SIAG/OPT

Views-and-News

A Forum for the SIAM Activity Group on Optimization

No. 9 Fall 1997

Contents

Case Studies
pPCx: Solving Linear Programs in Parallel,
T.F. Coleman, C. Sun, M. Wagner 1
Chairman's Column
Jorge Moré 5
Essays
I Know It When I See It: Toward a Definition
of Direct Search Methods, M.W. Trosset 7

Case Studies

pPCx: Solving Linear Programs in Parallel

Thomas F. Coleman
Computer Science Department and Center for Applied Mathematics, Cornell University, Ithaca, NY, 14853

Chunguang Sun
Advanced Computing Research Institute, Cornell Theory Center,Cornell University,Ithaca, NY 14853

Michael Wagner
School of Operations Research and Industrial Engineering,Cornell University,Ithaca, NY 14853n

1. Introduction

With the use of linear programming techniques in industrial applications rising steadily, there is a definite need for efficient and robust solvers that can handle very large problems. Nowadays high–performance parallel machines make this vision possible. This article describes a step that was taken recently at Cornell and Argonne National Lab towards this goal: a parallel LP–solver based on interior point methods, targeted at the distributed memory IBM SP2 architecture and implemented entirely in C with MPI extensions.

Until 10 years ago, the Simplex method dominated the linear programming world; a lot of mindpower has produced efficient and sophisticated implementations of this algorithm. However, it is not easily parallelized and thus the memory resources available to a single processor places a bound of sorts on the problem size that can be handled by the simplex approach. With the revolutionary development of interior point (IP) methods for linear programming, this picture has changed significantly. Not only has this IP–framework shown itself to be a very robust and efficient (and polynomial) algorithm for linear programming, it is also much more easily parallelized since most of the computational work is done in the solution of a positive (semi–) definite system of linear equations. Now that a number of good serial implementations are freely available, it is time to do the next step and produce a parallel code to enable the solution of large problems.

2. Background

Let's first introduce the main ideas of the underlying algorithm. We'll be brief and restrict ourselves to exposing only the key elements of the algorithm where most of the computational work lies since we mainly want to focus on the parallel aspect of such a method. Readers interested in more details of the theory, definitions and convergence proofs are referred to the book by S. Wright [8], though it is not essential to know these details to grasp the gist of this article. We begin by stating a simple form of the linear programming problem and its dual:

$$\begin{array}{rl} \max\limits_{x} & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

and

$$\min_{\pi,s} \qquad b^T \pi$$

$$\text{s.t.}\quad A^T\pi + s = c$$
$$s \geq 0.$$

The well–known necessary and sufficient optimality conditions for this problem say that an optimal triple (x^*, π^*, s^*) with $x^* \geq 0, s^* \geq 0$ is a zero of the nonlinear map

$$F(x,\pi,s) = \begin{pmatrix} Ax - b \\ A^T\pi + s - c \\ XSe \end{pmatrix} \qquad (1)$$

with

$$X = \mathrm{Diag}(x), \quad S = \mathrm{Diag}(s), \quad e = (1,\ldots,1)^T.$$

One key element of interior–point methods is that they generate iterates (x_k, π_k, s_k) that are strictly feasible w.r.t. the inequality constraints, i.e., the primal variables x_k and the dual slack variables s_k are always kept strictly positive. The serial code (`PCx` [2]) on which we based our implementation follows a variant proposed by Sanjay Mehrotra [5] that has proven itself to be very successful in practice. The overall search direction in each iteration is a combination of the so–called affine scaling direction and a corrector step towards the interior of the positive orthant. The affine scaling (or predictor) step $(\Delta x^{\mathrm{aff}}, \Delta\pi^{\mathrm{aff}}, \Delta s^{\mathrm{aff}})$ is obtained by solving the system

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\pi \\ \Delta s \end{pmatrix} = - \begin{pmatrix} r_b \\ r_c \\ r_{xs} \end{pmatrix}$$

($r_b = Ax_k - b$ and $r_c = A^T\pi_k + s_k$ are the current residuals of the primal and dual equality constraints and $r_{xs} = X_kS_ke$ is a vector of current complementarity gaps) and finding the maximal stepsize that keeps x and s nonnegative if this step were to be taken. This step corresponds exactly to a (truncated) Newton step for the zero–finding problem (1); it will get the iterate close to the boundary. The solution of the system (corrector & centering step)

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S_k & 0 & X_k \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\pi \\ \Delta s \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ r_{xs} \end{pmatrix}$$

($r_{xs} = \Delta X^{\mathrm{aff}}\Delta S^{\mathrm{aff}}e - \sigma\tau e$, $\tau = n^{-1}x^Ts$ and σ is a heuristically chosen centering parameter, see [2]) keeps the residuals of the equality constraints at the same level while moving back towards the "central path" by readjusting the complementarity gaps x_is_i.

The overall search direction is then obtained by simply adding the two directions and taking a fraction of the maximum step towards the boundary. Notice that these two systems only differ in the right–hand side.

3. Linear Algebra and Parallelism

Since the matrix A is generally sparse, the coefficient matrix in this system also is sparse and highly structured. Simple block elimination on the system matrix yields the so–called "augmented system"

$$\begin{pmatrix} -D^2 & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\pi \end{pmatrix} = - \begin{pmatrix} r_c + X^{-1}r_{xs} \\ r_b \end{pmatrix}$$

(with $D = XS^{-1} > 0$) which is indefinite and numerically tricky to solve. A further elimination step yields the so–called "normal equations"

$$AD^2A^T\Delta\pi = -r_b + AD^2(-r_c + X^{-1}r_{xs}).$$

Provided that A has full row rank – a standard assumption in linear programming that can be enforced by preprocessing – this system is symmetric positive definite and can be solved directly via a Cholesky factorization. Since the two systems only differ in their right–hand sides, we can use the computed factor for the solution of both systems.

There are two drawbacks of these normal equations that we have to address. First, a single dense column in A will make the whole system completely dense. We can take care of (a reasonably small number of) dense columns quite effectively via the Sherman–Morrison–Woodbury formula at the expense of a few triangular solves per dense column. The second issue is that as we approach a solution, some components of x and s are likely to be zero which in turn will cause the system to become increasingly singular. An effective remedy for this is a modified Cholesky factorization which replaces small pivot elements by a large number. This approach has the advantage that the nonzero structure of the system does not change over the iterations, which implies that the symbolic phase of the factorization (incl. finding an ordering) has to be performed only once.

Before moving on to discuss sources of parallelism, we give a rough overview of the whole algorithm:

```
1. presolve
2. find ordering
3. symbolic factorization of $AA^T$
4. find initial point
repeat
    5. form $AD^2A^T$
    6. perform numerical factorization
    7. solve for affine scaling step
    8. compute centering parameter $\sigma$
    9. solve for corrector step
    10. compute steplength
until convergence
```

Presolving is a heuristic step which eliminates redundancies in the problem (e.g., row dependencies in the constraint matrix are eliminated, some variables might be fixed at one of their bounds if the other constraints imply so etc.). It is not clear how to parallelize this step efficiently just yet.

When a positive definite system is solved on a single processor, the obvious objective of permuting the matrix before factoring is to minimize the fill–in incurred in the Cholesky factor. This problem is known to be NP–hard; the most successful and widely used heuristic approaches are variants of Liu's Multiple Minimum Degree Ordering [4]. In a parallel setting, the ordering has another effect: it will determine the workload for each processor. A good ordering for a parallel factorization will have to balance these two objectives. Recently, permutations derived from graph partitioning ideas, known as nested dissection orderings, have been investigated (e.g., [6]) and are naturally much more successful in balancing the load. Since the ordering is done only once and its runtime is negligible in comparison, we can afford to spend some time with it. In practice, hybrid strategies are used [6].

The obvious source of parallelism in this algorithm is the Cholesky factorization. While the numerical part is performed once per iteration and can be done fully in parallel, the symbolic part is not easily parallelized. We do it on one processor. The triangular solves are performed in parallel. Another nice consequence of the fact that the structure of the systems does not change from iteration to iteration is that the load balancing is done only once, and so the memory for the Cholesky factor can be allocated statically and outside the main loop.

Steps 8 and 10 of the algorithm are minor heuristic computations that are performed on all processors simultaneously. Step 5 is done in parallel in the sense that each processor only forms the portions of the matrix AD^2A^T it will need for the factorization. This leads us from the instruction–level parallelism to data–level parallelism and to a part of our implementation where there is room for more progress towards the goal of solving really large problems: while both the matrix AD^2A^T and the Cholesky factor are kept in distributed form only, as of now we require the constraint matrix to be present on all processors simultaneously. Changing this would require, among other things, an integration of parallel level 2 BLAS operations, a parallel presolver and a parallel ordering.

PDS–20			
size of A	32287×106180		
ordering package	WGPP		
# processors	**1**	**4**	**16**
total running time	9291	4007	1817
av. factorization	142	59.4	24.6
av. Δ–solve	1.62	1.07	1.10
loop time	9056	3874	1730
in % of total	98%	97%	95%
forming AD^2A^T	98.2	44.5	16.3
in % of loop	1.1 %	1.0%	0.9%
pred. & corr. steps	141	173	178
in % of loop	2.6%	4.5%	10.3 %
factorization	8679	3624	1503
in % of loop	96%	94%	87%
# of iterations	60	60	60

Table 1.1: Results for PDS–20, one of the larger Netlib problems.

Numerous heuristics are inherited by `pPCx` from the serial code, and we refer the interested reader to the `PCx` user guide [2] for more details on these and other issues.

4. Computational Results

`pPCx` is implemented entirely in ANSI C with MPI extensions, making it thus a very portable software package. For the parallel solution of the positive definite systems it uses the implementation of a parallel multifrontal Cholesky factorization in the package `psspd` that was developed at the Cornell Theory Center by Chunguang Sun [7]. `psspd` is a self–contained package in the sense that it performs the symbolic and numerical factorization phases as well as the triangular solves. To test the performance of nested dissection orderings (`psspd` provides an implementation of Liu's MMD ordering) we experimented with an implementation by A. Gupta called `WGPP`. The results, as we will see in the next section, are quite interesting.

To give an impression of the performance of `pPCx` we provide some sample results obtained from runs on the IBM SP2 machine at the Cornell Theory Center. We used "thin" nodes roughly equivalent to an RS/6000 model 380 with 64KB data cache, a 64 bit memory bus and 128 MB of main memory. `pPCx` was tested on the larger problems that are contained in the standard LP library at netlib and was always run with the default parameter settings for `PCx`. All times are given in wallclock seconds, the total running time includes pre– and postprocessing, but not the time for reading in the data and converting it to the internal data structures. The "average triangular solve" time is the time required for both a forward– and backward–substitution with the computed Cholesky factor (that is, for *two* triangular solves). The loop time is the total time for the iterative process, that is, for repeated system solutions, line searches and updates.

A typical example run is given in Table 1.1. We see that the numerical factorization is very dominant, and hence the algorithm profits from the speedup of the parallel Cholesky procedure nicely. The triangular substitution does not scale as well and hence we see that with increasing number of processors solving for the predictor and corrector steps becomes more and more significant computationally.

To wrap up, Table 1.2 shows an overview of a few problems that enable us to get an impression of the differences between `WGPP` and `MMD` (for more computational results see [1]).

We see that MMD and WGPP are very comparable in the first 2 problems and that WGPP performs significantly better on the PDS problems. In fact, PDS–20 could not be solved on less than 8 processors using the minimum degree ordering due to memory limitations. The speedup of the factorization is between 3 and 6 (better on larger problems), the overall speedup is up to 5. Note that a significant portion of the computation is still done in serial.

name: **CRE–D**		size: 8926 × 69980		
# procs		**1**	**4**	**16**
total time	MMD	238	109	85.2
	WGPP	262	174	196
avg. num. fact. time	MMD	2.06	0.79	0.60
	WGPP	2.69	1.59	1.38
avg. Δ solve time	MMD	0.16	0.10	0.15
	WGPP	0.18	0.16	0.35

name: **KEN–18**		size: 105127 × 154699		
# procs		**1**	**4**	**16**
total time	MMD	1036	479	355
	WGPP	1610	673	480
avg. num. fact. time	MMD	13.8	4.48	2.02
	WGPP	19.63	5.94	2.64
avg. Δ solve time	MMD	2.19	0.89	0.72
	WGPP	2.42	1.04	0.72

name: **PDS-10**		size: 16558 × 49932		
# procs		**1**	**4**	**16**
total time	MMD	3185	1678	1433
	WGPP	1554	768	399
avg. num. fact. time	MMD	57.0	27.8	21.8
	WGPP	27.6	12.7	4.50
avg. Δ solve time	MMD	0.76	0.69	1.17
	WGPP	0.66	0.56	0.63

name: **PDS-20**		size: 33874 × 105728		
# procs		**1**	**4**	**16**
total time	MMD	*	*	10140
	WGPP	9291	4007	1817
avg. num. fact. time	MMD	*	*	108
	WGPP	142	59.4	24.6
avg. Δ solve time	MMD	*	*	4.81
	WGPP	1.62	1.07	1.10

Table 1.2: Results for a few problems from the Kennington library.

5. Conclusions and Future Directions

There is a need for efficient parallel LP solvers capable of handling very large problems. The code `pPCx`, currently implemented on the IBM–SP multiprocessor system, is a step in this direction. The numerical results presented here indicate that large problem instances can be solved with good efficiency, and they motivate further development.

Several improvements and investigations should be performed to further enhance performance of `pPCx` and its range of applicability. This work includes:

- Distribution of the matrix A among processors. The current implementation requires that A be available on every processor (although the Cholesky factor of AD^2A^T is distributed), a memory limitation that can be severe for very large problems.
- Further investigate ordering strategies, especially the graph partitioning algorithms. A distributed constraint matrix will require the use of a parallel ordering.
- Investigate algorithmic variants that allow concurrent solution of triangular systems with multiple right–hand sides.

This report is based on a talk given by the third author at the Eighth SIAM conference on Parallel Processing for Scientific Computing in Minneapolis, MN, March 1997.

REFERENCES

[1] T. F. Coleman, J. Czyzyk, C. Sun, M. Wagner, and S. J. Wright, *pPCx: Parallel Software for Linear Programming*, Proc. of the Eighth SIAM Conference on Parallel Processing in Scientific Computing, Minneapolis, MN, March 1997

[2] J. Czyzyk, S. Mehrotra, and S. J. Wright, *PCx User Guide*, Technical Report OTC 96/01, Optimization Technology Center, Argonne National Laboratory and Northwestern University, October 1996.

[3] A. Gupta, *WGPP: Watson Graph Partitioning (and sparse matrix ordering) Package*, IBM Research Report RC 20453 (90427), May 1996.

[4] J. W.-H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.

[5] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization, 2 (1992), pp. 575–601.

[6] E. Rothberg and B. Hendrickson, *Sparse matrix ordering methods for interior point linear programming*, Sandia National Laboratories Technical Report 96–0475, January, 1996.

[7] C. Sun, *Efficient parallel solutions of large sparse SPD systems on distributed-memory multiprocessors*, Technical Report CTC92TR102, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY, August 1992.

[8] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM Publications, Philadelphia, PA, 1996.

Chairman's Column

by Jorge J. Moré

This will be my last column as Chair of the SIAM Activity Group on Optimization. I have been fortunate to have a wonderful board to help with the running of the SIAG; I hope that the next Chair will be blessed with the same kind of cooperation.

The SIAG/OPT is alive and healthy. We continue growing; in 1995 the membership was 592, in 1996 we grew to 630, and now we are at 674. We are the fastest-growing SIAG by far, and the second largest SIAG.

We have accomplished a fair amount, but from my point of view four items stand out. The presentation (at the Victoria meeting) of the first SIAM Activity Group on Optimization (SIAG/OPT) Prize is the first item that I wish to highlight. Tim Kelley deserves high marks for chairing the awards committee.

We have also restarted the SIAG/OPT newsletter, *Views & News,* with Juan Meza as the new editor. The first issue looked very good, but each issue brings new challenges. Obtaining good contributions to the newsletter is not easy, so if you have any ideas, please contact Juan at `meza@ca.sandia.gov`. He needs suggestions from the membership!

We also started the SIAG/OPT Web site at `http://www.siam.org/siags/siagop/siagop.htm` We have encouraged members with home pages to register their home pages with SIAM, but only a relatively small number of members (about 100 out of 674) have taken advantage of this offer. If you wish to be listed, send a message to Laura Helfrich at `helfrich@siam.org` with your name and the URL for your Web page.

We would like to update the SIAG/OPT Web page on a regular basis. The aim is to make the

SIAG/OPT Web page an important source of information for the optimization community. I find the list of members and the list of interesting optimization sites quite useful. If you feel that we have not included an interesting site, let me know, and I will make sure that it gets added. Better yet, if you are interested in taking over maintaining the SIAG/OPT Web page, please drop me a note.

Our latest endeavor (April 1997) is an e-mail forum for the exchange of all questions related to optimization technology. You can use this forum for

- technical questions,
- announcements of papers,
- announcements of conferences, books, software, Web pages, and so forth,
- availability of jobs, and
- SIAG/OPT business.

The forum has worked well so far. Most of the messages have been announcements of new technical reports or of SIAG/OPT business. I encourage you to send in job postings, which are of special interest to the student members. I am puzzled about the relatively few messages on technical questions. Perhaps individuals are somewhat afraid of asking a question and sounding ignorant. On the other hand, well-formulated questions can lead to interesting discussions. Ideally, these discussions would be transacted off-line, with the outcome summarized in a message to the forum.

You can use this forum by sending a message to `opt@mailer.siam.org`. If you are a member and have not received any messages, send a message to `owner-opt@mailer.siam.org`. Jim Parker, a SIAM staff member, is the owner of this list. He has been very helpful in maintaining this list and tracking down members without e-mail addresses.

In summary, we have accomplished a lot. More could have been done, but I hope that you all are pleased with what has been done. Note that none of the officers has been indicted and that no special prosecutors have been needed. On the other hand, if we had a budget....

The business meeting at the International Mathematical Programming Symposium was well attended—in spite of stiff competition from a wine-tasting excursion. As expected, most of the discussion revolved around plans for the Sixth SIAM Conference on Optimization, which will be held in Atlanta on May 10–12, 1999. The meeting will be 2.5 days long, with seven plenary talks and roughly ten themes. Phil Gill and Tim Kelley, as co-chairs of the organizing committee, answered questions about the meeting. There was a lively debate on various issues, with the hottest issue being the length of the talks: researchers from outside North America generally felt that longer talks were needed.

The organizing committee is now seeking input from the SIAG membership on themes, minisymposia, plenary speakers, and short courses. Please send your ideas to the committee by Oct. 31.

Attendance at any optimization meeting invariably leads to reflection on progress and future directions of optimization. The issue of progress in optimization arose at a recent technical discussion when the following question was posed: *What problems can we solve today that we could not solve five years ago?*

This is a good question, and I would be interested in answers. For example, we now can reliably and efficiently solve many large nonlinear optimization problems when the user provides only the function. Five years ago we would have needed to use a derivative-free method or to approximate the gradient by differences. Neither approach would have led to a reliable and efficient method for solving large problems. Current automatic differentiation tools can produce the gradients of many partially separable problems accurately and quickly. We have also made significant advances at the algorithmic level, but the claims do not seem to be as strong. Or are they?

What about other areas? What were the important theoretical, algorithmic, or computational advances that were reported in Lausanne? Views on these issues would make interesting contributions to our newsletter, or to the `opt` e-mail forum. They could also be posted in our Web page as success stories.

Essays

I Know It When I See It: Toward a Definition of Direct Search Methods

Michael W. Trosset
Adjunct Associate Professor
Department of Computational & Applied Mathematics
Rice University, Houston, TX
(email: *trosset@caam.rice.edu*)

1. Introduction

Wright [18] has remarked that direct search methods presently enjoy a new respectability in the numerical optimization community. However, as we will endeavor to demonstrate, there is considerable ambiguity about precisely what constitutes a direct search method. As more and more researchers undertake to study these methods, this ambiguity may trouble some scholars. The status quo reminds us of Justice Potter Stewart's infamous pronouncement on hard-core pornography in *Jacobellis v. Ohio, 1964*: "I shall not today attempt to further define the kind of materials I understand to be embraced within that shorthand definition; and perhaps I could never succeed in doing so. But I know it when I see it."

In this report, we discuss and attempt to remove some of the ambiguities that potentially stigmatize direct search methods. To do so, we propose the following:

Definition 1 *A direct search method for numerical optimization is any algorithm that depends on the objective function only through the ranks of a countable set of function values.*

Thus, in the terminology of Stevens [15], we define direct search methods to be algorithms that utilize only ordinal information about the function values.

The remainder of this report explores the implications of Definition 1. In Section 2. we describe the origin of the phrase "direct search method" and note more recent descriptions offered by two prominent researchers. In Section 3. we consider the use of models in numerical optimization and distinguish between "direct search" and "derivative-free" methods. In Section 4. we consider the relation of direct search methods to notions of simple and sufficient decrease.

2. History

Although the phrase "direct search method" is widely used, it is used somewhat ambiguously and without general agreement as to precisely what it should mean. As far as we have been able to determine, the expression was introduced by Hooke and Jeeves [10], who obviously believed that they were coining a phrase:

> We use the phrase "direct search" to describe sequential examination of trial solutions involving comparison of each trial solution with the "best" obtained up to that time together with a strategy for determining (as a function of earlier results) what the next trial solution will be. The phrase implies our preference, based on experience, for straightforward search strategies which employ no techniques of classical analysis except where there is a demonstrable advantage in doing so.

Although Hooke and Jeeves did attempt a formal definition of direct search, they conceded that "it is not difficult, however, to devise procedures that are not altogether covered by this definition." Other researchers have declined to adopt the Hooke and Jeeves definition; however, two crucial notions that Hooke and Jeeves articulated are of enduring importance.

First, it seems evident that Hooke and Jeeves intended direct search methods to be applicable in situations in which one can do no more than compare objective function values:

> The application of direct search to a problem requires a space of points P which represent possible solutions, together with a means of saying that P_1 is a "better" solution than P_2 (written $P_1 \subset P_2$) for any

> two points in the space. There is presumably a single point P^*, the solution, with the property $P^* \subset P$ for all $P \neq P^*$.

This intent provides the inspiration for Definition 1. It has several implications, the most obvious and best-known of which is the requirement that direct search methods do not use derivative information.

Second, Hooke and Jeeves attempted to distinguish direct search methods from Newton's method and the method of steepest ascent/descent. The precise basis for this distinction is somewhat obscure. At one point it appears to be that direct search methods do not fit second- or first-order polynomials to the objective function; elsewhere Hooke and Jeeves stated that direct search "does not include methods which possess a continuum of states, such as those (e.g., Newton's method and methods of steepest ascent) which rely on the use of such tools as derivatives and power series approximations."

The legacy of Hooke and Jeeves [10] is easily discerned in more recent articles. Torczon [16] stated:

> The direct search methods are characterized by the fact that the decision-making process is based solely on function value information; these algorithms neither require nor estimate, in any direct sense, derivative information to determine a direction of descent.

In her somewhat more extensive commentary on the phrase "direct search method," Wright [18] observed:

> Unfortunately, this term is not precisely defined. Two necessary qualifications are:
>
> - A direct search uses only function values;
> - A direct search method does not 'in its heart' develop an approximate gradient.
>
> The second criterion is of course ill-defined: its intent is primarily to exclude methods such as finite-difference quasi-Newton methods that construct a vector subsequently treated as if it were the gradient, but one could argue that any comparison of function values constitutes development of an approximate gradient. Despite this ambiguity, there is general agreement about the methods that do and do not qualify as direct search methods...

The crucial notion that is common to both the Torczon [16] and Wright [18] passages is that direct search methods do not attempt to use or approximate derivative information. If this notion suffices to describe direct search methods, then direct search methods are identical to derivative-free methods. In the next section we will argue that these terms should not be used synonymously and that Definition 1 provides a useful way of distinguishing between them.

3. To Model or Not To Model?

We begin by examining Wright's [18] intention of excluding finite-difference quasi-Newton methods as direct search methods. Of course, this intention extends the intention of Hooke and Jeeves [10] to exclude Newton's method and the method of steepest descent. But whereas the latter methods can be excluded by the simple requirement that direct search methods may use only function values, finite-difference quasi-Newton methods are not excluded by this requirement. Depending on how one conceptualizes the class of quasi-Newton methods, there are different ways to proceed.

The phrase "quasi-Newton method" is itself somewhat ambiguous, having been used differently by different authors. However, it is universally appreciated that Newton's method uses analytic gradients and Hessians to model the objective function with a second-order Taylor polynomial. Many modifications of this fundamental method are available; the one most commonly designated quasi-Newton is the one that replaces the analytic Hessian with an approximation to it. If one also replaces the analytic gradient with a finite-difference approximation to it, then one obtains a finite-difference quasi-Newton method.

If one considers the essence of Newton's method to be its use of derivative information, i.e. its use of the *Taylor polynomial* to model the objective function, then presumably the essence of finite-difference quasi-Newton methods is their attempt to model derivative information. This is perhaps the predominant perception in the numerical optimization community and it naturally leads to the principle, articulated by Torczon [16] and Wright [18], that direct search methods do not model derivatives. However, if one looks beyond Taylor's series approximations and considers the essence of Newton's method to be its use of a quadratic model of the objective function, then the essence of finite-difference quasi-Newton methods is that they too use quadratic models of the objective function. This perception has also been articulated in the numerical optimization literature, perhaps most notably by Dennis and Schnabel [6]. (We also note that Gill, Murray, and Wright [8] catalogued finite-difference quasi-Newton methods under the heading *Non-Derivative Quasi-Newton Methods.*) It leads to a principle that is somewhat closer to the original spirit of Hooke and Jeeves [10], viz. that direct search methods do not model the objective function.

The principle that direct search methods do not model the objective function evidently leads to a more narrow characterization of direct search methods than does the principle that direct search methods do not model derivatives. It has the virtue of distinguishing between direct search methods, which do not model the objective function, and derivative-free methods, which do not model derivatives. Since derivative information is inevitably used to model the objective function, we may then view direct search methods as a subset of derivative-free methods.

We observe that the complement of direct search methods in the set of derivative-free methods, i.e. methods that model the objective function without recourse to derivative information, is an interesting and important class of algorithms. For optimization in the presence of noise, methods that model the objective function by quadratic regression include Box and Wilson [1] and their legacy of response surface methodology, Glad and Goldstein [9], and Elster and Neumaier [7]. For numerical optimization, similar methods that model the objective function without recourse to derivative information include Buckley and Ma [2], Powell [12, 13, 14], and Conn and Toint [5] and Conn, Scheinberg and Toint [3]. A recent survey of derivative-free methods for unconstrained numerical optimization was undertaken by Conn, Scheinberg and Toint [4].

Unfortunately, the principle that direct search methods do not model the objective function brings us no closer to a formal definition of direct search methods than Torczon [16] or Wright [18] because, as the latter author observed, it is the term "model" that is ambiguous. What we need is a formal definition that implicitly defines what it might mean to model the objective function. This need is met by Definition 1, which states that direct search methods may compare function values but that they do not have access to quantitative differences in function values. The restriction to ordinal information about the function values precludes constructing any of the usual kinds of models, e.g. polynomials, without necessitating quibbling about exactly what it means to construct a model.

4. Theories of Convergence

Given a current iterate x_c and a trial iterate x_t, an algorithm for numerical optimization must decide whether to accept $(x_+ = x_t)$ or reject $(x_+ = x_c)$ the trial iterate. This decision is based on the objective function values, $f(x_c)$ and $f(x_t)$, and the precise formulation of the decision rule plays a crucial role in establishing convergence of the algorithm.

Torczon [17] emphasized the distinction between decision rules based on *simple decrease* ($x_+ = x_t$ if $f(x_t) < f(x_c)$) and decision rules based on *sufficient decrease* ($x_+ = x_c$ if $f(x_c) - f(x_t) < \epsilon$ for some $\epsilon > 0$), e.g. the Armijo-Goldstein-Wolfe conditions. Most of the convergence theory developed by the numerical optimization community is derived from enforcing sufficient decrease; however, classical direct search methods such as the simplex algorithm of Nelder and Mead [11] are based on simple decrease. This poses a dilemma: one can either develop theoretical tools based on simple decrease that can be applied to existing methods or one can modify existing methods in order to apply theoretical tools based on sufficient decrease. Regardless of whether or not the modified method is superior to the original method, there is evident danger in the latter approach: if successful, it is enormously tempting to claim that convergence has been demonstrated for the *original* method.

In the present context, how can one decide if algorithmic modifications are benign or fundamental? Definition 1 provides an answer. By denying access to quantitative differences in function values, we insist that convergence theory for direct search methods be based on simple decrease. The litmus test is as follows: if the modified algorithm cannot be used in case only ordinal information about function values is available, then the modified algorithm is not a direct search method and convergence of the original algorithm cannot be claimed. As convergence claims proliferate in the literature, thoughtful readers will do well to keep this test in mind.

Acknowledgments

This work originated in many stimulating conversations with Virginia Torczon, who has profoundly influenced my thinking about direct search methods. Juan Meza and Virginia Torczon read an earlier version of this report and made many helpful suggestions.

REFERENCES

[1] G. E. P. Box and K. B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society, Series B*, 13:1–45, 1951. Includes discussion.

[2] A. G. Buckley and H. Ma. *A Derivative-Free Algorithm for Parallel and Sequential Optimization.* Technical Report, Computer Science Department, University of Victoria, Victoria, Canada, 1994.

[3] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108, CUP, 1997.

[4] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 1997. To appear.

[5] A. R. Conn and Ph. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In G. Di Pillo and F. Giannessi, editors, *Nonlinear Optimization and Applications*, pages 27–47, Plenum Publishing, New York, 1996.

[6] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[7] C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA Journal of Numerical Analysis*, 15:585–608, 1995.

[8] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, New York, 1981.

[9] T. Glad and A. Goldstein. Optimization of functions whose values are subject to small errors. *BIT*, 17:160–169, 1977.

[10] R. Hooke and T. A. Jeeves. "Direct search" solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8:212–229, 1961.

[11] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[12] M. J. D. Powell. A direct search optimization method that models the objective function by linear interpolation. In *Advances in Optimization and Numerical Analysis*, pages 51–67, Kluwer Academic, Dordrecht, Netherlands, 1994. Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico.

[13] M. J. D. Powell. A direct search optimization method that models the objective function by quadratic interpolation. 1994. Presentation at the Fifth Stockholm Optimization Days.

[14] M. J. D. Powell. Trust region methods that employ quadratic interpolation to model the objective function. 1996. Presentation at the Fifth SIAM Conference on Optimization, Victoria, British Columbia.

[15] S. S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.

[16] V. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines.* Technical Report 90-7, Department of Computational and Applied Mathematics, Rice University, Houston, TX, 1990. Author's 1989 Ph.D. dissertation.

[17] V. Torczon. On the convergence of pattern search methods. *SIAM Journal on Optimization*, 7:1–26, 1997.

[18] M. H. Wright. *Direct Search Methods: Once Scorned, Now Respectable.* Technical Report 96-4-02, Computing Science Research Center, AT&T Bell Laboratories, Murray Hill, NJ, 1996.