problem with 10 variables, 55 second-order derivatives must be programmed as a part of hessian matrix specification). Provided the appropriate tools of symbolic differentiation will be incorporated in optimization software, see e.g. [Paczyński (1993)], some variants of such algorithms will be extremely useful. We

present here also a regularized and coarse-grain parallelized version of such an algorithm - while fine-grain parallelized versions of such algorithms were known before, see e.g. [Bertsekas et al. (1989)].

The main algorithmic component of an unconstrained nonlinear optimization method is usually directional optimization or *directional search*; the specific algorithms differ then in the way the directions of search are generated. The directional search will constitute here the basic algorithmic task that might be executed parallel. Another family of unconstrained nonlinear optimization methods uses *restricted step or trust region* determination instead of directional search - see e.g. [Fletcher (1987)]. Restricted step determination can be used as an alternative basic algorithmic task of a pulsar algorithm, but we concentrate here on the use of directional searches. Moreover, we assume here that the basic algorithmic tasks - directional searches - are organized in a special *double-phased pulsar (pulsating) parallel optimization algorithm.*

Suppose the parametric imbedding results in generating M various instances of possible search directions; in a simplest case, they might be generated as versor directions, or correspond to various approximations and/or regularizations of the hessian matrix, or to various conjugate directions, or various choices of updates of the basis in linear programming, etc. In an *odd-numbered or divergent iteration* of a pulsar algorithm, directional searches are performed parallel along all these directions and result in *divergent approximations* of the optimal solution of the problem. These approximations can be then evaluated and, starting from each of them, another direction leading to the optimal solution is determined. In an *even-numbered or convergent iteration* of a pulsar algorithm, directional search performed along these new directions will lead to approximations of the solution that are — hopefully, at least in a sufficiently close neighborhood of the optimal solution — convergent, hence are called *convergent approximations.* Far from the optimal solution, even the convergent approximations can differ rather widely; but this spread of approximations helps in increasing the robustness of the algorithm with respect to numerical inaccuracies, ill-conditioning or even local minima. Thus, a full double iteration (composed of one divergent and one convergent iteration) of a pulsar algorithm ends with a choice of one - or several - approximation point from which a new double iteration will start.

This concept of a double-iterative parallel pulsar optimization algorithm is, as indicated, quite general and can be probably applied widely in optimization algorithms of various types - e.g. with constraints, nondifferentiable etc. However, we concentrate here on the differentiable, nonlinear case and the classes **b)** and **c)** of quasi-Newton or Newton-like algorithms with directional searches.

2 Parallel Pulsar Variants of Quasi-Newton and Newton-type Algorithms.

We start here with some basic concepts related to variable metric algorithms. Suppose the nonlinear unconstrained optimization problem consists in minimizing a twice differentiable function $f : \boldsymbol{R}^n \to \boldsymbol{R}^1$. The so-called variable metric is actually a $n \times n$ matrix - denoted further by $\boldsymbol{V}^{(k)}$, where k is an iteration index - that successively approximates the hessian of the minimized function, $\boldsymbol{H}(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$, or - which is assumed here - its inverse $\boldsymbol{H}^{-1}(\boldsymbol{x})$. In a preliminary analysis of such approximation it is usually assumed that the minimized function is quadratic:

$$
\begin{aligned}
f(\boldsymbol{x}) &= 0.5 < \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x} > + < \boldsymbol{b}, \boldsymbol{x} > + c \\
\nabla f(\boldsymbol{x}) &= \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{g} \in \boldsymbol{R}^n \\
\nabla^2 f(\boldsymbol{x}) &= \boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{A} \in L(\boldsymbol{R}^n, \boldsymbol{R}^n)
\end{aligned} \tag{1}
$$

where $< ., . >$ denotes the inner product; thus, the hessian matrix is constant. If an iterative method results in consecutive increments or steps $\boldsymbol{s}^{(k)}$ of the variable $\boldsymbol{x}$ and the corresponding gradient increments are denoted by $\boldsymbol{y}^{(k)}$, then the following relations hold for a quadratic function (with an invertible, say, positively definite hessian):

$$
\begin{aligned}
\boldsymbol{s}^{(k)} &= \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \\
\boldsymbol{y}^{(k)} &= \nabla f(\boldsymbol{x}^{(k+1)}) - \nabla f(\boldsymbol{x}^{(k)}) \\
\boldsymbol{y}^{(k)} &= \boldsymbol{H}\boldsymbol{s}^{(k)} \\
\boldsymbol{s}^{(k)} &= \boldsymbol{H}^{-1}\boldsymbol{y}^{(k)}
\end{aligned} \tag{2}
$$

We could thus construct a matrix $\boldsymbol{Y}$ of gradient increments $\boldsymbol{y}^{(k)}$ that correspond to (n linearly independent) steps $\boldsymbol{s}^{(k)}$ forming a matrix $\boldsymbol{S}$ for $k = 1, ..n$ and compute $\boldsymbol{H}^{-1} = \boldsymbol{S}\boldsymbol{R}^{-1}$. However, it is often more useful to apply an incremental approximation $\boldsymbol{V}^{(k+1)} = \boldsymbol{V}^{(k)} + \Delta\boldsymbol{V}^{(k)}$ of $\boldsymbol{H}^{-1}$ constructed in such a way that the following property holds:

$$
\boldsymbol{s}^{(j)} = \boldsymbol{V}^{(k+1)}\boldsymbol{y}^{(j)} \qquad \text{for} \quad j = 1, ..k \tag{3}
$$

which implies that $\boldsymbol{V}^{(n+1)} = \boldsymbol{H}^{-1}$, if all $\boldsymbol{y}^{(k)}$ are linearly independent.

There are many known formulae for $\Delta\boldsymbol{V}^{(k)}$ that result — under appropriate additional assumptions — in the above property. Most popular are rank-two variable metric formulae where $\Delta\boldsymbol{V}^{(k)}$ is a sum of two rank-one matrices, for example the BFGS formula, or its dual (identical when applied to the matrix $\boldsymbol{B}^{(k)} = (\boldsymbol{V}^{(k)})^{-1}$) DFP formula, see e.g. [Fletcher (1987)], [Gill et al. (1981)]). However, the property (3) holds for rank-two variable metrics under the assumption of *precise directional search in the following quasi-Newton directions*:

$$
\begin{aligned}
\boldsymbol{d}^{(k)} &= -\boldsymbol{V}^{(k)}\boldsymbol{g}^{(k)} \text{ for } k > 1 \\
\boldsymbol{d}^{(1)} &= -\boldsymbol{g}^{(1)} \\
\boldsymbol{g}^{(k)} &= \nabla f(\boldsymbol{x}^{(k)})
\end{aligned} \tag{4}
$$

and is related to the fact that these directions are *conjugate*. On the other hand, [Stachurski (1981)] has shown that the Broyden's class of rank-two variable metric methods - that includes DFP and BFGS formulae as special cases - approximates asymptotically the property (3) for twice differentiable nonlinear functions with uniformly positive definite and bounded hessians and results in superlinear convergence even without precise directional search and conjugacy, *if the quasi-Newton steps* $\boldsymbol{s}^{(k)} = \boldsymbol{d}^{(k)}$ *are consistently used.* Thus, these properties rely on essentially sequential quasi-Newton iterations and a consistent parallelization of rank-two variable metric methods is rather difficult.

There exists, however, a variant of the variable metric method that does not require even that the quasi-Newton directions (4) are used and neither precise directional search nor the conjugacy of directions (4) is actually needed in order to obtain the property

(3) and an inverse hessian approximation. It is the (symmetric) rank-one variable metric defined by formula:

$$\Delta \boldsymbol{V}^{(k)} = \frac{(\boldsymbol{s}^{(k)} - \boldsymbol{V}^{(k)}\boldsymbol{y}^{(k)}) \times (\boldsymbol{s}^{(k)} - \boldsymbol{V}^{(k)}\boldsymbol{y}^{(k)})}{< (\boldsymbol{s}^{(k)} - \boldsymbol{V}^{(k)}\boldsymbol{y}^{(k)}),\ \boldsymbol{y}^{(k)} >} \tag{5}$$

where . $\times$. denotes the outer product. This method is unpopular, because it has an essential drawback — it becomes ill-defined exactly then, when the matrix $\boldsymbol{V}^{(k)}$ approximates $\boldsymbol{H}^{-1}$ well, since the denominator of (5) is then close to zero:

$$< (\boldsymbol{s}^{(k)} - \boldsymbol{V}^{(k)}\boldsymbol{y}^{(k)}),\ \boldsymbol{y}^{(k)} > = < (\boldsymbol{H}^{-1} - \boldsymbol{V}^{(k)})\boldsymbol{y}^{(k)},\ \boldsymbol{y}^{(k)} > \tag{6}$$

It is possible to modify this method with special safeguards against ill-definition that result also in good theoretical properties of the method such as the monotonicity of approximation of $\boldsymbol{H}^{-1}$ and are rather effective in practical applications (see [Kręglewski and Wierzbicki (1981)]), but these possibilities were not widely utilized. However, more recently [Conn et al. (1991)] used similar safeguards to obtain a version of the rank-one variable metric method that is superior to rank-two methods in computing tests and practical applications. Moreover, they have shown that - under rather mild assumptions - the errors of hessian approximation $\|\boldsymbol{H}^{-1}(\hat{\boldsymbol{x}}) - \boldsymbol{V}^{(k)}\|$ have the same order of convergence as the sequence $\|\boldsymbol{x}^{(k)} - \hat{\boldsymbol{x}}\|$, where $\hat{\boldsymbol{x}}$ is the limit point of $\boldsymbol{x}^{(k)}$.

Precisely this modified rank-one variable metric method can be consistently parallelized, which was also printed out by Lootsma in [Grauer et al. 1991]. Since the property (3) does not depend in this method on the accuracy of directional search nor on conjugacy of search directions, hence the data $\boldsymbol{s}^{(k)}$ and $\boldsymbol{y}^{(k)}$ for this method can be gathered from independently, parallel working processors. One of processors should be reserved for computing the approximation (5) with appropriate safeguards - which are easier and safer in the parallel case, since we do not need in this case to use an incomplete approximation of $\boldsymbol{H}^{-1}(\boldsymbol{x})$ and can start a new iteration first when a complete and well-defined approximation is available.

2.1 Basic variant of variable metric pulsar algorithm.

We assume here that the dimension n of the optimization problem is possibly greater than the number of available processors P. In this case, each processor might be used for several algorithmic taksks. We shall index the processors by $\psi = 1,..P$, while $\psi = 1$ will be reserved for variable metric approximation and other coordinating or so-called housekeeping tasks. For counting the main iterations of the algorithm we shall use the index K that will increase by two during each double iteration.

In an odd-numbered (divergent) iteration of the pulsar algorithm, we assign subsequently to each processor with $\psi = 2,..P$ the following directions: first $\boldsymbol{d}^{(1,0)} = -\frac{\boldsymbol{g}^{(1)}}{\|\boldsymbol{g}^{(1)}\|}$ or $\boldsymbol{d}^{(K,0)} = -\boldsymbol{V}^{(K-1)}\boldsymbol{g}^{(K)}$ (steepest descent in the first iteration and a quasi-Newton direction in further iterations), then the directions of versors, $\boldsymbol{d}^{(K,j)} = \boldsymbol{e}_j = (0...1_{(j)}..0)^T$, $j = 1,..n$ (Jacobi searches along all coordinates). If $P = n + 2$, which is a special case, each processor with $\psi > 1$ has only one direction assigned. If there are less processors than $n + 2$, which is a typical case, then each processor has a queue of, say, χ directions assigned.

The basic algorithmic task for a processor with $\psi > 1$ is to perform an approximate directional search in each of these directions starting from a point $\boldsymbol{x}^{(K)}$ common to all processors:

$$\begin{aligned} \hat{\tau}^{(K,j)} &\approx \underset{\tau \in R^1}{\operatorname{argmin}} f(\boldsymbol{x}^{(K)} + \tau \boldsymbol{d}^{(j)}) \\ \boldsymbol{s}^{(K,j)} &= \hat{\tau}^{(K,j)} \boldsymbol{d}^{(j)} \qquad \text{for} \quad j = 0, \ldots n \end{aligned} \tag{7}$$

There are various approaches to directional search. Most widely accepted is the requirement of *rather precise* directional search, based on a cubic fit, gradient computations accompanying function value computations and the Wolfe-Powell stopping test inside the search, see e.g. [Fletcher (1987)].

An alternative is a *rather approximate* directional search, based on a quadratic fit, avoiding too many gradient computations and using the Goldstein stopping test that admits $\tau = 1$ as soon as possible.

It is known that the rather precise directional search gives better practical results than the rather approximate, if it is used with conjugate directions or rank-two variable metric methods. There are also various theoretical reasons why the rather precise directional search is usually prefered (see Appendix).

However, we do not propose here to use the property of conjugacy nor rank-two variable metric. In fact, in odd-numbered iterations of the pulsar algorithm, a rather approximate directional search might be preferable. In even-numbered iterations, when consitently quasi-Newton directions are used, a rather approximate version of directional search might be also advantageous. Thus, we describe here the case of using approximate directional search that starts preferably with $\tau = 1$ — while the final choice of search algorithm must be subject of testing and various arguments about this choice are presented in Appendix.

Being approximate, the directional search might fail to produce an essential improvement of the minimized function. Thus, let $\hat{\tau}^{(K,j)}$ denote not the optimal, but the value of the step-size coefficient τ at which the approximation (7) is actually stopped. This step-size coefficient should satisfy the following double test of Goldstein (see e.g. [Fletcher (1987)]; we use here strong inequalities in this test):

$$(1-\beta)\hat{\tau}^{(K,j)}\phi'^{(K,j)}(0) < \phi^{(K,j)}(\hat{\tau}^{(K,j)}) < \beta\hat{\tau}^{(K,j)}\phi'^{(K,j)}(0) \tag{8}$$

where:

$$\begin{aligned} \phi^{(K,j)}(\tau) &= f(\boldsymbol{x}^{(K)} + \tau \boldsymbol{d}^{(j)}) - f(\boldsymbol{x}^{(K)}) \\ \phi'^{(K,j)}(0) &= < \nabla f(\boldsymbol{x}^{(K)}), \boldsymbol{d}^{(j)} > \\ \beta &\in (0; 0.5) \end{aligned}$$

Rather small values of $\beta \in [0.1; 0.3]$ are suggested for an undemanding test; $\phi'^{(K,j)}(0) \leq 0$ is required with changing the sign of $\boldsymbol{d}^{(j)}$ and thus $\phi'^{(K,j)}(0)$ if necessary. Moreover, only the right-hand inequality in (8) is critical. If the left-hand inequality is violated, τ is increased a finite, given number of times κ; if the left-hand inequality is still violated, the search stops with a substantial decrease of the minimized function.

The results of directional searches determine the set of points:

$$\boldsymbol{x}^{(K,j)} = \boldsymbol{x}^{(K)} + \hat{\tau}^{(K,j)} \boldsymbol{d}^{(j)}$$

For such $j > 0$, however, that the right-hand-side inequality in (8) is violated:

$$\phi^{(K,j)}(\hat{\tau}^{(K,j)}) \geq \beta\hat{\tau}^{(K,j)}\phi'^{(K,j)}(0) \tag{9}$$

the following additional corrections are made:

$$\hat{\tau}^{(K,j)} := \tilde{\tau}^{(K)} = \min\left((0.5)^K,\ \|\boldsymbol{g}^{(K)}\|\ \right) \qquad \text{if} \quad j > 0; \tag{10}$$

If $\boldsymbol{g}^{(K)} \neq 0$, then $\phi'^{(K,j)}(0) = 0$ for $j > 0$ indicates that the Jacobi search along this coordinate produces $\hat{\tau}^{(K,j)} = 0$; the correction (10) is devised precisely for this case.

In order to control the diameter of the spread of points along all coordinates in an odd-numbered iteration of a pulsar algorithm, a parameter α is introduced that results in the points:

$$\boldsymbol{x}_\alpha^{(K,j)} = \boldsymbol{x}^{(K)} + \alpha\hat{\tau}^{(K,j)}\boldsymbol{d}^{(j)}$$

for $j > 0$. It can be shown that this parameter should be selected from the interval $(n; n^2)$ to obtain a spread of points $\boldsymbol{x}_\alpha^{(K,j)}$ that approximate the solution $\hat{\boldsymbol{x}}$ 'from all sides'.

The resulting function values, new gradients and their increments as compared to the starting point $\boldsymbol{x}^{(K)}$ are computed at the points $\boldsymbol{x}_\alpha^{(K,j)}$:

$$\begin{aligned} f_\alpha^{(K,j)} &= f(\boldsymbol{x}^{(K)} + \alpha\hat{\tau}^{(K,j)}\boldsymbol{d}^{(j)}) \\ \boldsymbol{g}_\alpha^{(K,j)} &= \nabla f(\boldsymbol{x}^{(K)} + \alpha\hat{\tau}^{(K,j)}\boldsymbol{d}^{(j)}) \\ \boldsymbol{y}_\alpha^{(K,j)} &= \nabla f(\boldsymbol{x}^{(K)} + \alpha\hat{\tau}^{(K,j)}\boldsymbol{d}^{(j)}) - \nabla f(\boldsymbol{x}^{(K)}) \end{aligned} \tag{11}$$

while $\alpha = 1$ results in $\boldsymbol{x}_\alpha^{(K,j)} = \boldsymbol{x}^{(K,j)}$ and is always used for $j = 0$. It is advatageous to compute also for $j > 0$ additionally the function values $f^{(K,j)} = f(\boldsymbol{x}^{(K,j)}$ for $\alpha = 1$, select the minimal value, compute at the corresponding point $\boldsymbol{x}^{(K,\hat{j})}$ the gradient $\boldsymbol{g}^{(K,\hat{j})}$ and use it in a corresponding stopping test described later, see (15).

The data $f_\alpha^{(K,j)}$, $\boldsymbol{g}_\alpha^{(K,j)}$, $\boldsymbol{y}_\alpha^{(K,j)}$ and $\boldsymbol{s}_\alpha^{(K,j)} = \boldsymbol{x}_\alpha^{(K,j)} - \boldsymbol{x}^{(K)}$ are successively (as they are computed) transmitted to the first processor. If a processor with $\psi > 1$ has a nonempty queue of directions assigned, it starts further computations for the next direction. Meanwhile, the first processor uses the data to update the approximation of the inverse hessian. Let:

$$\boldsymbol{r}_\alpha^{(K,j)} = \boldsymbol{s}_\alpha^{(K,j)} - \boldsymbol{V}^{(K,j)}\boldsymbol{y}_\alpha^{(K,j)}$$

There are several alternative rules for the update of variable metric. One of them is:

$$\Delta\boldsymbol{V}^{(K,j)} = \begin{cases} \dfrac{\boldsymbol{r}_\alpha^{(K,j)} \times \boldsymbol{r}_\alpha^{(K,j)}}{<\boldsymbol{r}_\alpha^{(K,j)}, \boldsymbol{y}_\alpha^{(K,j)}>} & \text{if} \quad < \boldsymbol{r}_\alpha^{(K,j)},\ \boldsymbol{y}_\alpha^{(K,j)} > \geq \gamma'\ \|\boldsymbol{y}_\alpha^{(K,j)}\|^{\,2} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\begin{aligned} \boldsymbol{V}^{(K,0)} &= \gamma''\boldsymbol{I} \\ \boldsymbol{V}^{(K,j+1)} &= \boldsymbol{V}^{(K,j)} + \Delta\boldsymbol{V}^{(K,j)} \qquad \text{for} \quad j = 0, 1, \ldots n \end{aligned} \tag{13}$$

The update (12) modifies the inverse hessian approximation only if the current data indicate that the eigenvalues of the matrix $\boldsymbol{H}^{-1} - \boldsymbol{V}^{(K,j)}$ are larger than γ'; thus, the coefficient γ' can be interpreted as a bound of these eigenvalues such that below them the approximation errors are considered unimportant. Alternatively, we can use the condition $< \boldsymbol{r}_\alpha^{(K,j)}\ ,\ \boldsymbol{y}_\alpha^{(K,j)} > \geq \gamma'\ \|\boldsymbol{y}_\alpha^{(K,j)}\|\ \|\boldsymbol{s}_\alpha^{(K,j)}\|$ in this update, in which case the coefficient γ' can be interpreted as a bound on relative accuracy of $\dfrac{\|\boldsymbol{H}^{-1} - \boldsymbol{V}^{(K,j)}\|}{\|\boldsymbol{H}^{-1}\|}$ in spectral norms.

In a sequential implementation of a variable metric algorithm, special care must be taken that $\boldsymbol{s}^{(k)}$ remain linearly independent for subsequent iterations; this is simply provided for by the choice of $\boldsymbol{d}^{(K,j)} = \boldsymbol{e}_j$ in the pulsar algorithm. In a sequential algorithm, the starting matrix $\boldsymbol{V}^{(0)}$ must be positive-definite with not too small eigenvalues - e.g. $\boldsymbol{V}^{(0)} = \boldsymbol{I}$ - because the quasi-Newton direction (4) must be a direction of improvement even if the inverse hessian approximation is incomplete. In the pulsar algorithm, however, the initial $\boldsymbol{V}^{(0)}$ is not needed for the determination of a direction of improvement. Therefore, $\boldsymbol{V}^{K,0} = 0$ could be used as a starting matrix; we propose to use instead one of the following alternatives. Either $\boldsymbol{V}^{K,0} = \gamma''\boldsymbol{I}$, where $\gamma'' = \frac{1}{\mu}$ is a lower bound on the eigenvalues of the inverse hessian (μ is an upper bound of hessian eigenvalues); this results in the following property of the approximation:

$$0 < \boldsymbol{V}^{(K,0)} \leq \boldsymbol{V}^{(K,1)} \leq .. \leq \boldsymbol{V}^{(K,j)} \leq \boldsymbol{V}^{(K,j+1)} \leq .. \leq \boldsymbol{V}^{(K,n)} \leq \boldsymbol{H}^{-1} \tag{14}$$

where the inequalities are understood in the sense of positive-definiteness of matrix differences. This variant will be called inverse hessian approximation 'from below'.

Or, if we have an upper bound $\frac{1}{\nu}$ on the eigenvalues of the inverse hessian (ν is a lower bound of hessian eigenvalues), we can start also the approximation 'from above' by setting $\gamma'' = \frac{1}{\nu}$, in which case the signs of inequalities in (14) would change. The approximation 'from above' gives usually better computational results in sequential implementations, see [Kręglewski and Wierzbicki (1981)]. However, in the approximation 'from above' we must change also the sign of the condition in (12) to read

$$\text{if} \quad < \boldsymbol{r}_\alpha^{(K,j)}, \boldsymbol{y}_\alpha^{(K,j)} > \leq -\gamma' \, \|\boldsymbol{y}_\alpha^{(K,j)}\|^2$$

or correpondingly with $\|\boldsymbol{y}_\alpha^{(K,j)}\|^2$ replaced by $\|\boldsymbol{y}_\alpha^{(K,j)}\| \, \|\boldsymbol{s}_\alpha^{(K,j)}\|$.

If the number of $n+1$ directions that need parallel processing is not an integer multiple of $P-1$, then additional directions $\boldsymbol{d}^{(K,j)}$ for $j > n$ can be generated e.g. as some linear combinations of the directions $\boldsymbol{d}^{(K,0)}$ and $\boldsymbol{e}_j$. The number of additional directions should be such that all processors have assigned the same number $\chi \geq \frac{n+1}{P-1}$ of directions to be processed. The resulting data should be used then in formulae (12), (13) for $j = 0, 1, ..., n'$ where $n' = \chi(P-1) - 1$. If such data does not add new information, it indicates that the matrix $\boldsymbol{H}^{-1}$ is already well approximated; if the data adds new information, the approximation of $\boldsymbol{H}^{-1}$ is improved.

Thus, the odd-numbered (divergent) iteration of the pulsar algorithm is in fact a parallel Jacobi-type iteration but with data used for a rank-one variable metric approximation of the inverse hessian matrix. The actual implementation of this approximation might use the formulae (12), (13) either directly or in a dual (suitably applied to the matrices $\boldsymbol{B}^{(K)} = (\boldsymbol{V}^{(K)})^{-1}$, while the data $\boldsymbol{s}_\alpha^{(K,j)}$ and $\boldsymbol{y}_\alpha^{(K,j)}$ is interchanged to obtain an update for $\Delta\boldsymbol{B}^{(K)}$) and appropriately factorized form.

The iteration ends when the first processor determines such $\hat{j}$ that corresponds to the minimal value of $f^{(K,j)}$ and checks whether:

$$\|\boldsymbol{g}^{(K,\hat{j})}\| \leq \varepsilon', \tag{15}$$

$$\|\boldsymbol{x}^{(K,\hat{j})} - \boldsymbol{x}^{(K)}\| \leq \varepsilon'' \tag{16}$$

$$\text{and sets } \boldsymbol{x}^{(K+1)} := \boldsymbol{x}^{(K,\hat{j})} \tag{17}$$

where ε' is an assumed gradient accuracy and ε'' is an assumed approximate solution accuracy. This stopping test can be applied because the algorithm actually converges

superlinearly (for non-quadratic functions with uniformly positive and bounded hessian). If these stopping tests are positive, the iterations end; if no, then $\boldsymbol{V}^{(K)} = \boldsymbol{V}^{(K,n'+1)}$ is sent to all other processors to be used in computing new directions of improvement. The computation of $\|\boldsymbol{g}^{(K,j)}\|$, the determination of $\hat{j}$, even the stopping tests might be performed in the first processor successively as the data arrive; this limits the delay of sending $\boldsymbol{V}^{(K)}$ to other processors after stopping approximation.

In an even-numbered (convergent) iteration of a pulsar algorithm the directional searches are performed from various points $\boldsymbol{x}_\alpha^{(K,j)}$ (each processor might store the data $\boldsymbol{x}_\alpha^{(K,j)}$, $\boldsymbol{g}_\alpha^{(K,j)}$ related to the directions assigned to it) and not in versor directions $\boldsymbol{e}_j$, but in quasi-Newton directions:

$$\boldsymbol{d}^{(K+1,j)} = -\boldsymbol{V}^{(K)}\boldsymbol{g}_\alpha^{(K,j)} \tag{18}$$

(for the sake of clarity of exposition, the count K is not increased here, $K :\neq K+1$). Other operations in an even-numbered iteration are similar as in an odd-numbered iteration; since there are no Jacobi searches in an even-numbered iteration, only $\alpha = 1$ is used in this case.

The essential feature of an even-numbered iteration is that all points $\boldsymbol{x}^{(K+1,j)}$ obtained in it should rather precisely approximate the minimum of the goal function f - if it is quadratic or if a sufficiently close neighborhood of a minimum of a non-quadratic twice differentiable function is already reached. This convergence to the minimum by a cluster of points, "from various sides", is advantageous: it increases the robustness of the algorithm with respect to numerical inaccuracies, to the ill-conditioning of the optimization problem, to local minima and even to the inaccuracies of the stopping tests of the optimization algorithm.

An odd-numbered and an entire double iteration of the pulsar algorithm ends when the first processor determined not only a new inverse hessian approximation but also a new common starting point $\boldsymbol{x}^{(K+2)}$ for the next iteration. This point is chosen between $\boldsymbol{x}^{(K+1,j)}$ as such that corresponds to the lowest $f^{(K+1,j)}$. The stopping tests (15) are repeated for $K := K + 1$.

Note that the first processor will always determine the time of an iteration of the variable metric pulsar algorithm. Thus, an efficient implementation should aim at such number χ of directions assigned to other processors that the computational capabilities of these processors are best used (the loads of processors are balanced). In other words, when starting for a given n with a low number of processors P and the correspondingly large χ and then changing these numbers, the actual computing time will first decrease with the increase of P and decrease of χ, but this decrease of computing time will stop when an optimal number χ is reached: further increase of the number of processors will not influence the execution time essentially. This optimal number χ of directions assigned to each processor will depend on the ratio of computational effort needed for the directional search to the effort needed for so-called housekeeping operations, mainly for inverse hessian approximation.

While the number of function evaluations $f(\boldsymbol{x})$ needed for an approximate directional search algorithm (see Appendix) grows usually slower than linearly with the problem dimension n, the effort for inverse hessian approximation grows rather as n^2 or even n^3, depending on implementation. Thus, the ratio of these efforts depends critically on the effort needed for one function evaluation, which might be small for test problems of academic character and very large for actual applications in the case of rather complicated models which determine function values. The value $\chi = 1$ might be optimal for problems

of small dimension and with complicated models that determine function values; but even for the case of complicated models, the optimal number χ might grow with n.

Note that if the function f is not quadratic, then the approximation $\boldsymbol{V}^{(K-1)} \cong \boldsymbol{H}^{-1}$ obtained either in the odd- or even-numbered iteration of a pulsar iteration is based on different points and data than in a sequential variable metric algorithm. Thus, it is necessary to analyze anew the rate of convergence of such an algorithm; the results of [Conn et al. (1991)] can be appropriately modified to conclude a superlinear convergence of the rank-one variable metric parallel pulsar algorithm. Actually, a convergence order even distinctly higher than 1 can possibly be established for quasi-Newton pulsar algorithms (see further remarks on the convergence order of the Newton-type variant). However, the main advantages of a parallel pulsar algorithm should relate rather to a fast and reliable attainment of the asymptotic convergence region and to robustness with respect to numerical inaccuracies than to an even higher convergence order.

2.2 Basic variant of Newton-type pulsar algorithm.

In a modified Newton-type parallel pulsar algorithm, instead of computing the inverse hessian approximation, the first processor can directly compute the inverse of the hessian matrix $\boldsymbol{V}^{(K)} = (\nabla^2 f(\boldsymbol{x}^{(K)}))^{-1}$ (at a point corresponding to the lowest function value from the previous iteration). An efficient implementation of such an algorithm should take into account an appropriate factorization of the matrix as well as an algebraic (symbolic) preparation of all formulae for first- and second- order derivatives.

One of the disadvantages of Newton-type algorithms is their sensitivity to numeric inaccuracies; however, the pulsating of results in a Newton-type pulsar algorithm can essentially improve its robustness. Additionally, the speed of the algorithm far away from the minimum can be also improved; close to the minimum a Newton-type algorithm is known to be very fast anyway and a careful implementation of a parallel pulsar algorithm should preserve the rapid convergence of a Newton-type method.

This does not mean that the convergence order $p = 2$ of a typical sequential Newton-type method will be preserved. Note that the pulsar Newton type-algorithm performs actually the following operations:

• In an odd-numbered (divergent) iteration starting with a point $\boldsymbol{x}^{(K)}$, the Jacobi searches along all coordinates as well as the search along a Newton-type direction is performed:

$$\begin{aligned} \boldsymbol{d}^{(K,0)} &= -(\boldsymbol{H}^{(K-1)})^{-1}\boldsymbol{g}^{(K)} \qquad \text{for} \quad K > 1 \\ \boldsymbol{d}^{(1,0)} &= -\boldsymbol{g}^{(1)} \end{aligned} \tag{19}$$

where:

$$\begin{aligned} \boldsymbol{H}^{(K-1)} &= \nabla^2 f(\boldsymbol{x}^{(K-1)}) \\ \boldsymbol{g}^{(K)} &= \nabla f(\boldsymbol{x}^{(K)}) \end{aligned}$$

The points $\boldsymbol{x}_\alpha^{(K,j)}$ are determined similarly as in the variable metric version of a pulsar algorithm. *Parallel,* the computations of $(\boldsymbol{H}^{(K)})^{-1}$ are performed in the first processor.

• In an even-numbered (convergent) iteration, starting with many points $\boldsymbol{x}_\alpha^{(K,j)}$ that are results of modified Jacobi and Newton-type searches in former iteration, new searches along many Newton-type directions are performed:

$$\begin{aligned} \boldsymbol{d}^{(K+1,j)} &= -(\boldsymbol{H}^{(K)})^{-1}\boldsymbol{g}_{\alpha}^{(K,j)} \\ \boldsymbol{g}_{\alpha}^{(K,j)} &= \nabla f(\boldsymbol{x}_{\alpha}^{(K,j)}) \end{aligned} \tag{20}$$

The points $\boldsymbol{x}^{(K+1,j)}$ (with $\alpha = 1$) are thus determined. *Parallel,* the computations of $(\boldsymbol{H}^{(K+1)})^{-1}$ are again performed in the first processor.

Note that the essence of this algorithm is the use of $(\boldsymbol{H}^{(K-1)})^{-1}$ *delayed* by one iteration as compared to a sequential algorithm, in order not to wait on the time-consuming matrix inversion (n.b. it is better to *actually invert* the matrix - via suitable factorization - and to send the inverted hessian to all other processors to be used in the determination of diverse directions for the next iteration, instead of solving an equivalent system of linear equations which is preferable in sequential implementations). In this case, the first processor *will not determine* the time of entire pulsar iteration and the issue of balancing processor loads might be resolved differently than in a variable metric pulsar algorithm,

For this use of one-iteration-delayed data in the inverted hessian, however, we pay in the estimate of asymptotic convergence order: it can be shown (the full description of the convergence properties of pulsar algorithms is deferred to the next version of the paper) that instead of $p = 2$, the convergence order of the pulsar algorithm is only $p = \frac{\sqrt{5}+1}{2} \cong 1.618$, while counting separately single phases as iterations. If we counted both phases of a pulsar algorithm jointly as one iteration, we could obtain $p = \frac{\sqrt{5}+3}{2} \cong 2.618$; but it is known that neither convergence rates nor convergence orders are invariant under aggregation of several iterations into a bigger one.

Note that we could substitute the directional searches in all variants of parallel pulsar algorithms described above by the determination of a *restricted step* as in trust region methods, see e.g. [Fletcher (1987)]. This possibility is discussed in the next section.

2.3 Regularized variants of pulsar algorithms.

If there are many processors in a scalably parallel architecture and the question is how to reasonably use them, then the pulsar algorithm might use additional directions generated as linear combinations of basic directions, as indicated above. However, the main difficulty of nonlinear optimization for higher dimensions n is that such problems are usually badly conditioned: the larger is n, the more probable it is that the hessian matrix $\boldsymbol{H}(\boldsymbol{x})$ has a large conditioning index, i.e. the ratio of its largest and smallest eigenvalue. Thus, additional computing power should be used to regularize the optimization problem.

The original problem of minimizing $f(\boldsymbol{x})$ can be regularized in various ways. A reliable way of such regularization is the *proximal point method* - a parametric imbedding into a family of problems of minimizing the function:

$$f_{\rho}(\boldsymbol{x}) = f(\boldsymbol{x}) + 0.5\rho \, \|\boldsymbol{x} - \boldsymbol{x}^{(K)}\|^{\,2} \tag{21}$$

where $\rho > 0$ can be interpreted as:

- a regularization parameter, related to the eigenvalues of the hessian matrix;
- a Lagrange multiplier for minimizing the function $f(\boldsymbol{x})$ subject to $\|\boldsymbol{x} - \boldsymbol{x}^{(K)}\| \leq h^{(K)}$, where $h^{(K)}$ is a restriction of the step-size;
- an operational parameter changed adaptively in trust region methods and restricted step determination, depending on a comparison of a quadratic approximation of minimized function to its actual value, see e.g. [Fletcher (1987)].

The latter interpretations are related to trust region methods and resticted step determination. The restricted step determination — appropriately decomposed into algorithmic tasks of hessian inversion or approximation performed jointly and actual step determination performed parallel, possibly from various starting points — can be the basis of alternative variants of pulsar algorithms. However, we defer these issues for further study and concentrate on the first interpretation of the parameter ρ.

An essential feature of proximal point methods is the improvement of the conditioning index of the hessian matrix:

$$\begin{aligned} \nabla f_\rho(\boldsymbol{x}) &= \nabla f(\boldsymbol{x}) + \rho(\boldsymbol{x} - \boldsymbol{x}^{(K)}) \\ \nabla^2 f_\rho(\boldsymbol{x}) &= \nabla^2 f(\boldsymbol{x}) + \rho \boldsymbol{I} \end{aligned} \tag{22}$$

Since increasing ρ results in a similar increase of the eigenvalues of $\nabla^2 f_\rho(\boldsymbol{x})$, the ratio of the largest and the smallest eigenvalue decreases. Thus, in a regularized parallel pulsar Newton-type algorithm it might be useful e.g. to invert the matrix $\nabla^2 f_\rho(\boldsymbol{x})$ for several values of ρ (while an efficient factorization algorithm used for this inversion should take advantage of the particular form (22) of this matrix). The values of the parameter ρ should be chosen from an interval that approximates (from above) the lower eigenvalues of the matrix, $\rho > \nu$ where ν is a lower bound of hessian eigenvalues. The interval of chosen values of ρ might also depend on the selected value of γ'' used in (13) in such a way as to start with $\rho = 0$ or $\rho = \gamma''$ but to obtain finally $\rho \gg \gamma''$. If a number ϑ of such parameter values is chosen, either one or up to ϑ processors for matrix inversion or approximation might be reserved. For example, a variable metric approximation might either use ϑ processors for parallel approximations of inverse hessian with various values of the parameter ρ, or employ only one processor for this purpose.

In a regularized pulsar parallel variable metric algorithm with directional searches, the proximal point method corresponds to a modification of the gradient increments $\boldsymbol{y}_\alpha^{(K,j)}$:

$$\boldsymbol{y}_\alpha^{(K,j)} := \boldsymbol{y}_\alpha^{(K,j)} + \rho \boldsymbol{s}_\alpha^{(K,j)} \tag{23}$$

The values of gradients $\nabla f(\boldsymbol{x})$ used in this method to determine the Newton or quasi-Newton search directions need not be modified - if the reference point $\boldsymbol{x}^{(K)}$ in formulae (21) , (22) is shifted to points $\boldsymbol{x}_\alpha^{(K,j)}$ for directional searches starting from these points.

If the computational efforts needed for matrix inversion, variable metric approximation and the directional search are comparable - which is a rather special case - then $\chi = 1$ in the basic variant of the parallel pulsar variable metric method and at least $n+2$ processors are needed. In order to increase the chances of solving an ill-conditioned problem, the number of its regularized variants for various values of ρ is increased by ϑ times, thus $P = (n+2)\vartheta$ processors are needed. Given a large number P processors in a parallel architecture, the number of variants solved might be $\vartheta = \frac{P}{n+2}$, if each of processors is used only once for directional search. In a more general case, when the computational efforts are different and the processors assigned to directional search can be used χ times in each iteration, $\vartheta = \frac{P}{1+\frac{n+1}{\chi}}$. Therefore, if $P > n+2$ and $\chi > 1$, this reserve of computational power can be used for solving ill-conditioned optimization problems. If $\frac{N}{1+\frac{n+1}{\chi}}$ is not an integer, then the reserve of computing power can be assigned to additional directions generated as linear combinations of basic ones.

The organization of the algorithm is thus similar as in its basic version with the difference that from 1 to ϑ processors might be assigned to matrix inversion or approximation.

It is then necessary to select best values of the goal function f obtained for various ρ and modify appropriately the stopping test and the selection of the common starting point for an odd-numbered, divergent iteration of the pulsar algorithm.

3 Computational experiments.

In order to preliminary test the potential of the concept of pulsar algorithms, a simple computational test was performed. The simplest, basic variant of the variable metric pulsar algorithm with $\chi = 1$ (each processor is assumed to perform only one directional search) was simulated on a sequential computer architecture (SUN SparcStation) and the computing times for simulated parallel processors were recorded. This rough experiment was intended just to test the basic concepts and, in particular, the robustness of pulsar algorithms. As a simple but demanding testing problem the [Schittkowski (1987)] problem No. 290 was used. The minimized function in this problem is quartic, i.e. it is a quadratic function of two variables that is additionally squared, $f(\boldsymbol{x}) = < \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x} >^2$. The problem is demanding for Newton-type or quasi-Newton algorithms, because the hessian matrix converges to a zero-matrix as the solution approaches the minimal point $\hat{\boldsymbol{x}} = \mathbf{o}$. Thus, a variable metric or a Newton-type algorithm must fail when it approaches the optimal solution sufficiently close. The following Table 1 shows the comparison of results for a sequential and a simulated pulsar parallel implementation of the symmetric rank-one variable metric.

The results indicate that the parallel pulsar implementation is much more robust indeed than the sequential implementation: while the sequential one failed on this problem after 17 iteration practically without producing useful results, the parallel pulsar worked and produced at this iteration a squared gradient norm less than 10^{-32}; it failed first after 42 iteration when producing a squared gradient norm less than 10^{-55}.

Results of sequential and pulsar parallel variants of the rank-one symmetric variable metric for a quartic problem

Iteration number	5	10	15	20	30	40	50
Sequential				failed	failed	failed	failed
solution norm^2	0.68E+00	0.72E+00	0.71E+00				
gradient norm^2	3.62E+01	3.34E+01	3.32E+01				
Pulsar parallel							failed
solution norm^2	5.69E-04	1.76E-07	5.21E-11	4.46E-13	4.26E-17	1.48E-19	
gradient norm^2	3.87E-09	1.03E-19	1.47E-24	1.54E-36	1.80E-48	1.07E-55	

These results show not only a substantially increased robustness but also an essential acceleration of computations per iteration; the total computing time is clearly longer in the parallel variant when simulated on sequential computer, but the simulated time per iteration (assuming actually parallel execution of computations) is comparable for the sequential and the parallel variant. Thus, the time to reach a given precision - say, a typical demand 10^{-7} for the gradient norm that translates to 10^{-14} in the table above - might be much smaller: for the example presented in the table above, it is reached in 8 iterations by the parallel pulsar and never by the sequential implementation.

Another test involved n-dimensional generalizations of Rosenbrock banana function. In fact, there are several such generalizations - more complicated that introduce additional local minima and stationary points as well as simpler ones that preserve the property of the uniqueness of the minimum; both cases were investigated. The results reported in the following table relate to the simpler case for dimensions $8, 16, 32$, and to the more difficult one for dimension 20. Actually parallel computations were performed on Paragon[1]

[1]Paragon is a trademark of Intell Corporation.

multiprocessor parallel computer.

Rosenbrock's bananna valley problem for various n: number of iterations needed for 10^{-6} accuracy in gradient norm				
Problem dimension	8	16	32	20
Sequential	61	93	136	106
Pulsar simulated parallel	43	36	58	72
Pulsar actually parallel	39	68	116	68

Note that the observed acceleration of computations is not due to the full utilization of computing power, because with $\chi = 1$ most "processors" were under-utilized in these tests. The acceleration is thus due to the basic properties of the pulsating, divergent-convergent algorithm. In the divergent phase, searches in various directions result in much better starting points for the next phase than in a sequential implementation. In the case of less regular functions, these searches might also account for various irregularities of the goal function that cannot be tested in a sequential algorithm. Thus, the potential of pulsar algorithms is considerable; clearly, it can be much better utilized than in this simple test. Much more detailed investigation of the properties of pulsar algorithms are thus substantiated, including testing on a wide range of examples, implementations on actual parallel and distributed computer architectures and various variants of utilizing the available computing power more efficiently.

4 Conclusions.

Although only very preliminary tests of simple variants of pulsar algorithms are presented in this paper, they fully support the theoretical conjecture of a large potential of the concept of a double-phased, divergent-convergent parallel pulsating algorithms for nonlinear optimization. Beside the necessary further research on the Newton-type and quasi-Newton pulsar algorithms for unconstrained differentiable optimization, research on applications of this concept in constrained and nondifferentiable optimization is also intended.

A Appendix: An Algorithm of Directional Search.

Since directional search constitutes the main unit of the algorithmic phases of a pulsar algorithm, it is important to apply an efficient version of this search. There are rather diversified opinions in the literature on this subject. For example, an otherwise excellent survey of optimization techniques in [Gill et al. (1981)] uses the argument that the methods of polynomial approximation, when applied for directional search, have only linear and slow convergence rate and thus other methods are preferable. On the other hand, since the directional search constitutes only a sub-iteration in an iterative nonlinear optimization algorithm, convergence rate arguments do not apply in this case, particularly if the search is used with a Newton-type or quasi-Newton algorithm.

In [Fletcher (1987)] a rather precise directional search algorithm based on polynomial approximation is presented as the most preferable. This algorithm uses cubic fit and the Wolfe-Powell stopping test (a bound on the absolute value of directional derivative) that both rely on gradient computations which are performed whenever a value of the minimized function is computed. There are many theoretical and practical arguments that support such a choice. If structural properties of the minimized function are utilized, the computations of gradient values are seldom more expensive than computing only several (as opposed to n) function values, see e.g. [Wierzbicki (1984)]. A precise directional search is usually assumed in the theory of conjugate directions and rank-two

variable metric methods; computational experience shows best results for such methods with rather precise search. Thus, the directional search based on a cubic fit with gradient computations is widely accepted as a standard.

To select an efficient directional search for Newton-type methods, however, it should be noted that its stopping test might be rather imprecise in order to admit as soon as possible the unit step-size that is sufficient for Newton-type methods close to the optimum. Moreover, the unit step-size is asymptotically admissible for such methods even if the double Goldstein stopping test is used — which is simpler than the Wolfe-Powell test. This fact is also noted in [Fletcher (1987)], but without concluding that the Goldstein test might be better for Newton-type methods. Gradient computations might cost only several times as much as function computations, nevertheless they should be avoided when not needed. Cubit fit might be substituted by quadratic fit, if high accuracy of directional search is not needed.

Thus, for the basic variants of a pulsar algorithm the alternative of rather approximate directional search is proposed (while the option of rather precise directional search with cubic fit should be also tested). Recall the denotation:

$$\begin{aligned} \phi(\tau) &= f(\boldsymbol{x} + \tau \boldsymbol{d}) - f(\boldsymbol{x}) \\ \phi'(0) &= < \nabla f(\boldsymbol{x}), \boldsymbol{d} > \end{aligned} \tag{24}$$

where $\boldsymbol{d}$ is the search direction and the functions f, ϕ are supposed to be minimized. If $\phi'(0) < 0$, then the search is performed for $\tau \geq 0$ (otherwise, we can substitute $\boldsymbol{d} := -\boldsymbol{d}$; we consider only the case $\tau \geq 0$ here). An initial step-size coefficient τ_1 should be given; in Newton-type or quasi-Newton methods of optimization, it is reasonable to use standard $\tau_1 = 1$. In Jacobi searches along directions $\boldsymbol{e}_j$, $\tau_1 = 1$ can be also used in the first iteration of the pulsar algorithm because of lack of better information; in further iterations, however, either $\tau_1 = \tilde{\tau}^{(K)}$ (see (10)) or $\tau_1 = \hat{\tau}^{(K-2,j)}$ might be preferable.

After computing $\phi(\tau_1)$ there is enough data to determine a *two-point quadratic approximation* of ϕ, of the form $\tilde{\phi}(\tau) = \phi'(0)\tau + 0.5a\tau^2$, where the coefficient a is computed as:

$$a = \frac{2(\phi(\tau_1) - \phi'(0)\tau_1)}{\tau_1^2}$$

If $a \leq 0$ (if $\phi(\tau_1) \leq \phi'(0)\tau_1$) is obtained, then the quadratic approximation would have a maximum instead of minimum and cannot be used. But this is checked by a stronger requirement — the double-sided Goldstein test, with an accuracy coefficient $\beta \in (0; 0.5)$:

$$(1 - \beta)\tau\phi'(0) < \phi(\tau) < \beta\tau\phi'(0) \tag{25}$$

which is modified here slightly by using strong instead of weak inequalities. If a value of β not too close to 0.5 is used, then this test is rather undemanding or *broad*; if β is close to 0.5, then this test is demanding or *narrow*. In practice, $\beta = 0.20$ gives a test that is reasonably broad and sufficient for most applications; in the first, divergent phase of a parallel pulsar algorithm, even $\beta = 0.1$ can be applied.

If the Goldstein test is satified by $\phi_1 = \phi(\tau_1)$, then the search stops. If the left-hand side inequality in (25) is violated, τ is increased iteratively e.g. by substituting $\tau_1 := 2\tau_1$ until the left-hand side is satisfied and thus $a > 0$ is obtained - or an assumed number κ of goal function evaluations is utilized. In the latter case, the directional search stops with a substantive (more than $(1 - \beta)|\phi'(0)\tau|$) decrease of the minimized function, which

can happen only far from the optimal solution and is sufficient there. If the left-hand side of the test (25) is satisfied but its right-hand side is violated, τ_2 is determined that corresponds to the minimum of the quadratic approximation $\tilde{\phi}(\tau)$:

$$\tau_2 = \frac{-\phi'(0)\tau_1^2}{2(\phi(\tau_1) - \phi'(0)\tau_1)} \tag{26}$$

while $\tau_2 < \tau_1$ (which is implied by the violation of the right-hand side of (25)). Then $\phi_2 = \phi(\tau_2)$ is computed. The Goldstein stopping test is checked again. If the test fails, one of the following cases is considered:

a) if $\phi_2 > \phi_1$, which might happen only in rather irregular cases, then it is reasonable to repeat two-point quadratic approximation based on data τ_2, ϕ_2 (we give preference to the local minimum closer to $\tau = 0$ in such a case).

Otherwise:

b') put $\tau_0 := 0$, $\phi_0 := 0$, reorder τ_1 and τ_2 together with corresponding values ϕ_1 and ϕ_2 in such a way that $\tau_1 < \tau_2$ and start a three-point quadratic approximation which does not use the data $\phi'(0)$.The data τ_0, ϕ_0, τ_1, ϕ_1, τ_2, ϕ_2 in this case is such that the approximating quadratic function has its minimum at the point:

$$\tau_3 = \frac{(\phi_0 - \phi_1)(\tau_2^2 - \tau_1^2) + (\phi_2 - \phi_1)(\tau_1^2 - \tau_0^2)}{2((\phi_0 - \phi_1)(\tau_2 - \tau_1) + (\phi_2 - \phi_1)(\tau_1 - \tau_0))} \tag{27}$$

Thus, τ_3 and $\phi_3 := \phi(\tau_3)$ is computed and the stopping test (25) performed on this data together with the test on the number of function values computed, see further comments. If these tests fail, then:

b") Additionally, if $\phi_2 > 0$ — which might indicate that $\tau_2 - \tau_1$ is much larger than $\tau_1 - \tau_0$, in which case the approximation might be not quite satisfactory — or if $\tau_3 = \tau_2$ — which, together with the failure of the stopping test, indicates that the approximation is not useful any more — determine $\tau_4 = \frac{\tau_3 + \tau_2}{2}$ and $\phi_4 = \phi(\tau_4)$.

Reorder τ_i together with corresponding ϕ_i to obtain:

$$\tau_0 < \tau_1 < \tau_2 < \tau_3 \ (< \tau_4)$$

set:

$$\begin{aligned} \hat{i} &= \operatorname*{argmin}_{i=0,\ldots 3(4)} \phi_i \\ \tau_0 &:= \tau_{\hat{i}-1}; \quad \tau_1 := \tau_{\hat{i}}; \quad \tau_2 := \tau_{\hat{i}+1} \\ \phi_0 &:= \phi_{\hat{i}-1}; \quad \phi_1 := \phi_{\hat{i}}; \quad \phi_2 := \phi_{\hat{i}+1} \end{aligned} \tag{28}$$

and return to computing τ_3 as in (27). The data for this approximation are chosen again in such a way that (27) determines the minimum of the approximating function.

The stopping tests performed before the substitutions (28) are twofold. The first is the Goldstein stopping test (25). This test might be satisfied even after only a few goal function evaluations, if the problem is sufficiently regular. E.g. if the goal function is sufficiently well approximated by a quadratic one and a Newton direction $\boldsymbol{d} = -(\nabla^2 f(\boldsymbol{x}))^{-1}\nabla f(\boldsymbol{x})$ is used, then the test (25) is usually satisfied even by $\tau_1 = 1$ — moreover, this is certain for sufficiently advanced iterations of a quasi-Newton or Newton-type method. The test (25) has been criticized on the grounds that a directional minimum might not satisfy it if the function is far from quadratic one. However, the second stopping test applied here takes care of such cases.

The second stopping test is simply the bound κ on the number of evaluations of the goal function f or ϕ. If this bound is reached, it is necessary to check whether at all such a value of τ is found that $\phi(\tau) < 0$; if not, then a sequential optimization algorithm should be stopped with an error signal that the directional minimization failed (which might be caused e.g. by a bad scaling of the goal function or some optimization variables). In the parallel pulsar method, the failure of one directional search does not necessitate stopping the entire algorithm; however, a sufficient spread of points in the odd-numbered, diverging iterations of this method is needed. Thus, the test whether $\phi(\tau) \geq \beta\tau\phi'(0)$ is performed at the value of τ where the search was stopped; if yes, the correction (10) in the odd-numbered iterations of the pulsar algorithm is applied. The pulsar algorithm might be stopped with an error signal e.g. if more than half directional searches fail in its even-numbered iteration.

The approximate directional search algorithm presented here, although rather complicated in its logic, combines the efficiency and speed for regular cases with robustness for less regular ones. In a parallel pulsar variable metric or modified Newton-type algorithm, where the main unit of algorithmic phases is the directional search, the method described here might be preferable; but alternative methods should be also tested.

A reasonable bound of the number of goal function evaluations used as one of stopping tests in the described method depends on the dimension n of the optimization problem but grows slower than linearly with it; a rule-of-thumb formula:

$$\kappa = \kappa_0(1 + \sqrt[4]{n}) \qquad \text{with e.g.} \quad \kappa_0 = 5 \tag{29}$$

might be used to determine this parameter. Thus, the computational effort needed for directional search might grow more slowly with n than the effort necessary for hessian matrix inversion or approximation; clearly, the comparison of these efforts depends mostly on the complexity of the mathematical model that determines the goal function f.

References

[Bertsekas et al. (1989)] Bertsekas D.P. and J.N. Tsitsiklis (1989) *Parallel and Distributed Computation.* Prentice Hall, Englewood Cliffs.

[Carey (1989)] Carey, G.F., ed. (1989) *Parallel Supercomputing: Methods, Algorithms and Applications.* Wiley, Chichester - New York.

[Chronopulos et al. (1989)] Chronopoulsos, A.T., and C.W. Gear (1989) s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics* Vol. 25 pp. 153-168.

[Conn et al. (1991)] Conn, A.R., N.I.M. Gould and Ph.L. Toint (1991) Convergence of quasi-Newton matrices generated by the symmetric rank-one update. *Mathematical Programming* Vol. 50 pp. 177-195.

[Fletcher (1987)] Fletcher, R. (1987) *Practical Methods of Optimization.* Wiley, Chichester - New York.

[Gill et al. (1981)] Gill, E.P., W. Murray and M.H. Wright (1981) *Practical Optimization.* Academic Press, London-New York.

[Grauer et al. (1991)] Grauer, M. and D.B. Pressmar, eds. (1991) *Parallel Computing and Mathematical Optimization.* Springer-Verlag, Berlin-Heidelberg.

[Kręglewski and Wierzbicki (1981)] Kręglewski, T. and A.P.Wierzbicki (1981) Further properties and modifications of the rank-one variable metric method. In S. Walukiewicz and A.P. Wierzbicki, eds.: *Methods of Mathematical Programming*, PWN Warsaw.

[Nogi (1986)] Nogi, T. (1986) Parallel computation. In: Patterns and Waves - Qualitative Analysis of Nonlinear Differential Equations, *Studies in Mathematics and Applications* Vol. 18 pp. 279-318.

[Paczyński (1993)] Paczyński, J. (1993): Models of Dynamic Discrete Nonlinear Systems - Issues of Description Languages and Preprocessing. Report of the Institute of Automatic Control, Warsaw University of Technology.

[Ruszczyński (1989)] Ruszczyński, A. (1989) An augmented Lagrangian decomposition method for block diagonal linear programming problems, *Operations Research Letters 8*, pp. 287-294.

[Schittkowski (1987)] Schittkowski, K. (1987) *More test Examples for Nonlinear Programming Codes* Springer-Verlag, Berlin-Heidelberg.

[Stachurski (1981)] Stachurski, A. (1981) Superlinear convergence of Broyden's bounded θ-class of methods, *Mathematical Programming 20* pp. 196-212.

[Wierzbicki (1984)] Wierzbicki, A.P. (1984) Models and Sensitivity of Control Systems. Elsevier - WNT, Amsterdam-Warsaw.

[Wierzbicki (1993)] Wierzbicki, A.P. (1993) Augmented Simplex - a Modified and Parallel Version of Simplex Method Based on Multiple Criteria and Subdifferential Optimization Approach. Report of the Institute of Automatic Control, Warsaw University of Technology.