COMPUTATIONAL ISSUES IN HIGH PERFORMANCE SOFTWARE FOR NONLINEAR OPTIMIZATION

Edited by

Almerico Murli
and
Gerardo Toraldo

Kluwer Academic Publishers

COMPUTATIONAL ISSUES IN HIGH PERFORMANCE SOFTWARE FOR NONLINEAR OPTIMIZATION

edited by

Almerico Murli
Gerardo Toraldo
Università di Napoli "Federico II"

A Special Issue of
COMPUTATIONAL OPTIMIZATION AND APPLICATIONS
An International Journal
Volume 7, No. 1 (1997)

KLUWER ACADEMIC PUBLISHERS
Boston / Dordrecht / London

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

Printed on acid-free paper.

Printed in the United States of America

COMPUTATIONAL OPTIMIZATION AND APPLICATIONS

An International Journal

Volume 7, No. 1, January 1997

Special Issue on: Computational Issues in High Performance Software for Nonlinear Optimization
Guest Editors: Almerico Murli and Gerardo Toraldo

Computational Optimzation and Applications, 7, 1–2 (1997)

Introduction

A. MURLI AND G. TORALDO {murli,toraldo}@matna2.dma.unina.it
Center for Research on Parallel Computing and Supercomputers (CPS), Italian National Research Council & University of Naples "Federico II" Italy

This special issue of *Computational Optimization and Applications* is devoted to a selection of papers from the conference "High Performance Software for Nonlinear Optimization: Status and Perspectives," held on June 22–23, 1995, in Capri, Italy. The papers provide a dynamic overview of some of the recent developments related to software for nonlinear optimization.

The conference was organized by the Center for Research on Parallel Computing and Supercomputers (CPS), a recently established research center whose aim is to promote research in the field of high-performance computing through the development of efficient parallel algorithms and software. The conference was also co-sponsored by the CRAY Research Inc., the Dipartimento di Matematica e Applicazioni "R. Caccioppoli" of the University of Naples "Federico II," the GNIM (Gruppo Nazionale per l'Informatica Matematica), the IBM Semea and the SIAM Activity Group on Optimization.

High-quality software is a blend of several ingredients, whose distinctive nature requires contributions from various areas. Thus, the aim of the conference was to supply an overview from different (and complementary) standpoints of the field of software for nonlinear optimization.

The articles in this issue reflect these different standpoints and provide stimulating insights in several directions. For example, there are two papers – Conn, Gould, and Toint; Lucidi and Roma – that provide computational results and a deeper theoretical understanding of well-established and efficient algorithms for nonlinear problems. Murray presents a comprehensive survey article about classical methods, and Mavridou and Pardalos provide new and exciting applications problems.

Fast algorithms for large-scale optimization problems raise several challenging issues. Yang and Zenios describe the relationship between fast algorithms and parallelism, while Bouaricha and Moré discuss the importance of partial separability. The latter work also emphasizes the importance of providing a "software environment" to ensure that the products of the optimization community are widely applied in diverse fields.

The paper of Matsoms provides an overview of recent research on direct methods for the solution of sparse linear algebra problems – the main computational kernel for many nonlinear optimization codes. Finally, the paper by Billups, Dirkse and Ferris on LCP solvers, gives a comprehensive overview of one of the most interesting optimization problems from the applications.

We thank the authors for their contributions and the referees for their careful comments. Special thanks go to Prof. Jorge J. Moré and to Prof. Panos Pardalos for their contributions as scientific committee members of the conference, to Dr. Maria L. De Cesare and Dr. Maria R. Maddalena from the organizing committee, and to Prof. Hager for publishing this special issue of *Computational Optimization and Applications* devoted to the HPSNO

Conference. The spirit of a journal such as *Computational Optimization and Applications*, which so strongly encourages the development of high-quality software, fits the aim of CPS; we look forward to fruitful collaboration between CPS and *Computational Optimization and Applications* in the future.

Finally, we thank Diego, Ida, Mariella, Marilú, and Teddy for their invaluable and generously offered help, which ensured the successful organization of the HPSNO 95 Conference.

Computational Optimization and Applications, 7, 3–25 (1997)

A Comparison of Large Scale Mixed Complementarity Problem Solvers*

STEPHEN C. BILLUPS sbillups@carbon.cudenver.edu
Mathematics Department, University of Colorado, Denver, Colorado 80217

STEVEN P. DIRKSE steve@gams.com
GAMS Development Corporation, Washington, DC 20007

MICHAEL C. FERRIS ferris@cs.wisc.edu
Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

Abstract. This paper provides a means for comparing various computer codes for solving large scale mixed complementarity problems. We discuss inadequacies in how solvers are currently compared, and present a testing environment that addresses these inadequacies. This testing environment consists of a library of test problems, along with GAMS and MATLAB interfaces that allow these problems to be easily accessed. The environment is intended for use as a tool by other researchers to better understand both their algorithms and their implementations, and to direct research toward problem classes that are currently the most challenging. As an initial benchmark, eight different algorithm implementations for large scale mixed complementarity problems are briefly described and tested with default parameter settings using the new testing environment.

Keywords: complementarity problems, variational inequalities, computation, algorithms

1. Introduction

In recent years, a considerable number of new algorithms have been developed for solving large scale mixed complementarity problems. Many of these algorithms appear very promising theoretically, but it is difficult to understand how well they will work in practice. Indeed, many of the papers describing these algorithms are primarily theoretical papers and include only very minimal computational results. Even with extensive testing, there are inadequacies in the way the results are reported, which makes it difficult to compare one approach against another.

The purpose of this paper is to describe a testing environment for evaluating the strengths and weaknesses of various codes for solving large scale mixed complementarity problems. We believe that the environment is ideally suited for the computational study, development, and comparison of algorithm implementations. The careful description and documentation of the environment given here should help algorithm designers focus their developmental efforts toward practical and useful codes. To exhibit its intended usage, we benchmark eight different algorithm implementations for large scale mixed complementarity problems with the new testing environment. At the same time, we intend to provide a convenient mechanism for modelers to provide new and challenging problems for use in solver comparison.

* This material is based on research supported by National Science Foundation Grant CCR-9157632 and the Air Force Office of Scientific Research Grant F49620-94-1-0036.

As an added benefit, we believe the environment will help modelers determine which code best fits their needs.

The mixed complementarity problem (MCP) is a generalization of a system of nonlinear equations and is completely determined by a nonlinear function $F : R^n \to R^n$ and upper and lower bounds on the variables. The variables z must lie between the given bounds ℓ and u. The constraints on the nonlinear function are determined by the bounds on the variables in the following manner:

$$\begin{aligned} \ell_i < z_i < u_i &\Rightarrow F_i(z) = 0 \\ z_i = \ell_i &\Rightarrow F_i(z) \geq 0 \\ z_i = u_i &\Rightarrow F_i(z) \leq 0. \end{aligned}$$

We will use the notation B to represent the set $[\ell, u]$.

Several special cases of this formulation are immediately obvious. For example, if $\ell = -\infty$ and $u = +\infty$ then the last two implications are vacuous and MCP is the problem of determining $z \in R^n$ such that $F(z) = 0$.

As another example, the Karush-Kuhn-Tucker conditions for nonlinear programs of the form

$$\begin{aligned} &\min \ f(x) \\ &s.t. \ \ g(x) \leq 0 \end{aligned}$$

are given by

$$\begin{aligned} &\nabla f(x) + \lambda \nabla g(x) = 0. \\ &g(x) \leq 0, \ \lambda \geq 0, \ \lambda^T g(x) = 0. \end{aligned}$$

These are easily recast as an MCP by setting

$$z = \begin{bmatrix} x \\ \lambda \end{bmatrix}, \ F(z) = \begin{bmatrix} \nabla f(x) + \lambda \nabla g(x) \\ -g(x) \end{bmatrix}, \ B = R^n \times R^m_+.$$

Here R^m_+ represents the nonnegative orthant of R^m. Many problems in economic equilibrium theory can be cast as MCPs and an overview of how this is accomplished is given in [31]. Other application areas are detailed in [7, 12]. There has been much recent interest in less traditional applications of the complementarity framework. Some of these are based on the generalized equation literature [28] that reformulates the MCP as $0 \in F(z) + N_B(z)$. Here $N_B(z)$ is the classical normal cone to the set B at the point z defined by

$$N_B(z) := \left\{ y \mid y^T(x - z) \leq 0 \ \forall x \in B \right\},$$

if $z \in B$ and is empty otherwise.

Nonlinear complementarity problems appeared in the literature in [5]. The first algorithms for these problems were based on simplicial labeling techniques originally due to Scarf [32]. Extensions of these algorithms led to fixed point schemes [18, 33]. Newton techniques [8, 22, 30] that are based on successive linearization of the nonlinear problem have proven very useful for solving these problems, although their convergence analysis is less satisfactory

than the fixed point theory. Recent extensions have looked at reformulating the nonlinear complementarity problem as a system of nonsmooth nonlinear equations and solving these using a damped Newton or Gauss-Newton approach [6, 8, 10, 11, 13, 16, 19, 20, 21, 23, 24, 25, 26, 27, 29, 34, 35].

We are concerned in this paper with computational testing and comparison of such algorithms. We see several problems with the current state of affairs in the way solvers are developed and compared.

1. Codes are tweaked to solve particular problems, with different choices of control parameters being used to solve different problems. This is contrary to how solvers are used in practice. In general, modelers are not interested in parameter adjustment; instead, they usually run codes only with default options. A good code will have a set of default parameters that performs well on most problems.

2. Even when a consistent set of control parameters is used, codes are developed and tuned using the same limited set of test problems for which computational results are reported. Consequently, the results do not give a fair picture of how the codes might behave on other problems. Enlarging the test suite and adding real world problems alleviates some of these difficulties.

3. There is no clear understanding of what makes problems difficult. Thus, test cases reported do not necessarily reflect the various difficulties that can cause algorithms to fail. As a result, it is extremely difficult for a modeler to determine which algorithm will work best for his particular class of problems.

4. The majority of papers written are theoretical in nature and provide computational results only for naive implementations of the algorithms. While this can exhibit the potential of a particular approach, it is inadequate for evaluating how an algorithm will work in practice. Instead, computational results need to be reported for sophisticated implementations of the algorithms. In particular, algorithm specific scaling, preprocessing or heuristics are crucial for improved robustness and developer supplied *default settings* should be used in all solver comparisons.

5. Test problems do not reflect the interests of users with real-world applications. Thus, algorithms are developed which are good at solving "toy" problems, but are not necessarily good at solving problems of practical importance.

These problems in the way solvers are currently tested result in two major deficiencies in the usefulness of test results. First, the reported results are inadequate for modelers to determine which codes will be most successful for solving their problems. Second, it is difficult for algorithm developers to determine where additional research needs to be directed.

In order to overcome these difficulties, this paper proposes that a testing environment for large scale mixed complementarity problems be developed. The goals of this environment are again twofold: first, it should provide a means of more accurately evaluating the strengths and weaknesses of various codes, and second, it should help direct algorithm developers toward addressing the issues of greatest importance. A preliminary version of such an

environment is described in Section 2 and was used to generate the computational results reported in Section 4. A brief description of each of the codes tested is provided in Section 3.

2. Testing Environment

This section describes a testing environment that aims to correct many of the problems discussed in the introduction concerning how codes are developed and tested. This environment has four main components: a library of test problems, GAMS and MATLAB interfaces that allow these problems to be easily accessed, a tool for verifying the correctness of solutions, and some awk scripts for evaluating results.

2.1. Test Library

The centerpiece of the testing environment is a large publicly available library of test problems that reflects the interests of users with real-world applications, and that also includes problems having known types of computational difficulties. Many of these problems are contained in the standard GAMS distribution [3], while others are part of the expanding collection of problems called MCPLIB[7]. All of the problems that are used in this work are publicly available and can be accessed both from within the GAMS modeling system [3] and from within MATLAB[14].

Because most of the problems in the test library come from real-world applications, the library reflects, as much as possible, the needs of the user community. As this library has become more popular among code developers, we have observed an increased interest among modelers to contribute more and more challenging problems to the library. The motivation is simple: modelers want to encourage the development of codes capable of solving their most difficult problems.

We note that many of the problems contained in the test library are difficult for varying reasons. We believe that it is important to identify the characteristics that make problems hard. This is a daunting task; toward this end, we give an incomplete classification of the types of problem difficulties that may prove challenging for different algorithms.

1. Nonlinearity. We characterize the nonlinearity of a problem by how well a local linearization of the function models the original problem. One difficulty encountered in highly nonlinear problems is the presence of local minima of the underlying merit function that do not correspond to solutions. Several algorithms include features that allow them to escape such local minima, for example, perturbational schemes and nonmonotone watchdog procedures. Thus, we expect that certain algorithms will be more sensitive to the degree of nonlinearity than others.

2. Active Set Determination. For many problems, once the active set is determined, (that is, once we determine which variables are at their upper and lower bounds) the given algorithm is quick to converge. Thus, quick identification of the active set can greatly improve the performance of the algorithm. This seems to be particularly true for problems that are nearly linear.

3. Problem Size. Some algorithms may be better at exploiting problem structure than others, making them less sensitive to the size of the problem. One weakness of our current test suite is that it does not address the issue of size very well. We have attempted to include problems of reasonable size, but it is clear that the test library needs to be expanded in this area.

4. Sensitivity to Scaling. Our experience is that modelers, of necessity, tend to become very good at scaling their models so that relevant matrices are reasonably well-conditioned. Indeed, most of the problems in our model library are well scaled. However, models under development are often poorly scaled. Frequently, solutions are used to scale models properly and to aid in the model construction. Thus, sensitivity to scaling is quite important. In general it is very difficult to scale highly nonlinear functions effectively, so that an algorithm that is less sensitive to scaling may prove to be more practical for highly nonlinear problems.

5. Others. Several other problem characteristics have been proposed, but have not been well studied in the context of real models. These include monotonicity, multiple solutions, and singularity at the solution.

Tables 1 and 2 describe the problems that are included in the test library. Further documentation on these problems can be found in [31] and [7] respectively. Since the starting point can greatly influence the performance of an algorithm, the library includes multiple starting points for most problems. We note that many of the economic problems have the first starting point very close to a solution. This is the "calibration" point and is used by a modeler to test whether the model reproduces benchmark data. The following abbreviations are used when referring to the type of the problem:

MCP	General mixed complementarity problem
LMCP	Linear mixed complementarity problem
NCP	Nonlinear complementarity problem
LCP	Linear complementarity problem
MPSGE	General economic equilibrium problems defined with the MPSGE macro language
NE	Nonlinear equations
NLP	Optimality conditions of a nonlinear program

The tables also include a column labeled "other". In this column we have added some known characteristics of the problems. Thus "M" is entered in this column if the problem is known to be monotone. Similarly a digit "4" for example indicates the number of known solutions. If an "S" occurs in this column then the submatrix of the Jacobian corresponding to the "active constraints" is known to have condition number greater than 10^8 at a solution. The fact that one of these entries does not appear in the table only signifies that the authors do not know whether the problem has this particular characteristic.

Table 1. GAMSLIB Models

Model	Type	n	nnz	density	other
cafemge	MPSGE	101	900	8.82%	
cammcp	NCP	242	1621	2.77%	
cammge	MPSGE	128	1227	7.49%	
cirimge	MPSGE	9	33	40.74%	
co2mge	MPSGE	208	1463	3.38%	
dmcmge	MPSGE	170	1594	5.52%	
ers82mcp	MCP	232	1552	2.88%	
etamge	MPSGE	114	848	6.53%	
finmge	MPSGE	153	1915	8.18%	
gemmcp	MCP	262	2793	4.07%	
gemmge	MPSGE	178	3441	10.86%	
hansmcp	NCP	43	398	21.53%	
hansmge	MPSGE	43	503	27.20%	
harkmcp	NCP	32	131	12.79%	
harmge	MPSGE	11	60	49.59%	
kehomge	MPSGE	9	75	92.59%	3
kormcp	MCP	78	423	6.95%	
mr5mcp	NCP	350	1687	1.38%	
nsmge	MPSGE	212	1408	3.13%	
oligomcp	NCP	6	21	58.33%	
sammge	MPSGE	23	117	22.12%	
scarfmcp	NCP	18	150	46.30%	
scarfmge	MPSGE	18	181	55.86%	
shovmge	MPSGE	51	375	14.42%	
threemge	MPSGE	9	77	95.06%	
transmcp	LCP	11	34	28.10%	
two3mcp	NCP	6	29	80.56%	
unstmge	MPSGE	5	25	100.00%	
vonthmcp	NCP	125	760	4.86%	S
vonthmge	MPSGE	80	594	9.28%	
wallmcp	NE	6	25	69.44%	

2.2. *Interfaces*

To make the test library useful, two interfaces are provided that make the problems easily accessible both for testing of mature codes and for evaluating prototype algorithms.

The first interface is a means for programs to communicate directly with the GAMS modeling language [3]. For realistic application problems, we believe that the use of a modeling system such as AMPL[17] or GAMS is crucial. In earlier work with Rutherford [9], we developed the GAMS/CPLIB interface that provides simple routines to obtain function and Jacobian evaluations and recover problem data. This makes it easy to hook up any solver that is written in Fortran or C as a subsystem of GAMS. The advantages of using a modeling system are many; some of the most important advantages include automatic differentiation, easy data handling, architecture-independent interfaces between models and solvers, and the ability to extend models easily to answer new questions arising from solutions of current models. In addition, modeling languages provide a ready library of examples on which to test solvers. GAMS was chosen for our work instead of AMPL

Table 2. MCPLIB Models

Model	Type	n	nnz	density	other
bertsekas	NCP	15	74	32.89%	
billups	NCP	1	1	100.00%	1
bert_oc	LMCP	5000	21991	0.09%	M
bratu	NLP	5625	33749	0.11%	
choi	NCP	13	169	100.00%	
colvdual	NLP	20	168	42.00%	
colvnlp	NLP	15	113	50.22%	
cycle	LCP	1	1	100.00%	M1
ehl_kost	MCP	101	10201	100.00%	
explcp	LCP	16	152	59.38%	1
freebert	MCP	15	74	32.89%	
gafni	MCP	5	25	100.00%	
hanskoop	NCP	14	129	65.82%	
hydroc06	NE	29	222	26.40%	
hydroc20	NE	99	838	8.55%	
josephy	NCP	4	16	100.00%	1
kojshin	NCP	4	16	100.00%	2
mathinum	NCP	3	9	100.00%	
mathisum	NCP	4	14	87.50%	
methan08	NE	31	225	23.41%	
nash	MCP	10	100	100.00%	
obstacle	LMCP/NLP	2500	14999	0.24%	M1
opt_cont	LMCP	288	4928	5.94%	M1
pgvon105	NCP	105	796	7.22%	S
pgvon106	NCP	106	898	7.99%	S
pies	MCP	42	183	10.37%	
powell	NLP	16	203	79.30%	S
powell_mcp	NCP	8	54	84.38%	
scarfanum	NCP	13	98	57.99%	
scarfasum	NCP	14	109	55.61%	
scarfbnum	NCP	39	361	23.73%	
scarfbsum	NCP	40	614	38.38%	
sppe	NCP	27	110	15.09%	
tobin	NCP	42	243	13.78%	

because it is a mature product with many users, resulting in the availability of many real-world problems.

While we believe that any mature code should be connected with a modeling language, we also feel that there should be an easier means for making the library of test problems available to prototype algorithms. The MATLAB interface described in [14] provides such a means. Using MATLAB, it is possible to quickly implement a prototype version of a new algorithm, which can be tested on the entire suite of test problems with the MATLAB interface. Thus, the test library can play an active role in influencing the development of new algorithms. It must be noted, however, that there are subtle differences between the MATLAB models and the GAMS models. In particular, many GAMS models vary not only the starting point for different runs, but also some of the underlying nonlinearities, whereas the MATLAB models vary only the starting point. Thus, a completely accurate comparison must be carried out exclusively in GAMS or exclusively in MATLAB.

2.3. *Verification of Solutions*

Since stopping criteria vary from algorithm to algorithm, a standardized measure is needed to ensure that different algorithms produce solutions that have some uniformity in solution quality. To achieve this goal, we developed an additional solver, accessible through GAMS, that evaluates the starting point and returns the value of the following merit function:

$$\left\| F(\pi_B(x)) + x - \pi_B(x) \right\|_2, \tag{1}$$

where π_B represents the projection operator onto the set B. To use this verification test, we first solve the problem with the algorithm we are testing, and pass the solution to our "special" solver to verify that the standardized residual is not too large. Since the special solver is callable from GAMS, this can be achieved by adding a few lines to the GAMS problem files.

2.4. *Data Extraction*

The output of MCP codes is typically quite extensive and varies from solver to solver. To extract pertinent information from this output, we have written several awk scripts that read through the files, and then generate data tables. These scripts require slight modifications for each solver, but are a tremendous help in extracting data to produce meaningful information.

3. Description of Algorithms

Ideally, the computational study of algorithms should be performed using only mature, sophisticated codes, so that the strengths and limitations of each algorithm would be accurately reflected in the numerical results. Unfortunately, many of the algorithms proposed for complementarity problems are not accompanied by such mature codes. Of the algorithms described below, the implementations of MILES, PATH, and SMOOTH are the most mature. For the remaining algorithms, we have developed our own implementations which incorporate the GAMS interface.

All of the algorithms outlined have been coded to take explicit advantage of the MCP structure; however, several of them were originally devised for the special case of the nonlinear complementarity problem (NCP)

$$z \geq 0, F(z) \geq 0, z^T F(z) = 0$$

and will be described below in this context. We now give a brief description of the codes that were tested and indicate pertinent references for further details.

3.1. *MILES*

MILES [30] is an extension of the classical Josephy-Newton method for NCP in which the solution to each linearized subproblem

$$0 \in F(z^k) + \nabla F(z^k)(z - z^k) + N_B(z)$$

is computed via Lemke's almost-complementary pivot algorithm. This Newton point is used to define the Newton direction, which is then used in a damped linesearch. The merit function used measures both the violation in feasibility and in complementarity. MILES also employs a restart procedure in cases where the Newton point cannot be computed due to termination in a secondary ray. Every linearized subproblem is rescaled to equilibrate the elements appearing in the data of the subproblem.

3.2. *PATH*

The PATH solver [8] applies techniques similar to those used in Newton methods for smooth systems to the following reformulation of the MCP

$$0 = F(\pi_B(x)) + x - \pi_B(x).$$

Here π_B represents the projection operator onto the set B, which is in general not differentiable. The algorithm consists of a sequence of major iterations, each consisting of an approximation or linearization step similar to that of MILES, the construction of a *path* to the Newton point (the solution to the approximation), and a possible search of this path. When the Newton point does not exists or the path cannot be entirely constructed, a step along the partially computed path is taken before the problem is relinearized. A nonmonotone watchdog strategy is employed in applying the path search; this helps avoid convergence to local minima of the merit function (1), and keeps the number of function evaluations required as small as possible.

Other computational enhancements employed by PATH are a projected Newton preprocessing phase (used to find an initial point that better corresponds to the optimal active set) and the addition of a diagonal perturbation term to the Jacobian matrix when rank deficiency is detected. The Jacobian elements are also automatically scaled by the algorithm at each major iteration.

3.3. *NE/SQP*

The NE/SQP algorithm [26] is based upon reformulating the NCP as the system of nonsmooth equations

$$0 = H(z) := \min\{z, F(z)\}.$$

In [2] the NE/SQP algorithm is extended to the MCP by using the reformulation

$$0 = H(z) := \min\{z - \ell, \max\{z - u, F(z)\}\} \tag{2}$$

Both algorithms use a Gauss-Newton approach that attempts to minimize

$$\theta(z) := \|H(z)\|^2 \tag{3}$$

to find a zero of H. The nonsmoothness of the equations is handled using directional derivatives of H. Specifically, at each iteration, a search direction is calculated by minimizing a convex quadratic program whose objective function is formed by squaring a linear approximation of H. At points where the derivative is not well defined, the linear approximation is created by choosing a particular element of the subdifferential. Once this direction is determined, an Armijo-type linesearch is used to calculate the step size to be taken along that direction. The advantage of this approach is that the direction finding subproblems are always solvable. This is in contrast to Newton-based approaches, which may fail due to a singular Jacobian matrix, and to PATH and MILES, which determine the search direction by attempting to solve a linear complementarity problem, which may, in fact, be unsolvable.

One weakness of the algorithm is that it is vulnerable to converging to local minima of the merit function θ that are not solutions to the problem. The code uses scaling of the subproblems and enforces a small cushion between the iterates and the boundary of B as suggested in [26].

3.4. *SMOOTH*

The SMOOTH algorithm [4] is based upon reformulating the NCP as a system of nonsmooth equations

$$x = \pi_{R^n_+}(x - F(x)),$$

and then approximately solving a sequence of smooth approximations, which lead to a zero of the nonsmooth system. More precisely, at each iteration, a smooth approximation to the original system is formed where the accuracy of the approximation is determined by the residual of the current point, that is $\left\| x - \pi_{R^n_+}(x - F(x)) \right\|$. The smooth approximation p_α to $\pi_{R^n_+}$ corresponds to an integration of the sigmoid function that is commonly used in machine learning. Applying a single step of Newton's method to this smooth function generates a search direction. The next iterate is then generated by performing an Armijo-type linesearch of the merit function

$$\|x - p_\alpha(x - F(x))\|$$

along this direction. Assuming this new point produces an improved residual, the next iteration is based upon a tighter approximation of the nonsmooth equations.

An initial scaling of the data is used in the code, and the PATH preprocessor is used. However, in SMOOTH, the preprocessor is used to try to solve the MCP instead of merely to identify the active set. If this technique fails, the code is restarted and the smoothing technique is then used to find a solution.

3.5. *QPCOMP*

QPCOMP [2] is an enhancement of the NE/SQP algorithm, which adds a proximal perturbation strategy that allows the iterates to escape local minima of the merit function θ

defined in (3). In essence, the algorithm detects when the iterates appear to be converging to a local minimum, and then approximately solves a sequence of perturbed problems to escape the domain of convergence of that local minimum. The perturbed problems are formed by replacing F with the perturbed function

$$F^{\lambda,\bar{z}} := F(z) + \lambda(z - \bar{z}), \tag{4}$$

where the centering point $\bar{z}$ is generally chosen to be the current iterate, and the perturbation parameter λ is chosen adaptively in a manner that guarantees global convergence to a solution when F is both continuously differentiable and pseudomonotone at a solution. In general, the perturbed function is updated after each iteration. Thus, the perturbed problems are not solved exactly; they are just used to determine the next step.

An important aspect of the algorithm is that F is perturbed only when the iterates are not making good progress toward a zero of the merit function. In particular, during the perturbation strategy, whenever an iterate is encountered where the merit function (of the unperturbed problem) has been sufficiently reduced, the algorithm reverts to solving the unperturbed problem. Thus, near a solution, the algorithm maintains the fast local convergence rates of the underlying NE/SQP algorithm.

We note that NE/SQP is equivalent to QPCOMP without the proximal perturbation strategy. Thus, to test NE/SQP, we simply ran the QPCOMP algorithm with the proximal perturbation strategy turned off.

3.6. PROXI

PROXI [1], like NE/SQP and QPCOMP is based upon reformulating the MCP as the system of nonsmooth equations (2). However, instead of solving this system using a Gauss-Newton approach, PROXI uses a nonsmooth version of Newton's method. Specifically, at each iteration, the search direction is calculated by solving a linear system that approximates H at the current iterate. Again, if H is not differentiable at the current iterate, the linear approximation is created by choosing a particular element of the subdifferential.

Like QPCOMP, PROXI uses a proximal perturbation strategy to allow the iterates to escape local minima of the merit function θ defined in (3). This strategy also allows the algorithm to overcome difficulties resulting from singular Jacobian matrices. In particular, if the Newton equation is unsolvable at a particular iteration, the algorithm simply creates a slightly perturbed problem using (4) with a very small λ. The resulting Newton equation for the perturbed function will then be solvable. This strategy for dealing with unsolvable Newton subproblems is considerably more efficient than the Gauss-Newton approach used by NE/SQP and QPCOMP.

3.7. SEMISMOOTH

SEMISMOOTH [1] is an implementation of an algorithm described in [6]. This algorithm is based upon the function

$$\phi(a,b) = \sqrt{a^2+b^2} - (a+b),$$

which was introduced by [15]. This function has the property that

$$\phi(a,b) = 0 \iff a \geq 0, b \geq 0, ab = 0.$$

Using this function, the NCP is reformulated as the semismooth system of equations

$$0 = \Phi(z),$$

where $\Phi_i(z) := \phi(z_i, F_i(z))$. This reformulation has the nice feature that the natural merit function $\Psi(z) := \|\Phi(z)\|^2$ is continuously differentiable. The SEMISMOOTH algorithm described in [1] extends the approach to the MCP by using the reformulation of MCP given by

$$\Phi_i(z) := \phi(z_i - \ell_i, \phi(u_i - z_i, -F_i(z))).$$

To solve the reformulated system of equations, a generalization of Newton's method is used wherein at each iteration, the search direction d^k is found by solving the system

$$H^k d = -\Phi(z^k),$$

where H^k is an element of the B-subdifferential of Φ. The next point z^{k+1} is then chosen by performing a nonmonotone, Arimijo linesearch along the direction d^k.

3.8. SEMICOMP

SEMICOMP [1] is an enhancement of the SEMISMOOTH algorithm, which, like QPCOMP and PROXI, adds a proximal perturbation strategy to allow iterates to escape local minima of the merit function. The algorithm is identical to SEMISMOOTH except when the iterates stop making satisfactory progress toward a zero of Φ. In this case, the proximal perturbation strategy described for the QPCOMP algorithm is employed to allow the iterates to escape the troublesome region. Specifically, at each iteration, a perturbed function is created by (4), and then the SEMISMOOTH algorithm is used to calculate a new point based on this perturbed function. The perturbed function is then updated and the process repeats. The process continues until a new point is encountered where the merit function is sufficiently smaller than the merit function at any previous point. At this point, the algorithm reverts back to the unperturbed SEMISMOOTH algorithm.

4. Computational Comparison

With the exception of NE/SQP and QPCOMP, each of the eight algorithms described in the previous section was run on all of the problems in the test library from all of the starting points. Since NE/SQP and QPCOMP were implemented using a dense QP code, we only ran the problems with fewer than 110 variables for these solvers. Table A.1 in appendix A

shows the execution time needed by each algorithm on a SPARC 10/51, while Table A.2, also in Appendix A, shows the number of function and Jacobian evaluations required by each algorithm. To abbreviate the results, we excluded any problems that were solved in less than 2 seconds by all of the algorithms we tested.

Each algorithm minimizes its own merit function as described in Section 3 and all were terminated when this measure was reduced below 10^{-6}. Since the merit functions are different for each code, we tested the solutions to ensure that the standardized residual given by (1) was always less than 10^{-5}. It is possible that one more or one less "Newton" step would be carried out if the same merit function was used for every algorithm. Since this is impractical, the method we now outline for reporting our results makes these small changes entirely irrelevant.

How one chooses to summarize data of this nature depends on what one's goals are. From a modeling standpoint, one could determine which models were the most difficult to solve by aggregating results for each model. From a computational standpoint, one can compare the solvers using many different criteria, including number of successes/failures, cumulative solution time required, number of cases where solution time is "acceptable", number of function/gradient evaluations required, etc. As examples of useful metrics, we have chosen the following:

success Success is achieved if a solution is computed.

competitive We say the time T_C for a code C is "competitive" with the time $T_{\min}$ taken by the best code on that run if $T_C \leq 2\,T_{\min}$, and "very competitive" if $T_C \leq \frac{4}{3}\,T_{\min}$.

Tables 3 and 4 summarize our results for two sets of models, the large ones (≥ 110 variables) for which only the sparsity-exploiting solvers were run, and the smaller ones on which all solvers were run.

Table 3. Code Comparisons – Large Models

	MILES	PATH	PROXI	SEMI-COMP	SEMI-SMTH	SMTH
very comp.	37%	65%	50%	22%	17%	59%
competitive	56%	80%	67%	41%	43%	76%
success	83%	98%	89%	91%	86%	98%

Table 4. Code Comparisons – Small Models

	MILES	NE/SQP	PATH	PROXI	QP COMP	SEMI-COMP	SEMI-SMTH	SMTH
very comp.	32%	0%	43%	34%	0%	26%	25%	24%
competitive	45%	2%	67%	54%	1%	44%	40%	52%
success	84%	67%	94%	95%	90%	88%	65%	92%

5. Conclusions

The testing environment we have described addresses many of the problems we have observed about how codes are developed and tested. In particular, with a large collection of test problems available, it is more difficult to tune a code to the test set. Moreover, even if such tuning is successful, the resulting code will be good at solving the types of problems that are represented in the library, namely, the problems that are of interest to the user community. The inclusion of problems with known difficulties allows codes to be compared by how well they solve different classes of problems, thus allowing users to more accurately choose codes that meet their needs. Finally, by categorizing problems with different computational difficulties, the library can be used to highlight the areas where research energies most need to be directed.

Our testing indicates superior performance by the PATH, SMOOTH, and PROXI algorithms. However, as the codes continue to mature, it is possible that their relative performance will change. It is not our intention to declare a winner, but rather to "clarify the rules" so that code developers will focus on the right issues when developing algorithms. To a large extent, we have accomplished this with our testing environment.

It is unfortunate that the scope of our testing could not have been more broad. Some of the algorithms mentioned above were coded by the authors of this paper (not the originators of the algorithm), while there are numerous other algorithms that we were not able to test at all. This is due primarily to the fact that these algorithms do not have GAMS interfaces. It is our hope that as the CPLIB interface becomes more widely known, other code developers will hook up their solvers to GAMS. This will allow their algorithms to be easily compared with other codes using our testing environment.

Lastly, we wish to emphasize that the test library is continually being expanded. In particular, we are always eager to add challenging new real world models to the library. To this end, we have begun to augment the MCPLIB by adding new models that have recently come to our attention. The 10 models listed in Table 5 have been used in various disciplines to answer questions that give rise to complementarity problems. Some of these models are solved from many different starting points, indicated by the "solves" column. The first 6 are economic models, the next two arise from applications in traffic equilibrium and multi-rigid-body contact problems, the final two correspond to complementarity problems for which all solutions are required. The numbers of solutions for the last two problems are known to be odd, the number listed below is a lower bound. These problems appear to be more difficult than most of the problems solved in this paper. Certainly, some are much larger, while others have singularities either at solutions or starting points. Most of these problems do not have underlying monotonicity.

The results that we present in Tables 6 and 7 for these models are somewhat different to the results in Appendix A and are motivated more by the models themselves. For the games and tinloi models, it is important to find all solutions of the model, and so after a fixed number of runs from a variety of starting points, we report the number of distinct solutions found for these models in Table 6.

For the remaining problems, we just report one statistic in Table 7 for each model. If every problem was solved, we report the total resources used to solve the complete model,

Table 5. New Models

Model	Type	n	nnz	density	solves	other
shubik	MCP	33	207	19.01%	48	S
jmu	MCP	2253	10123	0.20%	1	
asean9a	NE	10199	72320	0.07%	1	
eppa	MPSGE	1269	10130	0.63%	8	
uruguay	MPSGE	2281	90206	1.73%	2	
hanson	NE	487	3868	1.63%	2	S
trafelas	MCP	2904	15000	0.18%	2	
lincont	LCP	419	23626	13.46%	1	
games	NCP	16	256	60.94%	25	5
tinloi	LCP	146	5694	26.71%	64	3

Table 6. Distinct Solutions Found

Model	MILES	PATH	PROXI	S/COMP	S/SMTH	SMOOTH
games	3	5	2	3	3	4
tinloi	2	3	1	1	1	2

otherwise we report an error using a letter to signify some sort of failure "F", memory error "M", time limit exceeded "T" or iteration limit exceeded "I". Only the first error is listed per problem, while the numbers in parentheses are the number of problems that failed to solve.

Table 7. Summary for New Models

Model	MILES	PATH	PROXI	S/COMP	S/SMTH	SMOOTH
shubik	I(13)	F(9)	F(25)	I(34)	I(42)	I(15)
jmu	I	110.81	F	F	T	214.32
asean9a	T	62.08	M	92.85	94.3	91.62
eppa	249.61	203.79	M	T(7)	F(7)	239.73
uruguay	I(1)	2760.17	M	M	68161.10	4519.53
hanson	F(1)	39.36	F(1)	F(1)	I(1)	4.94
trafelas	T(2)	150.55	F(1)	T(2)	T(2)	346.23
lincont	9.99	10.76	F	F	T	718.27

It is our intention to add these models and newer models that are brought to our attention to MCPLIB. In this way we hope that the problem library will continue to serve as a guide for code developers so that they will direct their energies into areas that will best serve the users.

Appendix A

Table A.1. Execution Times (sec.)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP- COMP	SEMI- COMP	SEMI- SMTH	SMTH
bert_oc	1	6.15	–	2.63	2.61	–	11.38	13.50	3.23
bert_oc	2	7.07	–	3.13	3.24	–	46.44	54.41	2.57
bert_oc	3	fail	–	2.10	2.78	–	15.52	17.99	2.55
bert_oc	4	136.4	–	2.29	2.67	–	5.80	5.91	2.62
bertsekas	1	0.07	fail	0.08	0.39	2.83	0.64	fail	0.24
bertsekas	2	0.28	fail	0.04	0.27	2.41	0.59	fail	0.05
billups	1	fail	fail	fail	0.02	0.11	0.10	0.90	fail
bratu	1	fail	–	138.52	149.37	–	7452.38	fail	135.48
cafemge	1	0.18	18.16	0.29	0.50	20.11	0.50	0.66	0.41
cafemge	2	0.23	16.57	0.26	0.35	14.19	0.50	0.39	0.25
cammcp	1	0.50	–	0.21	2.89	–	fail	fail	0.23
choi	1	8.13	2.00	2.09	2.03	2.28	2.95	2.93	2.10
co2mge	2	0.43	–	0.50	0.48	–	2.02	2.42	0.52
co2mge	6	0.62	–	0.46	fail	–	fail	fail	1.96
colvdual	1	0.05	fail	0.11	0.25	5.76	0.12	0.10	0.11
colvdual	2	0.07	fail	0.09	0.50	5.39	fail	fail	0.10
colvnlp	1	0.03	fail	0.05	0.09	2.13	0.08	0.09	0.06
colvnlp	2	0.05	fail	0.03	0.05	1.62	0.06	0.05	0.05
dmcmge	1	0.20	–	3.75	fail	–	fail	fail	5.42
dmcmge	2	0.50	–	0.55	fail	–	133.73	fail	0.60
ehl_kost	1	23.58	fail	3.86	18.50	611.41	18.99	15.02	4.73
ehl_kost	2	23.92	248.79	13.56	37.67	250.28	49.06	58.25	12.58
ehl_kost	3	24.15	fail	9.76	64.88	866.08	233.23	240.12	90.38
finmge	2	0.38	–	1.95	11.34	–	fail	fail	5.16
finmge	4	0.48	–	1.72	12.34	–	fail	fail	9.18
freebert	1	0.03	fail	0.07	0.39	2.72	0.51	fail	0.04
freebert	3	0.10	fail	0.05	0.25	2.86	0.55	fail	0.04
freebert	4	fail	fail	0.09	0.31	2.47	0.60	fail	fail
freebert	5	fail	fail	0.04	0.12	1.38	0.15	0.12	0.04
freebert	6	fail	fail	0.08	0.33	3.02	0.53	fail	fail
gemmcp	1	2.12	–	0.21	0.16	–	0.19	0.18	0.24
gemmge	2	0.47	–	3.24	3.31	–	3.31	3.60	4.18
gemmge	3	0.55	–	1.85	1.89	–	2.88	3.92	1.85
gemmge	4	0.52	–	2.51	2.37	–	2.84	3.22	1.84
gemmge	5	0.55	–	8.85	5.00	–	5.32	6.93	2.28
hanskoop	1	0.07	0.37	0.05	0.10	0.37	fail	fail	0.33
hanskoop	2	0.08	0.04	0.06	0.01	0.05	fail	fail	0.02

Table A.1. Execution Times (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
hanskoop	3	fail	0.34	0.11	0.09	0.42	fail	fail	0.23
hanskoop	4	1.10	0.06	0.05	0.01	0.05	fail	fail	0.02
hanskoop	5	0.07	fail	0.09	0.10	0.70	0.07	0.08	0.30
hanskoop	7	0.07	fail	0.05	0.09	0.86	fail	fail	0.22
hanskoop	9	fail	0.50	0.10	0.24	0.43	0.09	0.10	0.23
hansmcp	1	0.10	fail	0.47	0.14	fail	0.16	0.16	0.13
hansmge	1	0.10	3.14	0.36	0.70	2.86	0.84	0.88	0.64
harkmcp	4	0.07	6.96	0.12	0.21	9.31	fail	fail	0.37
harmge	1	0.03	fail	0.06	0.44	1.86	1.52	fail	0.09
harmge	2	0.80	fail	0.03	0.02	0.14	0.01	fail	0.03
harmge	3	0.07	fail	0.04	0.03	0.13	0.02	fail	0.04
harmge	4	0.08	fail	0.05	0.03	0.15	0.01	fail	0.04
harmge	5	0.08	fail	0.05	0.04	0.16	0.02	fail	0.04
harmge	6	fail	fail	0.06	fail	3.24	0.02	fail	2.08
hydroc20	1	fail	16.11	0.38	0.44	13.31	0.54	0.41	0.36
josephy	1	fail	fail	0.03	0.02	0.08	0.02	0.01	0.03
josephy	2	fail	fail	0.04	0.02	0.07	0.02	0.02	0.02
josephy	4	fail	fail	0.02	0.01	0.04	0.01	0.01	0.02
josephy	6	fail	0.04	fail	0.02	0.05	0.01	0.01	0.02
kojshin	1	fail	fail	0.03	0.01	0.07	0.02	0.03	0.03
kojshin	3	0.03	fail	0.06	0.05	0.12	0.06	0.07	0.11
kormcp	1	0.23	2.82	0.08	0.06	2.82	0.07	0.05	0.05
mr5mcp	1	0.60	–	0.62	2.17	–	2.09	2.01	0.62
nsmge	1	0.25	–	0.91	1.64	–	1.69	1.65	2.40
obstacle	1	2.37	–	2.36	3.40	–	6.86	5.59	2.39
obstacle	2	fail	–	5.90	7.33	–	18.01	15.56	6.39
obstacle	3	fail	–	5.03	8.85	–	11.77	9.45	6.27
obstacle	4	3.98	–	4.84	9.29	–	11.01	10.66	6.12
obstacle	5	fail	–	8.04	4.52	–	15.08	14.59	7.13
obstacle	6	fail	–	8.86	9.92	–	19.62	21.14	10.07
obstacle	7	fail	–	7.39	7.57	–	12.84	15.52	7.97
obstacle	8	fail	–	13.84	7.54	–	14.76	14.32	10.58
opt_cont127	1	8.52	–	8.14	9.91	–	46.05	45.58	6.38
opt_cont255	1	fail	–	14.86	18.71	–	107.97	110.61	13.80
opt_cont31	1	2.10	–	1.36	1.51	–	5.55	4.45	1.65
opt_cont511	1	fail	–	39.51	43.19	–	348.63	360.42	37.52
pgvon105	1	fail	fail	1.54	7.99	fail	fail	fail	fail
pgvon105	2	0.42	41.51	0.77	2.18	50.91	fail	fail	fail

Table A.1. Execution Times (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
pgvon105	3	fail	33.47	1.58	52.13	58.80	fail	fail	fail
pgvon105	4	fail	fail	fail	fail	fail	28.09	fail	fail
pgvon106	1	fail	fail	19.77	13.21	fail	fail	fail	125.46
pgvon106	2	fail	fail	1.80	fail	fail	fail	fail	5.37
pgvon106	3	fail	fail	1.29	fail	fail	fail	fail	8.48
pgvon106	4	fail	fail	fail	2.46	fail	38.30	fail	fail
pgvon106	5	5.33	fail	fail	fail	fail	fail	fail	fail
pgvon106	6	fail	fail	fail	fail	fail	fail	fail	3.76
pies	1	0.07	fail	0.13	0.29	7.26	0.11	0.13	0.27
sammge	1	0.07	fail	0.01	0.01	fail	0.00	0.00	0.00
sammge	3	0.10	0.27	0.05	0.16	0.26	0.17	fail	0.18
sammge	5	0.12	0.42	0.07	0.12	0.48	0.36	fail	0.13
sammge	6	0.13	0.45	0.05	0.27	0.58	0.40	fail	0.13
sammge	7	0.18	0.69	0.06	0.13	0.58	0.23	fail	0.20
sammge	8	0.10	0.78	0.05	0.63	0.74	0.39	fail	0.19
sammge	9	0.10	0.71	0.07	0.45	0.69	0.65	fail	0.20
sammge	10	0.17	fail	0.01	0.01	fail	0.01	0.00	0.01
sammge	13	0.05	0.30	0.12	0.20	0.28	0.23	fail	0.23
sammge	14	0.12	0.29	0.11	0.17	0.35	0.23	fail	0.18
sammge	15	0.05	0.27	0.06	0.48	0.31	0.38	fail	0.25
sammge	16	0.05	0.47	0.11	0.26	0.46	0.31	fail	0.10
sammge	17	0.10	0.62	0.09	0.57	1.05	0.20	fail	0.17
sammge	18	0.08	0.37	0.11	0.46	0.45	0.50	fail	0.16
scarfasum	2	0.15	fail	0.04	0.15	1.51	0.15	0.12	0.10
scarfasum	3	0.13	0.29	0.07	0.15	0.37	fail	fail	0.05
scarfbnum	1	0.08	6.27	0.39	0.57	6.42	1.01	fail	0.32
scarfbnum	2	0.10	6.01	0.44	0.43	6.09	7.36	fail	0.32
scarfbsum	1	0.17	fail	fail	0.49	8.77	0.39	0.31	0.24
scarfbsum	2	0.18	fail	3.43	5.16	31.11	1.22	fail	0.66
threemge	7	0.08	–	0.06	fail	–	0.14	0.13	0.05
threemge	8	0.07	–	0.06	fail	–	0.12	0.14	0.05
threemge	11	0.12	–	0.05	fail	–	0.82	fail	0.05
transmcp	1	0.03	fail	0.04	0.09	1.22	0.23	fail	0.05
transmcp	2	0.03	fail	0.01	0.00	fail	0.00	fail	0.00
transmcp	3	0.03	0.02	0.02	0.01	0.02	0.03	fail	0.02
transmcp	4	0.03	0.11	0.02	0.01	0.10	0.04	fail	0.02
vonthmcp	1	fail	–	fail	fail	–	fail	fail	fail
vonthmge	1	0.08	fail	1.06	fail	fail	fail	fail	17.14

Table A.2. Function and Jacobian Evaluations f(j)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
bert_oc	1	13(12)	–	4(4)	4(3)	–	21(11)	21(11)	4(4)
bert_oc	2	21(13)	–	4(4)	4(3)	–	143(42)	143(42)	4(4)
bert_oc	3	fail	–	4(4)	4(3)	–	41(15)	41(15)	4(4)
bertsekas	1	259(36)	fail	27(6)	138(37)	151(44)	251(42)	fail	113(27)
bertsekas	2	5(4)	fail	5(5)	83(31)	126(40)	327(38)	fail	7(7)
bertsekas	3	12(11)	9(8)	12(12)	21(20)	9(8)	181(39)	181(39)	67(24)
billups	1	fail	fail	fail	23(22)	23(22)	631(76)	6903(345)	fail
bratu	1	fail	–	48(26)	48(25)	–	3164(538)	fail	48(26)
cafemge	1	8(6)	16(10)	9(7)	23(9)	16(10)	18(10)	18(10)	9(8)
cafemge	2	6(5)	15(8)	6(6)	17(7)	15(8)	11(8)	11(8)	6(6)
cammcp	1	4(3)	–	4(4)	77(23)	–	fail	fail	4(4)
choi	1	5(4)	5(4)	5(5)	5(4)	5(4)	6(5)	6(5)	5(5)
co2mge	2	9(6)	–	9(7)	7(5)	–	62(15)	63(16)	7(6)
co2mge	6	6(5)	–	7(7)	fail	–	fail	fail	81(13)
colvdual	1	4(3)	fail	15(13)	201(36)	252(78)	44(16)	44(16)	40(15)
colvdual	2	4(3)	fail	16(12)	250(55)	184(59)	fail	fail	52(17)
colvnlp	1	4(3)	fail	10(7)	77(16)	178(54)	46(16)	46(16)	37(14)
colvnlp	2	4(3)	fail	5(5)	29(12)	137(30)	26(15)	26(15)	23(10)
dmcmge	1	99(27)	–	34(18)	fail	–	fail	fail	97(23)
dmcmge	2	13(8)	–	6(6)	fail	–	3099(661)	fail	6(6)
ehl_kost	1	6(5)	fail	6(6)	25(14)	108(105)	32(15)	32(15)	6(6)
ehl_kost	2	8(7)	97(30)	19(19)	95(28)	97(30)	125(34)	125(34)	21(12)
ehl_kost	3	16(11)	fail	11(11)	144(44)	409(79)	671(114)	671(114)	262(55)
finmge	2	5(4)	–	7(7)	151(25)	–	fail	fail	60(13)
finmge	4	5(4)	–	8(8)	135(28)	–	fail	fail	110(20)
freebert	1	4(3)	fail	5(5)	138(37)	151(44)	266(46)	fail	6(6)
freebert	3	4(3)	fail	5(5)	106(35)	173(45)	206(42)	fail	6(6)
freebert	4	fail	fail	27(6)	138(37)	151(44)	240(42)	fail	fail
freebert	5	fail	fail	5(5)	53(14)	116(23)	49(14)	49(14)	5(5)
freebert	6	fail	fail	27(6)	106(35)	173(45)	200(40)	fail	fail
gemmcp	1	2(1)	–	2(2)	2(1)	–	2(1)	2(1)	2(2)
gemmge	1	fail	–	fail	2(1)	–	2(1)	2(1)	2(2)
gemmge	2	7(5)	–	18(7)	22(7)	–	16(9)	16(9)	21(6)
gemmge	3	6(5)	–	6(6)	6(5)	–	10(8)	10(8)	6(6)
gemmge	4	7(6)	–	6(6)	7(6)	–	8(7)	8(7)	6(6)
gemmge	5	10(7)	–	26(21)	25(11)	–	31(13)	31(13)	13(7)
hanskoop	1	6(4)	15(10)	13(7)	42(16)	15(10)	fail	fail	110(36)
hanskoop	2	2(1)	2(1)	14(6)	2(1)	2(1)	fail	fail	2(1)

Table A.2. Function and Jacobian Evaluations (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
hanskoop	3	fail	18(11)	23(14)	44(13)	18(11)	fail	fail	78(31)
hanskoop	4	2(1)	2(1)	14(6)	2(1)	2(1)	fail	fail	2(1)
hanskoop	5	6(5)	fail	19(11)	68(15)	27(11)	20(8)	20(8)	102(34)
hanskoop	7	7(5)	fail	11(6)	37(15)	45(13)	fail	fail	83(25)
hanskoop	9	fail	23(13)	16(13)	187(41)	23(13)	20(15)	20(15)	95(31)
hansmcp	1	6(4)	fail	45(18)	18(9)	fail	24(13)	24(13)	10(8)
hansmge	1	4(3)	11(8)	12(8)	37(15)	11(8)	47(17)	47(17)	26(13)
harkmcp	4	5(4)	29(13)	13(6)	23(14)	27(14)	fail	fail	31(17)
harmge	1	284(60)	fail	11(7)	222(38)	132(57)	672(75)	fail	33(11)
harmge	2	5(4)	fail	5(5)	5(4)	5(4)	3(2)	fail	5(5)
harmge	3	5(4)	fail	5(5)	5(4)	5(4)	3(2)	fail	5(5)
harmge	4	8(5)	fail	8(6)	5(4)	5(4)	3(2)	fail	8(6)
harmge	5	8(5)	fail	8(6)	8(5)	8(5)	3(2)	fail	8(6)
harmge	6	fail	fail	13(8)	fail	379(78)	3(2)	fail	1117(139)
hydroc20	1	fail	10(8)	11(9)	10(8)	10(8)	12(9)	12(9)	10(9)
josephy	1	fail	fail	7(7)	37(14)	13(7)	10(7)	10(7)	24(9)
josephy	2	fail	fail	15(11)	15(7)	15(7)	12(7)	12(7)	9(6)
josephy	4	fail	fail	4(4)	5(4)	5(4)	6(5)	6(5)	5(4)
josephy	6	fail	4(3)	fail	12(6)	12(6)	13(7)	13(7)	9(6)
kojshin	1	fail	fail	6(6)	18(9)	16(7)	22(10)	22(10)	33(10)
kojshin	3	11(10)	fail	17(17)	97(22)	35(10)	92(23)	122(30)	189(27)
kormcp	1	4(3)	4(3)	4(4)	4(3)	4(3)	4(3)	4(3)	4(4)
mr5mcp	1	7(6)	–	7(7)	64(15)	–	26(13)	26(13)	7(7)
nsmge	1	82(17)	–	10(8)	35(14)	–	23(12)	23(12)	44(18)
obstacle	1	50(14)	–	11(11)	11(10)	–	15(14)	15(14)	11(11)
obstacle	2	fail	–	12(12)	12(11)	–	17(14)	17(14)	12(12)
obstacle	3	fail	–	17(11)	21(13)	–	14(13)	14(13)	17(11)
obstacle	4	2(1)	–	12(11)	23(16)	–	17(16)	17(16)	12(11)
obstacle	5	fail	–	7(7)	8(6)	–	8(7)	8(7)	7(7)
obstacle	6	fail	–	16(10)	16(9)	–	20(13)	20(13)	16(10)
obstacle	7	fail	–	12(10)	17(9)	–	17(12)	17(12)	12(10)
obstacle	8	fail	–	17(11)	9(6)	–	10(7)	10(7)	17(11)
opt_cont127	1	8(3)	–	6(6)	6(5)	–	27(12)	27(12)	6(6)
opt_cont255	1	fail	–	6(6)	6(5)	–	31(14)	31(14)	6(6)
opt_cont31	1	2(1)	–	6(6)	5(4)	–	11(9)	11(9)	6(6)
opt_cont511	1	fail	–	6(6)	6(5)	–	73(20)	73(20)	6(6)
pgvon105	1	fail	fail	64(16)	403(75)	fail	fail	fail	fail
pgvon105	2	33(14)	199(39)	27(10)	135(23)	213(30)	fail	fail	fail

Table A.2. Function and Jacobian Evaluations (continued)

Problem Name	st. pt.	MILES	NE/ SQP	PATH	PROXI	QP-COMP	SEMI-COMP	SEMI-SMTH	SMTH
pgvon105	3	fail	153(32)	63(14)	3353(338)	322(40)	fail	fail	fail
pgvon105	4	fail	fail	fail	fail	fail	352(42)	fail	fail
pgvon106	1	fail	fail	772(101)	739(88)	fail	fail	fail	6428(482)
pgvon106	2	fail	fail	48(36)	fail	fail	fail	fail	109(37)
pgvon106	3	fail	fail	39(20)	fail	fail	fail	fail	233(49)
pgvon106	4	fail	fail	fail	86(28)	fail	412(65)	fail	fail
pgvon106	5	47(23)	–	fail	fail	fail	fail	fail	fail
pgvon106	6	fail	–	fail	fail	fail	fail	fail	58(27)
pies	1	3(2)	fail	13(13)	73(23)	54(49)	22(13)	22(13)	41(14)
sammge	1	1(0)	fail	1(1)	1(1)	fail	1(1)	1(1)	1(1)
sammge	3	4(3)	4(3)	6(4)	46(7)	4(3)	41(8)	fail	66(10)
sammge	5	14(6)	11(5)	11(6)	17(8)	11(5)	84(14)	fail	33(11)
sammge	6	4(3)	11(5)	9(5)	52(19)	11(5)	103(17)	fail	25(9)
sammge	7	6(4)	16(7)	5(5)	23(10)	16(7)	54(11)	fail	47(18)
sammge	8	4(3)	9(5)	9(5)	146(38)	9(5)	114(16)	fail	40(11)
sammge	9	17(7)	13(7)	5(5)	139(26)	13(7)	168(29)	fail	50(18)
sammge	10	1(0)	fail	1(1)	1(1)	fail	1(1)	1(1)	1(1)
sammge	13	4(3)	4(3)	17(6)	47(13)	4(3)	61(14)	fail	51(15)
sammge	14	4(3)	4(3)	16(6)	50(13)	4(3)	74(11)	fail	50(14)
sammge	15	4(3)	4(3)	5(5)	83(23)	4(3)	122(20)	fail	76(15)
sammge	16	4(3)	7(5)	8(8)	40(15)	7(5)	80(12)	fail	14(7)
sammge	17	4(3)	24(7)	6(6)	75(22)	43(7)	61(11)	fail	37(10)
sammge	18	4(3)	7(5)	8(8)	131(26)	7(5)	148(22)	fail	33(12)
scarfasum	2	12(7)	fail	5(5)	25(9)	73(26)	23(12)	23(12)	23(6)
scarfasum	3	5(4)	9(6)	8(6)	34(13)	9(6)	fail	fail	9(6)
scarfbnum	1	5(4)	70(20)	24(14)	164(46)	76(21)	241(51)	fail	71(20)
scarfbnum	2	5(4)	97(22)	25(15)	165(35)	58(19)	1497(341)	fail	95(24)
scarfbsum	1	4(3)	fail	462(57)	60(25)	26(22)	37(18)	37(18)	24(11)
scarfbsum	2	4(3)	fail	162(21)	1062(117)	157(83)	276(43)	fail	103(24)
threemge	7	6(5)	–	6(6)	fail	–	32(12)	32(12)	6(6)
threemge	8	6(5)	–	6(6)	fail	–	30(12)	30(12)	6(6)
threemge	11	6(5)	–	6(6)	fail	–	215(26)	fail	6(6)
transmcp	1	2(1)	fail	12(12)	92(26)	69(67)	193(105)	fail	24(15)
transmcp	2	1(0)	fail	1(1)	1(1)	fail	1(1)	fail	1(1)
transmcp	3	2(1)	2(1)	3(3)	3(2)	2(1)	15(8)	fail	4(3)
transmcp	4	6(5)	6(5)	3(3)	3(2)	6(5)	43(10)	fail	5(5)
two3mcp	1	6(5)	16(8)	6(6)	16(8)	16(8)	13(8)	13(8)	13(8)
two3mcp	2	5(4)	7(4)	5(5)	7(4)	7(4)	7(5)	7(5)	5(4)
unstmge	1	10(8)	10(8)	16(15)	11(8)	10(8)	11(9)	11(9)	8(7)
vonthmcp	1	fail	–	fail	fail	–	fail	fail	fail
vonthmge	1	18(14)	fail	34(22)	fail	fail	fail	fail	730(278)

References

1. S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations*. PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, August 1995.
2. S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming* forthcoming, 1996.
3. A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.
4. Chunhui Chen and O. L. Mangasarian. A class of smoothing functions for nonlinear and mixed complementarity problems. *Computational Optimization and Applications*, 5:97–138, 1996.
5. R. W. Cottle. *Nonlinear programs with positively bounded Jacobians*. PhD thesis, Department of Mathematics, University of California, Berkeley, California, 1964.
6. T. De Luca, F. Facchinei, and C. Kanzow. A semismooth equation approach to the solution of nonlinear complementarity problems. Preprint 93, Institute of Applied Mathematics, University of Hamburg, Hamburg, January 1995.
7. S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.
8. S. P. Dirkse and M. C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.
9. S. P. Dirkse, M. C. Ferris, P. V. Preckel, and T. Rutherford. The GAMS callable program library for variational and complementarity solvers. Mathematical Programming Technical Report 94-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.
10. F. Facchinei and J. Soares. A new merit function for nonlinear complementarity problems and a related algorithm. *SIAM Journal on Optimization*, forthcoming 1996.
11. F. Facchinei and J. Soares. Testing a new class of algorithms for nonlinear complementarity problems. In F. Giannessi and A. Maugeri, editors, *Variational Inequalities and Network Equilibrium Problems*, pages 69–83. Plenum Press, New York, 1995.
12. M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. Discussion Papers in Economics 95–4, Department of Economics, University of Colorado, Boulder, Colorado, 1995. Available from ftp://ftp.cs.wisc.edu/math-prog/tech-reports/.
13. M. C. Ferris and D. Ralph. Projected gradient methods for nonlinear complementarity problems via normal maps. In D. Du, L. Qi, and R. Womersley, editors, *Recent Advances in Nonsmooth Optimization*, pages 57–87. World Scientific Publishers, 1995.
14. M. C. Ferris and T. F. Rutherford. Accessing realistic complementarity problems within Matlab. In G. Di Pillo and F. Giannessi, editors, *Proceedings of Nonlinear Optimization and Applications Workshop, Erice June 1995*, Plenum Press, New York, 1996.
15. A. Fischer. A special Newton-type optimization method. *Optimization*, 24:269–284, 1992.
16. A. Fischer and C. Kanzow. On finite termination of an iterative method for linear complementarity problems. Preprint MATH–NM–10–1994, Institute for Numerical Mathematics, Technical University of Dresden, Dresden, Germany, 1994.
17. R. Fourer, D. Gay, and B. Kernighan. *AMPL*. The Scientific Press, South San Francisco, California, 1993.
18. C. B. Garcia and W. I. Zangwill. *Pathways to Solutions, Fixed Points, and Equilibria*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1981.
19. C. Geiger and C. Kanzow. On the resolution of monotone complementarity problems. *Computational Optimization and Applications*, 5:155–173, 1996.
20. S.-P. Han, J. S. Pang, and N. Rangaraj. Globally convergent Newton methods for nonsmooth equations. *Mathematics of Operations Research*, 17:586–607, 1992.
21. P. T. Harker and B. Xiao. Newton's method for the nonlinear complementarity problem: A B–differentiable equation approach. *Mathematical Programming*, 48:339–358, 1990.
22. Lars Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. *Mathematical Programming Study*, 23:144–162, 1985.
23. J. J. Moré. Global methods for nonlinear complementarity problems. Technical Report MCS-P429-0494, Argonne National Laboratory, Argonne, Illinois, April 1994.

24. J. S. Pang. Newton's method for B–differentiable equations. *Mathematics of Operations Research*, 15:311–341, 1990.
25. J. S. Pang. A B–differentiable equation based, globally and locally quadratically convergent algorithm for nonlinear programs, complementarity and variational inequality problems. *Mathematical Programming*, 51:101–132, 1991.
26. J. S. Pang and S. A. Gabriel. NE/SQP: A robust algorithm for the nonlinear complementarity problem. *Mathematical Programming*, 60:295–338, 1993.
27. D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research*, 19:352–389, 1994.
28. S. M. Robinson. Generalized equations. In A. Bachem, M. Grötchel, and B. Korte, editors, *Mathematical Programming: The State of the Art, Bonn 1982*, pages 346–367. Springer Verlag, Berlin, 1983.
29. S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.
30. T. F. Rutherford. MILES: A mixed inequality and nonlinear equation solver. Working Paper, Department of Economics, University of Colorado, Boulder, 1993.
31. T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control, forthcoming*, 1996.
32. H. E. Scarf. The approximation of fixed points of a continuous mapping. *SIAM Journal on Applied Mathematics*, 15:1328–1343, 1967.
33. M. J. Todd. *Computation of Fixed Points and Applications*, volume 124 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Heidelberg, 1976.
34. B. Xiao and P. T. Harker. A nonsmooth Newton method for variational inequalities: I: Theory. *Mathematical Programming*, 65:151–194, 1994.
35. B. Xiao and P. T. Harker. A nonsmooth Newton method for variational inequalities: II: Numerical results. *Mathematical Programming*, 65:195–216, 1994.

Computational Optimization and Applications, 7, 27–40 (1997)

Impact of Partial Separability on Large-Scale Optimization*

ALI BOUARICHA AND JORGE J. MORÉ bouarich@mcs.anl.gov, more@mcs.anl.gov
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439

Abstract. ELSO is an environment for the solution of large-scale optimization problems. With ELSO the user is required to provide only code for the evaluation of a partially separable function. ELSO exploits the partial separability structure of the function to compute the gradient efficiently using automatic differentiation. We demonstrate ELSO's efficiency by comparing the various options available in ELSO. Our conclusion is that the hybrid option in ELSO provides performance comparable to the hand-coded option, while having the significant advantage of not requiring a hand-coded gradient or the sparsity pattern of the partially separable function. In our test problems, which have carefully coded gradients, the computing time for the hybrid AD option is within a factor of two of the hand-coded option.

Keywords: large-scale optimization, partial separability, automatic differentiation

1. Introduction

ELSO is an environment for the solution of large-scale minimization problems

$$\min \{ f_0(x) : x \in \mathbb{R}^n \}, \tag{1}$$

where $f_0 : \mathbb{R}^n \to \mathbb{R}$ is partially separable, that is, f_0 can be written as

$$f_0(x) = \sum_{i=1}^{m} f_i(x), \tag{2}$$

where each element function f_i depends only on a few components of x, and m is the number of element functions. Algorithms and software that take advantage of partial separability have been developed for various problems (for example, [11, 19, 20, 17, 21, 22, 10]), but this software requires that the user provide the gradient of f_0. An important design goal of ELSO is to avoid this requirement.

For small-scale problems we can approximate the gradient by differences of function values, for example,

$$[\nabla f_0(x)]_i \approx \frac{f_0(x + h_i e_i) - f_0(x)}{h_i}, \qquad 1 \le i \le n,$$

* Work supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008.

where h_i is the difference parameter, and e_i is the i-th unit vector, but this approximation suffers from truncation errors, which can cause premature termination of an optimization algorithm far away from a solution. We also note that, even for moderately sized problems with $n \geq 100$ variables, use of this approximation is prohibitive because it requires n function evaluations for each gradient. For these reasons, the accurate and efficient evaluation of the gradient is essential for the solution of optimization problems.

ELSO is able to solve large-scale unconstrained optimization problems, while requiring only that the user provide the function in partially separable form. This is an important advantage over standard software that requires the specification of the gradient and the sparsity pattern of the partially separable function, that is,

$$\mathcal{S} = \{(i,j) : f_i \text{ depends on } x_j\} = \{(i,j) : \partial_j f_i(x) \not\equiv 0\}. \tag{3}$$

ELSO exploits the partial separability structure of the function to compute the gradient efficiently by using automatic differentiation (AD). The current version of ELSO incorporates four different approaches for computing the gradient of a partially separable function in the context of large-scale optimization software. These approaches are hand-coded, compressed AD, sparse AD, and hybrid AD. In our work we have been using the ADIFOR (Automatic Differentiation of Fortran) tool [4, 6], and the SparsLinC (Sparse Linear Combination) library [5, 6], but other differentiation tools can be used.

We demonstrate ELSO's efficiency by comparing the compressed AD, sparse AD, and hybrid AD options with the hand-coded approach. Our conclusion is that the performance of the hybrid AD option is comparable with the compressed AD option and that the performance penalty over the hand-coded option is acceptable for carefully coded gradients. In our test problems, which have carefully coded gradients, the computing time for the hybrid AD option is within a factor of two of the hand-coded option. Thus, the hybrid AD option provides near-optimal performance, while providing the significant advantage of not requiring a hand-coded gradient or the sparsity pattern of the partially separable function.

We describe in Section 2 the different approaches used by ELSO to compute the gradient of a partially separable function. In Section 3 we provide a brief description of the MINPACK-2 large-scale problems and show how to convert these problems into partially separable problems. In Section 4 we compare and analyze the performance of large-scale optimization software using the different options available in ELSO. We present results for both a superscalar architecture (IBM RS6000) and a vector architecture (Cray C90). Our results on the Cray C90 are of special interest because they show that if the hand-coded gradient does not run at vector speeds, the hybrid AD option can outperform the hand-coded option. Finally, we present our conclusions in Section 5.

2. Computing Gradients in ELSO

In addition to hand-coded gradients, ELSO supports three approaches based on automatic differentiation for computing the gradient of a partially separable function. In this section we describe and compare these approaches.

ELSO relies on the representation (2) to compute the gradient of a partially separable function. Given this representation of $f_0 : \mathbb{R}^n \mapsto \mathbb{R}$, we can compute the gradient of f_0 by noting that if the mapping $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ is defined by

$$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix}, \tag{4}$$

then the gradient ∇f_0 can be obtained by

$$\nabla f_0(x) = f'(x)^T e, \tag{5}$$

where $e \in \mathbb{R}^m$ is the vector of all ones. The key observation is that the partial separability of f_0 implies that the Jacobian matrix $f'(x)$ is sparse, and thus automatic differentiation techniques can be used to compute the gradient ∇f_0 efficiently. The aim is to compute the gradient so that

$$T\{\nabla f_0(x)\} \leq \Omega_T \, T\{f_0(x)\}, \tag{6}$$

$$M\{\nabla f_0(x)\} \leq \Omega_M \, M\{f_0(x)\}, \tag{7}$$

where $T\{\cdot\}$ and $M\{\cdot\}$ denote computing time and memory, respectively, and Ω_T and Ω_M are small constants; if the function f_0 is defined by a discretization of a continuous problem, we also wish the constants to be independent of the mesh size. Any automatic differentiation tool can be used to compute $f'(x)$ and thus the gradient of f_0, but efficiency requires that we insist on (6) and (7).

Automatic differentiation tools can be classified roughly according to their use of the *forward* or the *reverse* mode of automatic differentiation. See, for example, the survey of Juedes [16]. Automatic differentiation tools that use the forward mode generate code for the computation of $f'(x)V$ for any $V \in \mathbb{R}^{n \times p}$. If $L\{f\}$ and $M\{f\}$ are, respectively, the number of floating-point operations and the amount of memory required by the computation of $f(x)$, then an AD-generated code employing the forward mode requires

$$L\{f'(x)V\} \leq (2 + 3p)\, L\{f\}, \qquad M\{f'(x)V\} \leq (1 + p)\, M\{f\},$$

floating-point operations and memory, respectively, to compute $f'(x)V$. For many large-scale problems we can obtain the Jacobian matrix $f'(x)$ by computing $f'(x)V$ for a matrix $V \in \mathbb{R}^{n \times p}$ with p small. Thus, in this case, an automatic differentiation tool based on the forward mode satisfies (6) and (7). We elaborate on this point when we discuss the compressed AD approach.

Automatic differentiation tools that use the reverse mode generate code for the computation of $W^T f'(x)$ for any $W \in \mathbb{R}^{m \times q}$. We can also use the reverse mode to compute $f'(x)$, but since the reverse mode reverses the partial order of program execution and remembers (or recomputes) any intermediate result that affects the final result, the complexity of the reverse mode is harder to predict. In general, the reverse mode requires $\mathcal{O}(L\{f\})$ floating-point operations and up to $\mathcal{O}(L\{f\} + M\{f\})$ memory, depending on the code. In particular, there is no guarantee that (7) is satisfied. Griewank [12, 13] has discussed

how to improve the performance of the reverse mode, but at present the potential memory demands of the reverse mode are a disadvantage. For additional information on automatic differentiation, see the proceedings edited by Griewank and Corlis [14]; the paper of Iri [15] is of special interest because he discusses the complexity of both the forward and the reverse modes of automatic differentiation.

In ELSO we have used the ADIFOR [4, 6] tool and the SparsLinC library [5, 6] because, from a computational viewpoint, they provide all the flexibility and efficiency desired on practical problems. Indeed, Bischof, Bouaricha, Khademi, Moré [3] have shown that the ADIFOR tool can satisfy (6) and (7) on large-scale variational problems.

We now outline the three approaches used by ELSO to compute the gradient of f_0. As we shall see, all these approaches have advantages and disadvantages in terms of ease of use, applicability, and computing time.

2.1. *Compressed AD Approach*

In the compressed AD approach we assume that the sparsity pattern of the Jacobian matrix $f'(x)$ is known for all vectors $x \in \mathcal{D}$, where $\mathcal{D}$ is a region where all the iterates are known to lie. For example, $\mathcal{D}$ could be the set

$$\mathcal{D} = \{x \in \mathbb{R}^n : f_0(x) \leq f_0(x_0)\},$$

where x_0 is the initial starting point. Thus, in the compressed AD approach we assume that the *closure* of the sparsity pattern is known. The sparsity pattern $\mathcal{S}(x)$ for $f'(x)$ at a given $x \in \mathcal{D}$ is just the set of indices

$$\mathcal{S}(x) = \{(i,j) : [f'(x)]_{i,j} \neq 0\};$$

the closure of the sparsity pattern of $f'(x)$ in the region $\mathcal{D}$ is

$$\bigcup \{\mathcal{S}(x) : x \in \mathcal{D}\}.$$

To determine the closure of the sparsity pattern, we are required to know how the function f_0 depends on the variables. When f is given by (4), a pair (i,j) is in the closure of the sparsity pattern if and only if f_i depends on x_j. Hence, the closure of the sparsity pattern is the sparsity pattern (3) of the partially separable function when x is restricted to lie in $\mathcal{D}$.

Determining the closure of the sparsity pattern is straightforward for problems with a fixed structure. For example, for finite element problems where the triangulation is fixed during the iteration. This is the case for the problems considered in Section 3. If, on the other hand, the structure evolves over time, then the sparsity pattern is likely to change as the iteration progresses. In these cases we must be able to detect these changes, and re-compute the sparsity pattern. This is the topic of current research.

Given the sparsity pattern of $f'(x)$, we can determine the Jacobian matrix $f'(x)$ if we partition the columns of the Jacobian matrix into groups of *structurally orthogonal* columns, that is, columns that do not have a nonzero in the same row position. In our work we employ the partitioning software described by Coleman, Garbow, and Moré [8, 7].

```
do j = 1, n
   grad(j) = 0.0
   do k = jpntr(j), jpntr(j+1)-1
      i = indrow(k)
      grad(j) = grad(j) + c_fjac(i,ngrp(j))
   enddo
enddo
```

Figure 1. Computing $\nabla f_0(x)$ from the compressed Jacobian array `c_fjac`

Given a partitioning of the columns of $f'(x)$ into p groups of structurally orthogonal columns, we can determine the Jacobian matrix $f'(x)$ by computing the *compressed Jacobian* matrix $f'(x)V$, where $V \in \mathbb{R}^{n \times p}$. There is a column of V for each group, and the k-th column is determined by setting the i-th component of v_k to one if the i-th column is in the k-th group, and to zero otherwise. For many sparsity patterns, the number of groups p is small and independent of n. For example, if a matrix is banded with bandwidth β or if it can be permuted to a matrix with bandwidth β, Coleman and Moré [9] show that $p \leq \beta$.

The compressed Jacobian matrix contains all the information of the Jacobian matrix. Given the compressed Jacobian matrix, we can recover $f'(x)$ in a sparse data structure. We can eliminate the storage and floating-point operations required to determine the sparse representation of the Jacobian matrix $f'(x)$, however, by computing the gradient of f_0 directly from the compressed Jacobian array `c_fjac` and storing the result in the array `grad`. This way of computing the gradient of f_0 is shown in the code segment in Figure 1. In this figure, `indrow` is the row index of the sparse representation of $f'(x)$, and `jpntr` specifies the locations of the row indices in `indrow`. The row indices for column `j` are `indrow(k), k = jpntr(j),...,jpntr(j+1)-1`, and `ngrp` specifies the partition of the columns of the sparse representation of $f'(x)$; column `j` belongs to group `ngrp(j)`.

2.2. *Sparse AD Approach*

For the sparse AD approach we need an automatic differentiation tool that takes advantage of sparsity when V and most of the vectors involved in the computation of $f'(x)V$ are sparse. We also require the sparsity pattern of $f'(x)V$ as a by-product of this computation. At present, the SparsLinC library [5, 6] is the only tool that addresses this situation, but we expect that others will emerge.

The main advantage of the sparse AD approach over the compressed AD approach is that no knowledge of the sparsity pattern is required. A disadvantage, however, is that because of the need to maintain dynamic data structures for sparse vectors, the sparse AD approach usually runs slower than the compressed AD approach.

Numerical results [3] with ADIFOR and SparsLinC show that the compressed AD approach outperforms the sparse AD approach on various architectures. In fact,

$$T\{\nabla f_0(x) : \text{sparse ADIFOR}\} = \kappa\, T\{\nabla f_0(x) : \text{compressed ADIFOR}\}$$

where κ satisfies

SPARC 10	IBM RS6000	Cray C90
$3 \le \kappa \le 8$	$6 \le \kappa \le 20$	$15 \le \kappa \le 45$

These results show, for example, that the solution of an optimization problem with a relatively expensive function evaluation is likely to require at least three times longer if we use sparse ADIFOR instead of compressed ADIFOR. Of course, for the compressed AD option we need to supply the sparsity pattern of the partially separable function.

Also note that the performance penalty of sparse ADIFOR is worst on superscalar (IBM RS6000) and vector (Cray C90) architectures. Thus, for these architectures, there is a stronger need to obtain the advantages of the sparse AD approach without giving up the speed of the compressed AD approach.

2.3. *Hybrid AD Approach*

As stated in the introduction, an important design goal of ELSO is to avoid asking the user to provide code for the evaluation of the gradient or the sparsity pattern of the partially separable function. We can achieve this goal by using the sparse AD option. However, as noted above, this imposes a heavy performance penalty on the user.

In an optimization algorithm we can avoid this performance penalty by first using the sparse AD option, to obtain the sparsity pattern of the function, and then using the compressed AD option. This strategy must be used with care. We should not use the sparse AD option to obtain the sparsity pattern at the starting point because the starting point is invariably special, and not representative of a general point in the region $\mathcal{D}$ of interest. In particular, there are usually many symmetries in the starting point that are not necessarily present in intermediate iterates.

We can also use the sparse AD option for a number of iterates until we feel that any symmetries present in the starting point have been removed by the optimization algorithm. This strategy is not satisfactory, however, because optimization algorithms tend to retain symmetries for many iterations, possibly for all the iterates.

The current strategy in ELSO is to randomly perturb every component of the user's initial point, and compute the sparsity pattern at the perturbed point. This destroys any symmetries in the original iterates, and the resulting sparsity pattern is likely to be the closure of the sparsity pattern in $\mathcal{D}$.

This strategy may fail if the closure of the sparsity pattern in a neighborhood of the initial iterate is different from the sparsity pattern in a neighborhood of the solution. For most optimization problems, this does not occur. If it occurs, however, failure does not occur unless some entries in the current sparsity pattern are not present in the previous sparsity pattern. The justification of this remark comes about by noting that the compressed AD approach works provided the sparsity pattern of the Jacobian matrix $f'(x)$ is a subset of the sparsity pattern provided by the user. Of course, if the sparsity pattern provided by the user is too large, then the number of groups p is likely to increase, leading to increased memory requirements and some loss in efficiency in the computation of the gradient.

Table 1. MINPACK-2 test problems

Name	Description of the Minimization Problems
EPT	Elastic-plastic torsion problem
GL1	Ginzburg-Landau (1-dimensional) superconductivity problem
GL2	Ginzburg-Landau (2-dimensional) superconductivity problem
MSA	Minimal surface area problem
ODC	Optimal design with composite materials problem
PJB	Pressure distribution in a journal bearing problem
SSC	Steady-state combustion problem

3. Partially Separable Test Problems

We used the test problems in the MINPACK-2 collection to compare the performance of a large-scale optimization software employing the four approaches for computing the gradient of a partially separable function described in Section 2. This collection is representative of large-scale optimization problems arising from applications. Table 1 lists each test problem with a short description; see [1] for additional information on these problems.

The optimization problems in the MINPACK-2 collection arise from the need to minimize a function f of the form

$$f(v) = \int_{\mathcal{D}} \Phi(x, v, \nabla v)\, dx, \tag{8}$$

where $\mathcal{D}$ is some domain in either $\mathbb{R}$ or $\mathbb{R}^2$, and Φ is defined by the application. In all cases f is well defined if $v : \mathcal{D} \mapsto \mathbb{R}^p$ belongs to $H^1(\mathcal{D})$, the Hilbert space of functions such that v and $\|\nabla v\|$ belong to $L^2(\mathcal{D})$.

Finite element approximations to these problems are obtained by minimizing f over the space of piecewise linear functions v with values $v_{i,j}$ at $z_{i,j}, 0 \le i \le n_y+1, 0 \le j \le n_x+1$, where $z_{i,j} \in \mathbb{R}^2$ are the vertices of a triangulation of $\mathcal{D}$ with grid spacings h_x and h_y. The vertices $z_{i,j}$ are chosen to be a regular lattice so that there are n_x and n_y interior grid points in the coordinate directions, respectively. Lower triangular elements T_L are defined by vertices $z_{i,j}, z_{i+1,j}, z_{i,j+1}$, while upper triangular elements T_U are defined by vertices $z_{i,j}, z_{i-1,j}, z_{i,j-1}$. A typical triangulation is shown in Figure 2.

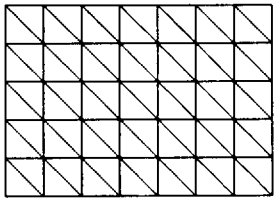

Figure 2. Triangulation of domain $\mathcal{D}$

In a finite element formulation of the variational problem defined by (8), the unknowns are the values $v_{i,j}$ of the piecewise linear function v at the vertices $z_{i,j}$. The values $v_{i,j}$ are obtained by solving the minimization problem

$$\min \left\{ \sum_{(i,j)} \left(f^L_{i,j}(v) + f^U_{i,j}(v) \right) : v \in \mathbb{R}^n \right\},$$

where $f^L_{i,j}$ and $f^U_{i,j}$ are the finite element approximation to the integrals

$$\int_{T_L} \Phi(x, v, \nabla v)\, dx, \qquad \int_{T_U} \Phi(x, v, \nabla v)\, dx,$$

respectively. Clearly, this is a partially separable problem because the element functions $f^L_{i,j}(v)$ and $f^U_{i,j}(v)$ depend only on the vertices $v_{i,j}, v_{i+1,j}, v_{i,j+1}$ and $v_{i,j}, v_{i-1,j}, v_{i,j-1}$, respectively. We can formulate this problem by setting

$$f(v) = \begin{pmatrix} f^L_{1,1}(v) \\ f^L_{1,2}(v) \\ \vdots \\ f^U_{1,1}(v) \\ f^U_{1,2}(v) \\ \vdots \end{pmatrix}. \tag{9}$$

In this case the number of element functions $m \approx 2n$. On the other hand, if we define

$$f(v) = \begin{pmatrix} f^L_{1,1}(v) + f^U_{1,1}(v) \\ f^L_{1,2}(v) + f^U_{1,2}(v) \\ \vdots \end{pmatrix}, \tag{10}$$

the number of element functions $m \approx n$. Since the number of element functions differs for (9) and (10), the number of groups p determined by the partitioning software [8, 7] is likely to be different, and thus the computing times for the compressed Jacobian matrix may depend on p. In our experience the computing time of formulation (9) is slightly better than that of (10). Therefore, we used formulation (9) in the numerical results of Section 4.

The problems in Table 1 are representative of a large class of optimization problems. These problems share some common characteristics. The main characteristics are that the computation of f requires order n flops and that the Jacobian matrix of f is sparse. Moreover, the number of groups p determined by the partitioning software leads to an almost dense compressed Jacobian matrix; the only exception is the GL2 problem, where the compressed Jacobian matrix is 50% dense. We expect that our numerical results are representative for any problem with these characteristics.

4. Numerical Results

Our aim in these experiments is to show that the performance of the hybrid AD option of ELSO is comparable to the compressed AD option and that the performance penalty over the hand-coded option is quite reasonable.

We chose a limited-memory variable metric method for these comparisons because codes of this type are commonly used to solve large-scale optimization problems. These methods are of the form

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k),$$

where $\alpha_k > 0$ is the search parameter, and the approximation H_k to the inverse Hessian matrix is stored in a compact representation that requires only the storage of $2n_v$ vectors, where n_v is chosen by the user. The compact representation of H_k permits the efficient computation of $H_k \nabla f(x_k)$ in $(8n_v + 1)n$ flops; all other operations in an iteration of the algorithm require $11n$ flops.

We used the `vmlm` implementation of the limited-memory variable metric algorithm (see Averick and Moré [2]) with $n_v = 5$. This implementation is based on the work of Liu and Nocedal [18]. The web page

`http://www.mcs.anl.gov/home/more/minpack-2/minpack-2.html`

contains additional information on the `vmlm` implementation.

In our numerical experiments we are interested in measuring performance in terms of time per iteration. Thus, instead of using a termination test, such as

$$\|\nabla f(x)\| \leq \tau \|\nabla f(x_0)\|,$$

we terminate after 100 iterations. This strategy is needed because optimization algorithms that require many iterations for convergence are affected by small perturbations in the function or the gradient, and, as a result, there may be large differences in the number of iterations required for convergence when the different `vmlm` options of ELSO are used.

All computations were performed on two platforms: an IBM RS6000 (model 370) using double-precision arithmetic, and a Cray C90 using single-precision arithmetic. The IBM RS6000 architecture has a superscalar chip and a cache-based memory architecture. Hence, this machine performs better when executing short vector operations, since these operations fill the short pipes and take advantage of memory locality. The Cray C90 is a vector processor without a cache that achieves full potential when the code has long vector operations. Without optimization of the source Fortran code, short vector loops and indirect addressing schemes perform poorly.

Table 2 has the computing time ratios of the compressed AD and sparse AD function-gradient evaluation to the hand-coded function-gradient evaluation on the IBM RS6000. These results show that the use of the sparse AD gradient can lead to a significant degradation in performance.

Tables 3 and 4 compare the computing time for the compressed AD, sparse AD, and hybrid AD options of `vmlm` to the computing time of the hand-coded option. The most important observation that can be made from these tables is that the computing times for

Table 2. Computing time ratios of the compressed AD and sparse AD function-gradient evaluation to the hand-coded function-gradient evaluation on the IBM RS6000 with $n = 10,000$

Prob	Compressed AD	Sparse AD
EPT	3.6	44.5
GL1	8.5	164.3
GL2	5.7	34.9
MSA	1.8	14.5
ODC	3.2	22.8
PJB	4.5	54.7
SSC	2.6	19.8

the hybrid AD option are approximately the same as those for the compressed AD option. The performance similarity between the hybrid AD option and the compressed AD option is expected because the difference in cost between the two options is only one sparse AD gradient evaluation and the partitioning of the columns of the Jacobian matrix into groups of structurally orthogonal columns. Tables 3 and 4 show that the hybrid AD option is clearly the method of choice because of its significant advantage of not requiring a hand-coded gradient or the sparsity pattern of the partially separable function.

Table 3. Computing time ratios of the compressed AD, sparse AD, and hybrid AD options of `vmlm` to the hand-coded option on the IBM RS6000 with $n = 10,000$

Prob	Compressed AD	Sparse AD	Hybrid AD
EPT	2.6	19.1	2.8
GL1	2.3	17.6	2.5
GL2	2.6	12.6	2.8
MSA	1.6	10.0	1.7
ODC	2.0	10.0	2.1
PJB	3.2	15.9	3.4
SSC	2.2	13.1	2.3

Table 4. Computing time ratios of the compressed AD, sparse AD, and hybrid AD options of `vmlm` to the hand-coded option on the IBM RS6000 with $n = 40,000$

Prob	Compressed AD	Sparse AD	Hybrid AD
EPT	2.8	19.2	3.0
GL1	2.3	17.6	2.5
GL2	2.8	12.2	2.9
MSA	1.7	10.1	1.9
ODC	2.1	10.0	2.2
PJB	3.3	14.8	3.4
SSC	2.3	13.3	2.4

The ratios in Tables 3 and 4 are below the corresponding ratios in Table 2. This result can be explained by noting that the ratios in Tables 3 and 4 can be expressed as

$$\frac{T_{ad} + T_{alg}}{T_{hc} + T_{alg}}, \tag{11}$$

where T_{ad}, T_{alg}, and T_{hc} are the computing times for the function and AD-generated gradient evaluation, the `vmlm` algorithm, and the function and hand-coded gradient evaluation, respectively. Since $T_{ad} > T_{hc}$, we have

$$\frac{T_{ad} + T_{alg}}{T_{hc} + T_{alg}} \leq \frac{T_{ad}}{T_{hc}},$$

which is the desired result. If T_{ad} and T_{hc} are the dominant costs, the ratio (11) should be close to T_{ad}/T_{hc}. This can be seen in the results for the MSA and SSC problem, since these are the two most expensive functions in the set.

Tables 3 and 4 also show that when we increase the problem dimension from $n = 10,000$ to $n = 40,000$, the corresponding compressed AD, sparse AD, and hybrid AD ratios remain about the same. This observation can be explained by noting that the ratio (11) can also be expressed as

$$\frac{rT_{hc} + T_{alg}}{T_{hc} + T_{alg}}, \tag{12}$$

where r is the ratio in Table 2. Since T_{hc} and T_{alg} grow by approximately a factor of 4 when n changes from 10,000 to 40,000, the ratio (12) remains constant.

We present results only for the SSC and GL2 problems on the Cray C90. We selected these problems because they have different characteristics. In particular, the number of groups in the compressed AD approach is $p = 3$ for the SSC problem, while $p = 9$ for the GL2 problem.

Table 5 presents the computing time ratios of the compressed AD and sparse AD function-gradient evaluation to the hand-coded function-gradient evaluation. The sparse AD approach uses the indirect addressing and dynamic memory allocation of the SparsLinC library [5, 6] and thus performs poorly on vector architectures [3]. As a result, the performance of the sparse AD approach is far from being practical on the Cray. In the rest of this section we present results only for the compressed and hybrid AD options.

Table 6 presents the computing time ratios of the compressed AD and hybrid AD options of `vmlm` to the hand-coded option. These results show that the performance of the hybrid AD option is comparable to that of the compressed AD option. On the other hand, the performance of the compressed AD option relative to the hand-coded option is poor for the GL2 problem. The reason for this poor performance is that the GL2 hand-coded gradient fully vectorizes, while the compressed AD gradient does not vectorize. Hence, the hand-coded gradient executes at vector speeds, while the compressed AD gradient executes at scalar speeds. The situation is different for the SSC function. In this case, neither the hand-coded gradient nor the compressed AD gradient vectorizes, so they both execute at scalar speeds.

Table 5. Computing time ratios of the compressed AD and sparse AD function-gradient evaluation to the hand-coded function-gradient evaluation on the Cray C90

Prob	n	Compressed AD	Sparse AD
GL2	10000	25.1	624.6
GL2	40000	28.3	694.4
SSC	10000	1.9	49.2
SSC	40000	1.9	49.4

Table 6. Computing time ratios of the compressed AD and hybrid AD options of `vmlm` to the hand-coded option on the Cray C90

Prob	n	Compressed AD	Hybrid AD
GL2	10000	14.2	18.0
GL2	40000	16.9	20.3
SSC	10000	1.9	2.4
SSC	40000	1.9	2.4

The poor performance of the compressed AD and hybrid AD options is due to the short innermost loops of length p, where p is the number of groups in the compressed AD approach. These loops are vectorizable, but when the compiler vectorizes only innermost loops, as is the case of the Cray C90, the performance degrades. We can vectorize the compressed AD gradient by *strip-mining* the computation of the gradient; that is, the gradient computation is divided into strips and each strip computes the gradient with respect to a few components of the independent variables. In the case of the compressed AD gradient, strip-mining can be done conveniently via the seed matrix mechanism. A disadvantage of the strip-mining approach is that the function is evaluated in every strip, resulting in a runtime overhead of `nstrips - 1` extra function evaluations, where `nstrips` is the number of strips. Using strips of size 5 is appropriate for the Cray C90 because the compiler unrolls innermost loops of length five or less, and, as a result, the loops that run over the grid points in the second coordinate direction are vectorized.

There is one additional complication. Since the value of p is not known at compile time, the Cray compiler cannot unroll a loop of length p even if the computed value of p at runtime is less than or equal to five. We fix this problem by setting the upper bound of the innermost loops to a fixed number at least equal to p but at most equal to 5. The generation of the compressed AD gradients with a fixed upper bound of the innermost loops can be done automatically by setting the appropriate ADIFOR flags [6].

The computing time ratios for the strip-mining approach (with loop unrolling) are shown in Tables 7 and 8. The improvement is dramatic for both the compressed AD and hybrid AD options. If we compare the results in Table 6 with those in Table 8, we find that the computing time ratios are reduced by a factor of 1.6 for the GL2 problem and a factor of 2 for the SSC problem.

Also note that the results in Tables 7 and 8 show that the compressed AD approach performs better on the SSC problem than the hand-coded approach. The reason for this

Table 7. Computing time ratios of the compressed AD function-gradient evaluation (with loop unrolling) to the hand-coded function-gradient evaluation on the Cray C90

Prob	n	Compressed AD
GL2	10000	13.3
GL2	40000	13.7
SSC	10000	0.4
SSC	40000	0.4

Table 8. Computing time ratios of the compressed AD and hybrid AD options of `vmlm` (with loop unrolling) to the hand-coded option on the Cray C90

Prob	n	Compressed AD	Hybrid AD
GL2	10000	8.9	12.7
GL2	40000	10.0	13.4
SSC	10000	0.5	1.0
SSC	40000	0.5	1.0

is that the strip-mining in the compressed AD approach improves the performance of this approach, while the hand-coded approach is still running at scalar speeds. These results illustrate the important point that the compressed and hybrid AD approaches can run faster than the hand-coded approach if the user does not provide a carefully coded gradient.

5. Conclusions

We have developed an environment for the solution of large-scale optimization problems, ELSO, in which the user is required to provide only code for the evaluation of a partially separable function. ELSO exploits the partial separability structure of the function to compute the gradient efficiently using automatic differentiation.

Our test results show that the hybrid option in ELSO provides performance that is often not more than two times slower than a well-coded hand-derived gradient on superscalar architectures, while having the significant advantage of not requiring a hand-coded gradient or the sparsity pattern of the partially separable function.

References

1. B. M. Averick, R. G. Carter, J. J. Moré, and G-L. Xue. The MINPACK-2 test problem collection. Preprint MCS-P153-0692, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.
2. B. M. Averick and J. J. Moré. Evaluation of large-scale optimization problems on vector and parallel architectures. *SIAM J. Optimization*, 4:708–721, 1994.
3. Christian Bischof, Ali Bouaricha, Peyvand Kahdemi, and Jorge J. Moré. Computing gradients in large-scale optimization using automatic differentiation. Preprint MCS-P488-0195, Argonne National Laboratory, Argonne, Illinois, 1995.

4. Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. ADIFOR: Generating derivative codes from Fortran programs. *Scientific Programming*, 1(1):1–29, 1992.
5. Christian Bischof, Alan Carle, and Peyvand Khademi. Fortran 77 interface specification to the SparsLinC library. Technical Report ANL/MCS-TM-196, Argonne National Laboratory, Argonne, Illinois, 1994.
6. Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs. Preprint MCS-P381-1194, Argonne National Laboratory, Argonne, Illinois, 1994. Also available as CRPC-TR94491, Center for Research on Parallel Computation, Rice University.
7. T. F. Coleman, B. S. Garbow, and J. J. Moré. Fortran subroutines for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 10:346–347, 1984.
8. T. F. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 10:329–345, 1984.
9. T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20:187–209, 1983.
10. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT. Springer Series in Computational Mathematics. Springer-Verlag, 1992.
11. A. Griewank and Ph. L. Toint. Numerical experiments with partially separable optimization problems. In D. F. Griffiths, editor, *Numerical Analysis: Proceedings Dundee 1983*, Lecture Notes in Mathematics 1066. Springer-Verlag, 1984.
12. Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse communication. *Optim. Methods Software*, 1:35–54, 1992.
13. Andreas Griewank. Some bounds on the complexity of gradients, Jacobians, and Hessians. In P.M. Pardalos, editor, *Complexity in Nonlinear Optimization*, pages 128–161. World Scientific Publishers, 1993.
14. Andreas Griewank and George F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. Society for Industrial and Applied Mathematics, 1991.
15. M. Iri. History of automatic differentiation and rounding error estimation. In A. Griewank and G. F. Corliss, editors, *Automatic Differentiation of Algorithms*, pages 3–16. SIAM, 1992.
16. David Juedes. A taxonomy of automatic differentiation tools. In Andreas Griewank and George Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 315–329. SIAM, 1991.
17. M. Lescrenier. Partially separable optimization and parallel computing. *Ann. Oper. Res.*, 14:213–224, 1988.
18. D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45:503–528, 1989.
19. Ph. L. Toint. Numerical solution of large sets of algebraic nonlinear equations. *Math. Comp.*, 46:175–189, 1986.
20. Ph. L. Toint. On large scale nonlinear least squares calculations. *SIAM J. Sci. Statist. Comput.*, 8:416–435, 1987.
21. Ph. L. Toint and D. Tuyttens. On large-scale nonlinear network optimization. *Math. Programming*, 48:125–159, 1990.
22. Ph. L. Toint and D. Tuyttens. LSNNO: A Fortran subroutine for solving large-scale nonlinear network optimization problems. *ACM Trans. Math. Software*, 18:308–328, 1992.

Computational Optimization and Applications, 7, 41–69 (1997)

On the Number of Inner Iterations Per Outer Iteration of a Globally Convergent Algorithm for Optimization with General Nonlinear Inequality Constraints and Simple Bounds

A. R. CONN arconn@watson.ibm.com
IBM T.J. Watson Research Center, Yorktown Heights, USA.

N. GOULD n.gould@letterbox.rl.ac.uk
Rutherford Appleton Laboratory, Chilton, Oxfordshire, England.

PH. L. TOINT pht@math.fundp.ac.be
Department of Mathematics, Facultés Universitaires ND de la Paix, Namur, Belgium.

Abstract. This paper considers the number of inner iterations required per outer iteration for the algorithm proposed by Conn *et al.* [9]. We show that asymptotically, under suitable reasonable assumptions, a single inner iteration suffices.

Keywords: Nonlinear optimization, inequality constraints, barrier methods, complexity.

1. Introduction

In this paper, we consider the nonlinear programming problem

$$\underset{x\in\Re^n}{\text{minimize}} \quad f(x) \tag{1}$$

subject to the *general constraints*

$$c_i(x) \geq 0, \quad i = 1, \ldots, m, \tag{2}$$

and the specific *simple bounds*

$$l \leq x \leq u. \tag{3}$$

We assume that the region $\mathcal{B} = \{x \in \Re^n \mid l \leq x \leq u\}$ is non-empty and may be infinite. We do not rule out the possibility that further simple bounds on the variables are included amongst the general constraints (2) if that is deemed appropriate. Indeed, it is conceivable that all simple bounds should be handled this way. Furthermore, we assume that

AS1. $f(x)$ and the $c_i(x)$ are twice continuously differentiable for all x in $\mathcal{B}$.

Our exposition will be conveniently simplified by taking the lower bounds as identically equal to zero and the upper bound as infinity for a subset of $\mathcal{N} \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$ in (3) and by assuming that the remaining variables are either not subjected to simple bounds or their simple bounds are treated as general constraints. Thus, in most of what follows, $\mathcal{B} = \{x \in R^n \mid x_j \geq 0 \text{ for all } j \in \mathcal{N}_b\}$, where $\mathcal{N}_b \subseteq \mathcal{N}$ is the index set of *bounded* variables. The modification required to handle more general bounds is indicated at the end of the paper.

The approach we intend to take is that of Conn *et al.* [9] and is based upon incorporating the equality constraints via a Lagrangian barrier function whilst handling upper and lower bounds directly. The sequential, approximate minimization of the Lagrangian barrier function is performed in a trust region framework such as that proposed by Conn *et al.* [5].

Our aim in this paper is to consider how these two different algorithms mesh together. In particular, we aim to show that ultimately very little work is performed in the iterative sequential minimization algorithm for every iteration of the outer Lagrangian barrier algorithm. This is contrary to most analyses of sequential penalty and barrier function methods in which the effort required to solve the inner iteration subproblems is effectively disregarded, the analysis concentrating on the convergence of the outer iteration (see for instance the books by Fiacco and McCormick[12] and Bertsekas [1]. Exceptions to this are the sequential penalty function method analyzed by Gould [14], and the sequential augmented Lagrangian algorithm considered by Conn *et al.* [8]).

This work was primarily motivated by observations that the authors made when testing a prototype of their large-scale nonlinear programming package LANCELOT, release B (see [7] for a description of release A), which includes an implementation of the algorithms discussed in this paper. It was often apparent that only a single iteration of the inner iteration subroutine SBMIN was ultimately required for every outer iteration of our sequential Lagrangian barrier program. While the conditions required in this paper to turn this observation to a proven result are relatively strong (and we feel probably about as weak as is possible), the package frequently exhibits the same behaviour on problems which violate our assumptions.

We define the concepts and notation that we shall need in section 2. Our algorithm is fully described in section 3 and analyzed in sections 4 and 5.

2. Notation

Let $g(x)$ denotes the gradient $\nabla_x f(x)$ of $f(x)$. Similarly, let $A(x)$ denote the Jacobian of $c(x)$, where

$$c(x) = [c_1(x), \cdots, c_m(x)]^T. \tag{4}$$

Thus

$$A(x)^T = [\nabla c_1(x), \cdots \nabla c_m(x)]. \tag{5}$$

We define the Lagrangian and Lagrangian barrier functions as

$$\ell(x,\lambda) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x), \tag{6}$$

and

$$\Psi(x,\lambda,s) = f(x) - \sum_{i=1}^{m} \lambda_i s_i \log(c_i(x) + s_i), \tag{7}$$

respectively, where the components λ_i of the vector λ are positive and are known as Lagrange multiplier estimates and where the elements s_i of the vector s are positive and are known as shifts. We note that $\ell(x,\lambda)$ is the Lagrangian with respect to the general constraints only.

Let $g_\ell(x,\lambda)$ and $H_\ell(x,\lambda)$ respectively denote the gradient, $\nabla_x \ell(x,\lambda)$, and Hessian, $\nabla_{xx}\ell(x,\lambda)$, of the Lagrangian. We define the vector $\bar{\lambda}$ by

$$\bar{\lambda}_i(x,\lambda,s) = \frac{\lambda_i s_i}{c_i(x) + s_i}, \tag{8}$$

for all $1 \leq i \leq m$. We note that $\nabla_x \ell(x,\bar{\lambda}) = \nabla_x \Psi(x,\lambda,s)$.

We denote the non-negativity restrictions by

$$x \in \mathcal{B} = \{x \in R^n \mid x_j \geq 0 \quad \text{for all} \quad j \in \mathcal{N}_b\} \tag{9}$$

where $\mathcal{N}_b \subseteq \mathcal{N}$. We will make much use of the projection operator defined componentwise by

$$(P[x,l,u])_j = \begin{cases} l_j & \text{if } x_j \leq l_j \\ u_j & \text{if } x_j \geq u_j \\ x_j & \text{otherwise.} \end{cases} \tag{10}$$

This operator projects the point x onto the region defined by the simple bounds (3). Let

$$P(x,v,l,u) = x - P[x - v, l, u]. \tag{11}$$

Furthermore, define $P[x] = P[x,\bar{l},\infty]$ and $P(x,v) = P(x,v,\bar{l},\infty)$, where $\bar{l}_j = 0$ for $j \in \mathcal{N}_b$ and $-\infty$ otherwise.

Let $x^{(k)} \in \mathcal{B}$ and $\lambda^{(k)}$ be given values of x and λ. If $h(x,\lambda,\ldots)$ is any function of x, $\lambda,\ldots$, we shall write $h^{(k)}$ as a shorthand for $h(x^{(k)},\lambda^{(k)},\ldots)$.

For any $x^{(k)}$ we have two possibilities for each component $x_j^{(k)}$, $j = 1,\ldots,n$, namely

(i) $j \in \mathcal{N}_b$ and $0 \leq x_j^{(k)} \leq (\nabla_x \Psi^{(k)})_j$ or

(ii) $j \in \mathcal{N}_f$ or $(\nabla_x \Psi^{(k)})_j < x_j^{(k)}$,

where $\mathcal{N}_f \stackrel{\text{def}}{=} \mathcal{N} \setminus \mathcal{N}_b$ is the index set of *free* variables. We shall call all $x_j^{(k)}$ that satisfy (i) *dominated* variables while the remaining $x_j^{(k)}$ are *floating* variables. It is important to notice that, as $x^{(k)} \in \mathcal{B}$,

$$(P(x^{(k)}, \nabla_x \Psi^{(k)}))_j = x_j^{(k)} \quad \text{whenever } x_j^{(k)} \text{ is dominated,} \tag{12}$$

while

$$(P(x^{(k)}, \nabla_x \Psi^{(k)}))_j = (\nabla_x \Psi^{(k)})_j \quad \text{otherwise.} \tag{13}$$

If x^* is the limit point of the (sub-)sequence $\{x^{(k)}\}_{k \in \mathcal{K}}$, we partition $\mathcal{N}$ into four index sets related to the two possibilities (i) and (ii) above and the corresponding x^*. We define

$$\begin{aligned}
\mathcal{D}_1 &\stackrel{\text{def}}{=} \{j \in \mathcal{N}_b \mid x_j^{(k)} \text{ is dominated } \forall k \in \mathcal{K} \text{ sufficiently large}\},\\
\mathcal{F}_1 &\stackrel{\text{def}}{=} \mathcal{N}_f \cup \{j \in \mathcal{N}_b \mid x_j^{(k)} \text{ is floating } \forall k \in \mathcal{K} \text{ sufficiently large, } x_j^* > 0\},\\
\mathcal{F}_2 &\stackrel{\text{def}}{=} \{j \in \mathcal{N}_b \mid x_j^{(k)} \text{ is floating } \forall k \in \mathcal{K} \text{ sufficiently large, } x_j^* = 0\},\\
\mathcal{F}_3 &\stackrel{\text{def}}{=} \mathcal{N} \setminus \mathcal{D}_1 \bigcup \mathcal{F}_1 \bigcup \mathcal{F}_2.
\end{aligned} \tag{14}$$

We also define

$$\begin{aligned}
\mathcal{I}(x) &\stackrel{\text{def}}{=} \{i \mid c_i(x) > 0\},\\
\mathcal{A}(x) &\stackrel{\text{def}}{=} \{i \mid c_i(x) \leq 0\},
\end{aligned} \tag{15}$$

the sets of *inactive* (strictly satisfied) and *active* (violated or just satisfied) constraints at the point x. We develop our algorithm so that the set $\mathcal{A}^* \equiv \mathcal{A}(x^*)$ at any limit point of our generated sequence is precisely the set of constraints for which $c_i(x^*) = 0$. We also write $\mathcal{I}^* \equiv \mathcal{I}(x^*)$.

We will use the notation that if $\mathcal{J}_1$ and $\mathcal{J}_2$ are any subsets of $\mathcal{N}$ and H is an n by n matrix, $H_{[\mathcal{J}_1, \mathcal{J}_2]}$ is the matrix formed by taking the rows and columns of H indexed by $\mathcal{J}_1$ and $\mathcal{J}_2$ respectively. Likewise, if A is an m by n matrix, $A_{[\mathcal{J}_1]}$ is the matrix formed by taking the columns of A indexed by $\mathcal{J}_1$.

We denote the (appropriately dimensioned) identity matrix by I; its j-th column is e_j. A vector of ones is denoted by e.

We will use a variety of vector and subordinate matrix norms. We shall only consider norms $\|\cdot\|_z$ which are *consistent* with the two-norm, that is, norms which satisfy the inequalities

$$\|v\|_z \leq a_0^{\frac{1}{2}} \|v\|_2 \quad \text{and} \quad \|v\|_2 \leq a_0^{\frac{1}{2}} \|v\|_z \tag{16}$$

for all vectors v and some constant $a_0 \geq 1$, independent of z. It then follows that, for any pair of two-norm-consistent norms $\|\cdot\|_y$ and $\|\cdot\|_z$,

$$\|v\|_z \leq a_0 \|v\|_y \quad \text{and} \quad \|v\|_y \leq a_0 \|v\|_z. \tag{17}$$

If r is any m-vector whose i-th component is r_i, we use the shorthand $r \equiv [r_i]_{i=1}^m$. Furthermore, if r is as above and $\mathcal{J}$ is a subset of $\{1, 2, \cdots, m\}$, $[r_i]_{i \in \mathcal{J}}$ is just the vector whose components are the r_i, $i \in \mathcal{J}$. Consequently, $\|[r_i]_{i=1}^m\| \equiv \|r\|$.

Following Conn *et al.* [9], we now describe an algorithm for solving (1), (2) and (9).

3. Statement of the algorithm

In order to solve the problem (1), (2) and (9), we consider the algorithmic model given in Figure 1.

We shall call the vector $P(x^{(k)}, \nabla_x \Psi^{(k)})$ the *projected gradient of the Lagrangian barrier function* or the *projected gradient* for short. The norms $\|\cdot\|_g$ and $\|\cdot\|_c$ are normally chosen to be either two or infinity norms.

Our decreasing sequence of $\mu^{(k)}$'s is given by $\mu^{(k)} = \mu_0(\tau)^{k_j}$, but any monotonic decreasing sequence of $\mu^{(k)}$'s converging to zero if Step 4 is executed an infinite number of times will suffice. It is also irrelevant, in theory, as to how we find a suitable point $x^{(k)}$ satisfying (21). However, from a practical perspective, a suitable point is found by an iterative procedure. In our algorithm, it is normal to try to start this inner iteration from, or close to, the solution to the last one. Indeed, from the point of view of the results we are about to establish, this is crucial. Such a starting point is desirable as function and derivative information from the conclusion of one inner iteration may be passed as input to the next. However, we need to bear in mind that the requirement of the second part of (21) may preclude us from picking such a starting point as it is possible that $c_i(x^{(k-1)}) + s_i^{(k)} \leq 0$ for some i. This issue is considered in depth in Conn *et al.* [9], where it is shown that $c_i(x^{(k)}) + s_i^{(k+1)} > 0$ for all $1 \leq i \leq m$ when Step 3 of the Outer-iteration Algorithm is executed, while techniques for finding a suitable alternative starting point when Step 4 occurs are given.

The main purpose of this paper is to show that asymptotically we take one inner iteration per outer iteration. More specifically, under certain assumptions, we first show that (23) is eventually satisfied at each outer iteration. We then show that, under additional assumptions, it is possible to satisfy the convergence test (21) after a single iteration of the algorithm given in Conn *et al.* [5].

The specific inner iteration algorithm we shall consider is given in Figure 2.

There are a number of possible ways of choosing $\gamma_0^{(k,j)}$ and $\gamma_3^{(k,j)}$ in Step 4. The simplest is merely to pick $\gamma_0^{(k,j)} = \gamma_0$ and $\gamma_3^{(k,j)} = \gamma_3$; other alternatives are discussed in Conn *et al.* [7].

It remains to give a description of the starting point, initial trust region radius and approximation to the Hessian of the Lagrangian, and of the calculation that is performed in Step 2 of the Inner-iteration Algorithm.

Let $0 < \theta < 1$. We let

$$\hat{x}_j^{(k-1)} = \begin{cases} 0 & \text{if } 0 \leq x_j^{(k-1)} \leq \theta(\nabla_x \Psi^{(k-1)})_j \quad \text{and} \quad j \in \mathcal{N}_b \\ x_j^{(k-1)} & \text{otherwise,} \end{cases} \tag{32}$$

and choose

$$x^{(k,0)} = \begin{cases} \hat{x}^{(k-1)} & \text{if } c(\hat{x}^{(k-1)}) + s^{(k)} > 0 \\ x^{(k-1)} & \text{otherwise.} \end{cases} \tag{33}$$

Thus variables which are significantly dominated at the end of the $(k-1)$-st iteration are set to their bounds while the remainder are left unaltered. This choice is made since, under

[Outer-iteration Algorithm]

Step 0 : [Initialization] The strictly positive constants η_0, ω_0, α_ω, β_ω, α_η, β_η, $\alpha_\lambda \leq 1$, $\tau < 1$, $\rho < 1$, $\gamma_2 < 1$, $\omega_* \ll 1$ and $\eta_* \ll 1$ for which

$$1 - (1 + \alpha_\lambda)^{-1} < \alpha_\eta < \min(1, \alpha_\omega) \quad \text{and} \quad \beta_\eta < \min(1, \beta_\omega). \tag{18}$$

are specified. A positive forcing parameter, $\bar{\mu}^{(0)}$, is given. Set

$$\mu^{(0)} = \min(\bar{\mu}^{(0)}, \gamma_2), \quad \omega^{(0)} = \omega_0 (\mu^{(0)})^{\alpha_\omega} \quad \text{and} \quad \eta^{(0)} = \eta_0 (\mu^{(0)})^{\alpha_\eta}. \tag{19}$$

An initial estimate of the solution, $x^{\text{est}} \in \mathcal{B}$, and vector of positive Lagrange multiplier estimates, $\lambda^{(0)}$, for which $c_i(x^{\text{est}}) + \mu^{(0)} (\lambda_i^{(0)})^{\alpha_\lambda} > 0$ are specified. Set $k = 0$.

Step 1 : [Inner iteration] Compute shifts

$$s_i^{(k)} = \mu^{(k)} (\lambda_i^{(k)})^{\alpha_\lambda}, \tag{20}$$

for $i = 1, \ldots, m$. Find $x^{(k)} \in \mathcal{B}$ such that

$$\| P(x^{(k)}, \nabla_x \Psi^{(k)}) \|_g \leq \omega^{(k)} \quad \text{and} \quad c_i(x^{(k)}) + s_i^{(k)} > 0, \quad (i = 1, \ldots, m). \tag{21}$$

Step 2 : [Test for convergence] If

$$\| P(x^{(k)}, \nabla_x \Psi^{(k)}) \|_g \leq \omega_* \text{ and } \| [c_i(x^{(k)}) \bar{\lambda}_i(x^{(k)}, \lambda^{(k)}, s^{(k)})]_{i=1}^m \|_c \leq \eta_*, \tag{22}$$

stop. If

$$\left\| \left[c_i(x^{(k)}) \bar{\lambda}_i(x^{(k)}, \lambda^{(k)}, s^{(k)}) / (\lambda_i^{(k)})^{\alpha_\lambda} \right]_{i=1}^m \right\|_c \leq \eta^{(k)}, \tag{23}$$

execute Step 3. Otherwise, execute Step 4.

Step 3 : [Update Lagrange multiplier estimates] Set

$$\begin{array}{lll} \lambda^{(k+1)} & = \bar{\lambda}(x^{(k)}, \lambda^{(k)}, s^{(k)}), & \\ \bar{\mu}^{(k+1)} & = \bar{\mu}^{(k)}, & \mu^{(k+1)} = \min(\bar{\mu}^{(k+1)}, \gamma_2), \\ \omega^{(k+1)} & = \omega^{(k)} (\mu^{(k+1)})^{\beta_\omega}, & \eta^{(k+1)} = \eta^{(k)} (\mu^{(k+1)})^{\beta_\eta}. \end{array} \tag{24}$$

Increase k by one and go to Step 1.

Step 4 : [Reduce the forcing parameter] Set

$$\begin{array}{lll} \lambda^{(k+1)} & = \lambda^{(k)}, & \\ \bar{\mu}^{(k+1)} & = \tau \bar{\mu}^{(k)}, & \mu^{(k+1)} = \min(\bar{\mu}^{(k+1)}, \gamma_2), \\ \omega^{(k+1)} & = \omega_0 (\mu^{(k+1)})^{\alpha_\omega}, & \eta^{(k+1)} = \eta_0 (\mu^{(k+1)})^{\alpha_\eta}. \end{array} \tag{25}$$

Increase k by one and go to Step 1.

End of Algorithm

Figure 1. Outer-iteration algorithm

[Inner-iteration Algorithm]

Step 0 : [Initialization] The positive constants $\mu < \eta < 1$ and $\gamma_0 \leq \gamma_2 < 1 \leq \gamma_3$ are given. The starting point, $x^{(k,0)}$, a nonnegative convergence tolerance, $\omega^{(k)}$, an initial trust region radius, $\Delta^{(k,0)}$, a symmetric approximation, $B^{(k,0)}$, to the Hessian of the Lagrangian, $H_\ell(x^{(k,0)}, \lambda^{(k)})$, and a two-norm-consistent norm $\|\cdot\|_g$ are specified. Compute $\Psi(x^{(k,0)}, \lambda^{(k)}, s^{(k)})$ and its gradient. Set the inner iteration counter $j = 0$.

Step 1 : [Test for convergence] If

$$\|P(x^{(k,j)}, \nabla_x \Psi^{(k,j)})\|_g \leq \omega^{(k)} \tag{26}$$

set $x^{(k)} = x^{(k,j)}$ and stop.

Step 2 : [Significantly reduce a model of the Lagrangian barrier function] Construct a quadratic model,

$$\begin{aligned} m^{(k,j)}(x^{(k,j)} + p) \stackrel{\text{def}}{=} \Psi(x^{(k,j)}, \lambda^{(k)}, \mu^{(k)}) + p^T \nabla_x \Psi(x^{(k,j)}, \lambda^{(k)}, \mu^{(k)}) \\ + \tfrac{1}{2} p^T (B^{(k,j)} + A(x^{(k,j)})^T D^{(k)}(x^{(k,j)}) A(x^{(k,j)}))p, \end{aligned} \tag{27}$$

of $\Psi(x + p, \lambda^{(k)}, \mu^{(k)})$, where

$$D^{(k)}(x) = \operatorname{diag}\left(\frac{\lambda_i^{(k)} s_i^{(k)}}{(c_i(x) + s_i^{(k)})^2} \right). \tag{28}$$

Compute a step $p^{(k,j)}$ which significantly reduces the value of $m^{(k,j)}(x^{(k,j)} + p)$.

Step 3 : [Compute a measure of the effectiveness of the step] Compute $\Psi(x^{(k,j)} + p^{(k,j)}, \lambda^{(k)}, s^{(k)})$ and the ratio

$$\rho^{(k,j)} = \frac{\Psi(x^{(k,j)}, \lambda^{(k)}, s^{(k)}) - \Psi(x^{(k,j)} + p^{(k,j)}, \lambda^{(k)}, s^{(k)})}{m^{(k,j)}(x^{(k,j)}) - m^{(k,j)}(x^{(k,j)} + p^{(k,j)})}. \tag{29}$$

Step 4 : [Accept or reject the step] For $\gamma_0^{(k,j)} \in [\gamma_0, 1)$ and $\gamma_3^{(k,j)} \in [1, \gamma_3]$, set

$$x^{(k,j+1)} = \begin{cases} x^{(k,j)} + p^{(k,j)} & \text{if } \rho^{(k,j)} > \mu \\ x^{(k,j)} & \text{otherwise,} \end{cases} \tag{30}$$

and

$$\Delta^{(k,j+1)} = \begin{cases} \gamma_0^{(k,j)} \Delta^{(k,j)} & \text{if } \rho^{(k,j)} \leq \mu \\ \Delta^{(k)} & \text{if } \mu < \rho^{(k,j)} < \eta \\ \gamma_3^{(k,j)} \Delta^{(k,j)} & \text{otherwise.} \end{cases} \tag{31}$$

Step 5 : [Updating] If necessary, compute the gradient of $\Psi(x^{(k,j+1)}, \lambda^{(k)}, \mu^{(k)})$ and a further approximation to the Hessian of the Lagrangian $B^{(k,j+1)}$. Increment the inner iteration counter j by one and go to Step 1.

End of Algorithm

Figure 2. Inner-iteration algorithm

a suitable non-degeneracy assumption (AS7 in section 4), the set of dominated variables is asymptotically the same as the set of variables which lie on their bounds (see [9], Theorem 5.4). Furthermore, under a second non-degeneracy assumption (AS5 in section 4), the assignment $x^{(k,0)} = \hat{x}^{(k-1)}$ is guaranteed for k sufficiently large. Our choice of $x^{(k,0)}$ then encourages subsequent iterates to encounter their asymptotic state as soon as possible.

We also pick $\Delta^{(k,0)}$ so that

$$\Delta^{(k,0)} \geq \kappa \|P(x^{(k,0)}, \nabla_x \Psi^{(k,0)})\|_g^{\zeta} \tag{34}$$

for some positive constants κ and $\zeta < 1$ (typical values might be $\kappa = 1$ and $\zeta = 0.9$). This value is chosen so that the trust region does not interfere with the asymptotic convergence of the algorithm, while providing a reasonable starting value in the earlier stages of the method.

Finally $B^{(k,0)}$ is taken to be any sufficiently good symmetric approximation to the Hessian of the Lagrangian function at $x^{(k)}$. We qualify what we mean by "sufficiently good" in the next section but suffice it to say that exact second derivatives satisfy this property and are often to be recommended.

The calculation in Step 2 is performed in two stages.

1. Firstly, the so-called *generalized Cauchy point*, $x^{C(k,j)} \equiv x^{(k,j)} + p^{C(k,j)}$, is determined. This is merely an approximation to the first local minimizer of the quadratic model, $m^{(k,j)}(x^{(k,j)} + p)$, along the Cauchy arc. The *Cauchy arc* is the path $x^{(k,j)} + p$, where

$$p = p^{(k,j)}(t) \stackrel{\text{def}}{=} P[x^{(k,j)} - t\nabla_x \Psi(x^{(k,j)}, \lambda^{(k)}, \mu^{(k)}), l, u] - x^{(k,j)}, \tag{35}$$

as the parameter t increases from 0, which finishes when the path first intersects the boundary of the trust region,

$$\|p\|_t \leq \Delta^{(k,j)}, \tag{36}$$

for some two-norm-consistent norm $\|\cdot\|_t$. Thus the Cauchy arc is simply the path which starts in the steepest descent direction for the model but which is subsequently "bent" to follow the boundary of the "box" region defined by the feasible region (9) (or, in general, (3)) and which stops on the boundary of the trust region (36). The two or infinity norm is normally chosen, the latter having some advantages as the trust region is then aligned with the feasible region (9). (Indeed, it is possible to extend the Cauchy arc along the boundary of the trust region when the infinity norm is used. Further reduction of the quadratic model along this extended Cauchy arc may prove beneficial.)

The method proposed by Conn *et al.* [5] calculates the exact generalized Cauchy point by marching along the Cauchy arc until either the trust region boundary is encountered or the model starts to increase. An alternative method by Moré [15] finds an approximation $p^{C(k,j)} = p^{(k,j)}(t^{C(k,j)})$ which is required to lie within the trust-region and to satisfy the Goldstein-type conditions

$$m^{(k,j)}(x^{(k,j)} + p^{(k,j)}(t^{C(k,j)})) \leq m^{(k,j)}(x^{(k,j)}) + \mu_1 p^{(k,j)}(t^{C(k,j)})^T \nabla_x \Psi(x^{(k,j)}, \lambda^{(k)}, s^{(k)}) \tag{37}$$

and

$$t^{C(k,j)} \geq \nu_1 \quad \text{or} \quad t^{C(k,j)} \geq \nu_2 t^{L(k,j)}, \tag{38}$$

where $t^{L(k,j)} > 0$ is any value for which

$$m^{(k,j)}(x^{(k,j)} + p^{(k,j)}(t^{L(k,j)})) \geq m^{(k,j)}(x^{(k,j)}) + \mu_2 p^{(k,j)}(t^{L(k,j)})^T \nabla_x \Psi(x^{(k,j)}, \lambda^{(k)}, s^{(k)}) \tag{39}$$

or

$$\|p^{(k,j)}(t^{L(k,j)})\|_t \geq \nu_3 \Delta^{(k,j)}, \tag{40}$$

and the positive constants μ_1, μ_2, ν_1, ν_2 and ν_3 satisfy the restrictions $\mu_1 < \mu_2 < 1$, $\nu_2 < 1$ and $\nu_3 < 1$. Condition (37) ensures that a sufficient reduction in the model takes place at each iteration while condition (38) is needed to guarantee that every step taken is non-negligible. Moré shows that it is always possible to pick such a value of $t^{C(k,j)}$ using a backtracking linesearch, starting on or near to the trust region boundary. Similar methods have been proposed by Calamai and Moré [4], Burke and Moré [2], Toint [16] and Burke *et al.* [3].

2. Secondly, we pick $p^{(k,j)}$ so that $x^{(k,j)} + p^{(k,j)}$ lies within (9), $\|p^{(k,j)}\|_t \leq \beta_2 \Delta^{(k,j)}$ and

$$m^{(k,j)}(x^{(k,j)}) - m^{(k,j)}(x^{(k,j)} + p^{(k,j)}) \geq \beta_3 [m^{(k,j)}(x^{(k,j)}) - m^{(k,j)}(x^{(k,j)} + p^{C(k,j)})] \geq 0 \tag{41}$$

for some positive $\beta_2 \geq 1$ and $\beta_3 \leq 1$. In fact, we typically choose $\beta_2 = \beta_3 = 1$, in which case we are merely requiring that the computed step gives a value of the model which is no larger than the value at the generalized Cauchy point.

In order to accelerate the convergence of the method, it is normal to try to bias the computed step towards the Newton direction.

The convergence analysis given by Conn *et al.* [5] for the Outer-iteration Algorithm indicates that it is desirable to construct improvements beyond the Cauchy point only in the subspace of variables which are free from their bounds at the Cauchy point. In particular, with such a restriction and with a suitable non-degeneracy assumption, it is then shown that the set of variables which are free from their bounds at the solution is determined after a finite number of iterations. This has the advantage of allowing one to analyze the asymptotic convergence rate of the method purely as if it were an unconstrained calculation, merely by focusing on the set of free variables.

Let $\mathcal{F}$ be a subset of $\mathcal{N}$ and let $\mathcal{D} = \mathcal{N} \setminus \mathcal{F}$. Furthermore, let

$$H^{(k,j)} \stackrel{\text{def}}{=} B^{(k,j)} + A(x^{(k,j)})^T D^{(k)}(x^{(k,j)}) A(x^{(k,j)}) \tag{42}$$

denote the composite approximation to the Hessian of the Lagrangian barrier function.

The specific Model-reduction Algorithm we shall consider is summarized in Figure 3.

In Step 2 of this method, the value of $p_{[\mathcal{F}]}$ would normally be computed as the aggregate step after a number of Conjugate Gradient (CG) iterations, where CG is applied to minimize the model in the subspace defined by the free variables. The CG process will end when either a new bound is encountered or the convergence test (45) is satisfied. The Model-reduction Algorithm is itself finite as the number of free variables at each pass of Step 2 is strictly monotonically decreasing. See the paper by Conn *et al.* [6] for further details.

4. Convergence analysis

We wish to analyze the asymptotic behaviour of the Outer-iteration Algorithm, that is in the case where $\omega_* = \eta_* = 0$. We require the following additional assumptions.

AS2. The matrix $A(x^*)_{[\mathcal{A}^*, \mathcal{F}_1]}$ is of full rank at any limit point x^* of the sequence $\{x^{(k)}\}$ generated by the Outer-iteration Algorithm with the set $\mathcal{F}_1$ defined by (14).

Under these assumptions we have the following result.

[Model-reduction Algorithm]

Step 0 : [Initialization] Select positive constants $\nu < 1, \xi < 1, \beta_2 \geq 1$ and $\beta_3 \leq 1$.

Step 1 : [Calculate the generalized Cauchy point] Calculate an approximation to the generalized Cauchy point $x^{C(k,j)} = x^{(k,j)} + p^{C(k,j)}$ using one of the previously mentioned techniques. Compute the set of variables, $\mathcal{F}^{C(k,j)}$, which are free from their bounds at $x^{C(k,j)}$. Set $x = x^{C(k,j)}$, $s = p^{C(k,j)}$ and $\mathcal{F} = \mathcal{F}^{C(k,j)}$.

Step 2 : [Further improve the model] Let $\mathcal{C}(\beta_2) = \mathcal{S} \bigcap \mathcal{T}(\beta_2)$, where

$$\mathcal{S} = \{p_{[\mathcal{F}]} \mid x^{(k,j)} + p \in \mathcal{B} \quad \text{and} \quad p_{[\mathcal{D}]} = p_{[\mathcal{D}]}^{C(k,j)}\} \tag{43}$$

and

$$\mathcal{T}(\beta_2) = \{p_{[\mathcal{F}]} \mid \|p\|_t \leq \beta_2 \Delta^{(k,j)} \quad \text{and} \quad p_{[\mathcal{D}]} = p_{[\mathcal{D}]}^{C(k,j)}\}. \tag{44}$$

If $p_{[\mathcal{F}]}$ lies on the boundary of $\mathcal{T}(\beta_2)$, set $p^{(k,j)} = p$ and stop. (If $\|\cdot\|_t$ is the infinity norm, it is possible to transfer components of $\mathcal{F}$ which lie on the trust-region boundary to $\mathcal{D}$ and to continue.) Otherwise, recompute $p_{[\mathcal{F}]}$ so that (41) is satisfied and either $p_{[\mathcal{F}]}$ lies strictly interior to $\mathcal{C}(\beta_2)$ with

$$\begin{aligned} &\|H_{[\mathcal{F},\mathcal{F}]}^{(k,j)} p_{[\mathcal{F}]} + (\nabla_x \Psi_{[\mathcal{F}]}^{(k,j)} + H_{[\mathcal{F},\mathcal{D}]}^{(k,j)} p_{[\mathcal{D}]})\|_g \\ &\quad \leq \min(\nu, \|P(x^{(k,j)}, \nabla_x \Psi^{(k,j)})\|_g^{\xi}) \cdot \|P(x^{(k,j)}, \nabla_x \Psi^{(k,j)})\|_g \end{aligned} \tag{45}$$

or $p_{[\mathcal{F}]}$ lies on the boundary of $\mathcal{C}(\beta_2)$. Reset $x_{[\mathcal{F}]}$ to $x_{[\mathcal{F}]} + p_{[\mathcal{F}]}$.

Step 3 : [Test for convergence] If $p_{[\mathcal{F}]}$ lies strictly interior to $\mathcal{C}(\beta_2)$ and (45) is satisfied or if it is decided that sufficient passes have been made, set $p^{(k,j)} = p$ and stop. Otherwise remove all of the indices in $\mathcal{F}$ for which $p_{[\mathcal{F}]_i}$ lies on the boundary of $\mathcal{S}$ and perform another pass by returning to Step 2.

End of Algorithm

Figure 3. Model-reduction Algorithm

THEOREM 1 ([9], Theorem 4.4) *Assume that AS1 and AS2 hold, that x^* is a limit point of the sequence $\{x^{(k)}\}$ generated by the Outer-iteration Algorithm and that*

$$\bar{\lambda}_i^{(k)} \stackrel{\text{def}}{=} \frac{\lambda_i^{(k)} s_i^{(k)}}{c_i(x^{(k)}) + s_i^{(k)}}, \tag{46}$$

for $i = 1, \cdots, m$. Then x^* is a Kuhn-Tucker (first order stationary) point for (1), (2) and (9) and the corresponding subsequences of $\{\bar{\lambda}^{(k)}\}$ and $\{\nabla_x \Psi^{(k)}\}$ converge to a set of Lagrange multipliers, λ^*, and the gradient of the Lagrangian, $g_\ell(x^*, \lambda^*)$, for the problem, respectively.

Now consider the following further assumptions.

AS3. The second derivatives of the functions $f(x)$ and the $c_i(x)$ are Lipschitz continuous at all points within an open set containing $\mathcal{B}$.

AS4. Suppose that (x^*, λ^*) is a Kuhn-Tucker point for the problem (1), (2) and (9), and

$$\begin{aligned} \mathcal{A}_1^* &\stackrel{\text{def}}{=} \{i \mid c_i(x^*) = 0 \quad \text{and} \quad \lambda_i^* > 0\} \\ \mathcal{A}_2^* &\stackrel{\text{def}}{=} \{i \mid c_i(x^*) = 0 \quad \text{and} \quad \lambda_i^* = 0\} \end{aligned} \tag{47}$$

and

$$\begin{aligned} \mathcal{J}_1 &\stackrel{\text{def}}{=} \mathcal{N}_f \cup \{j \in \mathcal{N}_b \mid (g_\ell(x^*, \lambda^*))_j = 0 \quad \text{and} \quad x_j^* > 0\} \\ \mathcal{J}_2 &\stackrel{\text{def}}{=} \{j \in \mathcal{N}_b \mid (g_\ell(x^*, \lambda^*))_j = 0 \quad \text{and} \quad x_j^* = 0\}. \end{aligned} \tag{48}$$

Then we assume that the matrix

$$\begin{pmatrix} H_\ell(x^*, \lambda^*)_{[\mathcal{J},\mathcal{J}]} & (A(x^*)_{[\mathcal{A},\mathcal{J}]})^T \\ A(x^*)_{[\mathcal{A},\mathcal{J}]} & 0 \end{pmatrix} \tag{49}$$

is non-singular for all sets $\mathcal{A}$ and $\mathcal{J}$, where $\mathcal{A}$ is any set made up from the union of $\mathcal{A}_1^*$ and any subset of $\mathcal{A}_2^*$ and $\mathcal{J}$ is any set made up from the union of $\mathcal{J}_1$ and any subset of $\mathcal{J}_2$.

AS5. (Strict complementary slackness condition 1) Suppose that (x^*, λ^*) is a Kuhn-Tucker point for problem (1), (2) and (9). Then

$$\mathcal{A}_2^* = \{i \mid c_i(x^*) = 0 \quad \text{and} \quad \lambda_i^* = 0\} = \emptyset. \tag{50}$$

AS6. The Outer-iteration Algorithm has a single limit point, x^*.

Under these additional assumptions, we are able to derive the following result.

THEOREM 2 ([9], Theorems 5.3 and 5.5) *Assume that AS1–AS6 hold. Then there is a constant* $\mu_{\min} > 0$ *such that the penalty parameter* $\mu^{(k)}$ *generated by the Outer-iteration Algorithm satisfies* $\mu^{(k)} = \mu_{\min}$ *for all* k *sufficiently large. Furthermore,* $x^{(k)}$ *and* $\bar{\lambda}^{(k)}_{[\mathcal{A}^*]}$ *satisfy the bounds*

$$\begin{aligned} &\|x^{(k)} - x^*\|_g \leq a_x(\mu_{\min})^{\alpha_\eta + k\alpha_\lambda\beta_\eta} \quad \textit{and} \\ &\|(\bar{\lambda}^{(k)} - \lambda^*)_{[\mathcal{A}^*]}\|_g \leq a_\lambda(\mu_{\min})^{\alpha_\eta + k\alpha_\lambda\beta_\eta}, \end{aligned} \tag{51}$$

for the two-norm-consistent norm $||.||_g$ *and some positive constants* a_x *and* a_λ*, while each* $|\bar{\lambda}^{(k)}_i|$*,* $i \in \mathcal{I}^*$*, converges to zero at a Q-superlinear rate.*

We shall now investigate the behaviour of the Outer-iteration Algorithm once the penalty parameter has converged to its asymptotic value, $\mu_{\min}$. There is no loss of generality in assuming that we restart the algorithm from the point which is reached when the penalty parameter is reduced for the last time. We shall call this iteration $k = 0$ and will start with $\mu^{(0)} = \mu_{\min}$. By construction, (23) is satisfied for all k and the updates (24) are always performed. Moreover,

$$\omega^{(k)} = \omega_0(\mu_{\min})^{\alpha_\omega + k\beta_\omega} \quad \text{and} \quad \eta^{(k)} = \eta_0(\mu_{\min})^{\alpha_\eta + k\beta_\eta}. \tag{52}$$

We require the following extra assumptions.

AS7. (Strict complementary slackness condition 2) Suppose that (x^*, λ^*) is a Kuhn-Tucker point for problem (1), (2) and (9). Then

$$\mathcal{J}_2 = \{j \in \mathcal{N}_b \mid (g_\ell(x^*, \lambda^*))_j = 0 \quad \text{and} \quad x^*_j = 0\} = \emptyset. \tag{53}$$

AS8. If $\mathcal{J}_1$ is defined by (48), the approximations $B^{(k,0)}$ satisfy

$$\|(B^{(k,0)} - \nabla_{xx}\ell(x^*, \lambda^*))_{[\mathcal{J}_1,\mathcal{J}_1]}p^{(k,0)}_{[\mathcal{J}_1]}\|_g \leq \upsilon\|p^{(k,0)}_{[\mathcal{J}_1]}\|^{1+\varsigma}_g, \tag{54}$$

for some positive constants υ and ς and all k sufficiently large.

AS9. Suppose that (x^*, λ^*) is a Kuhn-Tucker point for the problem (1), (2) and (9), and that $\mathcal{J}_1$ is defined by (48). Then we assume that the second derivative approximations $B^{(k,0)}$ have a single limit, B^* and that the perturbed Kuhn-Tucker matrix

$$\begin{pmatrix} B^*_{[\mathcal{J}_1,\mathcal{J}_1]} & A(x^*)^T_{[\mathcal{A}^*,\mathcal{J}_1]} \\ A(x^*)_{[\mathcal{A}^*,\mathcal{J}_1]} & -(D^*_{[\mathcal{A}^*,\mathcal{A}^*]})^{-1} \end{pmatrix} \tag{55}$$

is non-singular and has precisely m negative eigenvalues, where D^* is the limiting diagonal matrix with entries

$$D^*_{i,i} \equiv \lim_{k\to\infty} D^{(k)}(x^{(k)})_{i,i} = \begin{cases} (\lambda^*_i)^{1-\alpha_\lambda}/\mu_{\min} & \text{if } i \in \mathcal{A}^* \\ 0 & \text{if } i \in \mathcal{I}^* \end{cases} \tag{56}$$

Assumptions AS5 and AS7 are often known as strict complementary slackness conditions. We observe that AS8 is closely related to the necessary and sufficient conditions for superlinear convergence of the inner iterates given by Dennis and Moré [10]. We also observe that AS9 is entirely equivalent to requiring that the matrix

$$B^*_{[\mathcal{J}_1,\mathcal{J}_1]} + A(x^*)^T_{[\mathcal{A}^*,\mathcal{J}_1]} D^*_{[\mathcal{A}^*,\mathcal{A}^*]} A(x^*)_{[\mathcal{A}^*,\mathcal{J}_1]} \tag{57}$$

is positive definite (see, for instance, Gould [13]). The uniqueness of the limit point in AS9 can also be relaxed by requiring that (57) has its smallest eigenvalue uniformly bounded from below by some positive quantity for all limit points B^* of the sequence $B^{(k,0)}$. Moreover it is easy to show that that AS4, AS5 and AS7 guarantee AS9 provided that $\mu_{\min}$ is sufficiently small and sufficient second-order optimality conditions (see Fiacco and McCormick [12], Theorem 4) hold at x^* (see Wright [17], Theorem 8, for the essence of a proof of this in our case). Although we shall merely assume that AS9 holds in this paper, it is of course possible to try to encourage this eventuality. We might, for instance, insist that Step 4 of the Outer-iteration Algorithm is executed rather than Step 3 so long as the matrix $H^{(k,0)}$ is not positive definite. This is particularly relevant if exact second derivatives are used.

We now show that if we perform the step calculation for the Inner-iteration Algorithm using the Model-reduction Algorithm, a single iteration of the Inner-iteration Algorithm suffices to complete an iteration of the Outer-iteration Algorithm when k is sufficiently large. Moreover, the solution of one inner-iteration subproblem, $x^{(k-1)}$ and the shifted starting point for the next inner iteration (33) are asymptotically identical. We do this by showing that, after a finite number of iterations,

(i) moving to the new starting point does not significantly alter the norms of the projected gradient or constraints. Furthermore, the status of each variable (floating or dominated) is unchanged by the move;

(ii) the generalized Cauchy point $x^{C(k,0)}$ occurs before the first "breakpoint" along the Cauchy arc — the breakpoints are the values of $t > 0$ at which the Cauchy arc changes direction as problem or trust region bounds are encountered. Thus the set of variables which are free at the start of the Cauchy arc $x^{(k,0)}$ and those which are free at the generalized Cauchy point are identical;

(iii) any step which satisfies (45) also satisfies $p_{[\mathcal{F}_1]}$ lies strictly interior to $\mathcal{C}(\beta_2)$. Thus a single pass of Step 2 of the Model-reduction Algorithm is required;

(iv) the step $p^{(k,0)}$ is accepted in Step 4 of the Inner-iteration Algorithm;

(v) the new point $x^{(k,1)}$ satisfies the convergence test (26); and

(vi) $x^{(k+1,0)} = x^{(k)}$.

We have the following theorem.

THEOREM 3 *Assume that assumptions AS1–AS9 hold and that the convergence tolerances β_ω and β_η satisfy the extra condition*

$$\beta_\omega < (1 + \min(\xi, \varsigma))\alpha_\lambda \beta_\eta. \tag{58}$$

Then for all k sufficiently large, a single inner iteration of the Inner-iteration Algorithm, with the step computed from the Model-reduction Algorithm, suffices to complete an iteration of the Outer-iteration Algorithm. Moreover, the solution to one inner iteration subproblem provides the starting point for the next without further adjustment, for all k sufficiently large.

Proof. In order to make the proof as readable as possible, we will make frequent use of the following shorthand: the iterates will be abbreviated as

$$x \equiv x^{(k)} \overset{(32)}{\rightarrow} \hat{x} \equiv \hat{x}^{(k)} \overset{(33)}{\rightarrow} x^{\oplus} \equiv x^{(k+1,0)} \overset{(26)}{\rightarrow} x^{+} \equiv x^{(k+1,1)}, \tag{59}$$

the shifts as

$$s \equiv s^{(k)} \rightarrow s^{+} \equiv s^{(k+1)}, \tag{60}$$

and the Lagrange multiplier estimates as

$$\lambda \equiv \lambda^{(k)} \rightarrow \bar{\lambda} \equiv \bar{\lambda}^{(k)} \equiv \bar{\lambda}(x, \lambda, s) \rightarrow \lambda^{+} \equiv \lambda^{(k+1)} \tag{61}$$

and

$$\bar{\lambda}^{+} \equiv \bar{\lambda}(x^{\oplus}, \lambda^{+}, s^{+}). \tag{62}$$

Other quantities which occur at inner iterations $(k+1, 0)$ and $(k+1, 1)$ will be given suffices $\oplus$ and $+$ respectively. Thus $H^{\oplus} \equiv H^{(k+1,0)}$ and $H^{+} \equiv H^{(k+1,1)}$.

Recall, we have used Theorem 2 to relabel the sequence of iterates so that

$$\|P(x^{(k)}, \nabla_x \Psi^{(k)})\|_g \leq \omega_0 (\mu_{\min})^{\alpha_\omega + k\beta_\omega} \tag{63}$$

and

$$\left\| \left[c_i(x^{(k)}) \bar{\lambda}_i^{(k)} / (\lambda_i^{(k)})^{\alpha_\lambda} \right]_{i=1}^{m} \right\|_c \leq \eta_0 (\mu_{\min})^{\alpha_\eta + k\beta_\eta} \tag{64}$$

for all $k \geq 0$. Let $\bar{\Omega}$ be any closed, bounded set containing the iterates $x^{(k)}$ and $x^{(k+1,0)}$. We shall follow the outline given above.

(i) Status of the starting point. The strict complementary slackness assumption AS7 ensures that for all k sufficiently large, each variable belongs exclusively to one of the sets $\mathcal{F}_1$ and $\mathcal{D}_1$ (see [9], Theorem 5.4); moreover,

$$g_\ell(x^*, \lambda^*)_j = 0 \quad \text{for all} \quad j \in \mathcal{F}_1 \quad \text{and} \quad x_j^* > 0 \quad \text{for all} \quad j \in \mathcal{F}_1 \cap \mathcal{N}_b \tag{65}$$

and

$$x_j^* = 0 \quad \text{and} \quad g_\ell(x^*, \lambda^*)_j > 0 \quad \text{for all} \quad i \in \mathcal{D}_1. \tag{66}$$

As one of $x_j^{(k)}$ and $\nabla_x \Psi_j^{(k)}$ ($\equiv \nabla_x \ell(x, \bar{\lambda})_j$) converges to zero while its partner converges to a strictly positive limit for each $j \in \mathcal{N}_b$ (assumption AS7), we may define nontrivial regions which separate the two sequences for all k sufficiently large. Let

$$\epsilon_x \stackrel{\text{def}}{=} \frac{\theta}{1+\theta} \min_{j \in \mathcal{N}_b} \max[x_j^*, g_\ell(x^*, \lambda^*)_j] > 0, \tag{67}$$

where θ is as in (32). Then there is an iteration k_0 such that for variables in $\mathcal{F}_1$,

$$|x_j^{(k)} - x_j^*| \leq \epsilon_x \quad \text{and} \quad |\nabla_x \Psi_j^{(k)}| < \epsilon_x, \tag{68}$$

while for those in $\mathcal{D}_1$,

$$|x_j^{(k)}| \leq \epsilon_x \quad \text{and} \quad |\nabla_x \Psi_j^{(k)} - g_\ell(x^*, \lambda^*)_j|, \leq \epsilon_x \tag{69}$$

for all $k \geq k_0$. Hence, for those variables in $\mathcal{D}_1$, (67) and (69) give that

$$\begin{aligned} x_j^{(k)} \leq \epsilon_x &= \theta[\min_{j \in \mathcal{N}_b} \max[x_j^*, g_\ell(x^*, \lambda^*)_j] - \epsilon_x] \\ &\leq \theta[g_\ell(x^*, \lambda^*)_j - \epsilon_x] \leq \theta(\nabla_x \Psi^{(k)})_j. \end{aligned} \tag{70}$$

Thus, by definition (32), $\hat{x}_j^{(k)} = 0$ for each $j \in \mathcal{D}_1$ when $k \geq k_0$. Similarly, when $j \in \mathcal{F}_1 \cap \mathcal{N}_b$ and $k \geq k_0$, $x_j^{(k)} > \theta(\nabla_x \Psi^{(k)})_j$ and hence, using (32), $\hat{x}_j^{(k)} = x_j$ for all $j \in \mathcal{F}_1$. Thus $\hat{x}^{(k)}$ converges to x^*.

The other strict complementary slackness assumption, AS5, ensures that each constraint belongs exclusively to one of the sets $\mathcal{I}^*$ and $\mathcal{A}^*$, for all k sufficiently large. Moreover,

$$c_i(x^*) = 0 \quad \text{and} \quad \lambda_i^* > 0 \quad \text{for all} \quad i \in \mathcal{A}^* \tag{71}$$

and

$$c_i(x^*) > 0 \quad \text{and} \quad \lambda_i^* = 0 \quad \text{for all} \quad i \in \mathcal{I}^*, \tag{72}$$

and thus one of $c_i(x^{(k)})$ and $\lambda_i^{(k+1)}$ converges to zero while its partner converges to a strictly positive limit for each i.

Using the shorthand introduced in (59)–(60), we have that $c_i(x) + s_i^+ > c_i(x) > 0$ for each $i \in \mathcal{I}^*$ and all k sufficiently large. Thus, as $\hat{x}$ converges to x^* and s_i^+ converges to zero, $2c_i(x^*) > c_i(\hat{x}) + s_i^+ > \frac{1}{2} c_i(x^*) > 0$ for all $i \in \mathcal{I}^*$ and k sufficiently large. On the other hand, if $i \in \mathcal{A}^*$, $c_i(x) + s_i^+ > 0$ for all k (see [9], Lemma 3.1). In this case, as s_i^+ converges to $s_i^* \equiv \mu_{\min}(\lambda_i^*)^{\alpha_\lambda} > 0$ and $c_i(x)$ converges to zero, the convergence of $\hat{x}$ to x^* and λ_i^+ to λ^* implies that $2s_i^* > c_i(\hat{x}) + s_i^+ > \frac{1}{2} s_i^* > 0$ for all k sufficiently large. Hence, from (33), $x^\oplus = \hat{x}$ and thus there is an integer $k_1 \geq k_0$ for which

$$x_j^\oplus = \begin{cases} x_j & \text{for all } j \in \mathcal{F}_1 \\ 0 & \text{for all } j \in \mathcal{D}_1, \end{cases} \tag{73}$$

for all $k \geq k_1$.

We next let r be any real number and consider points on the line

$$x(r) \stackrel{\text{def}}{=} x + r(x^{\oplus} - x). \tag{74}$$

We firstly show that the diagonal matrix $D(x(r))$ is bounded for all $0 \leq r \leq 1$, where D is given by (28). As x and $x^{\oplus}$ both converge to x^*, the definition (28) implies that $D(x(r))$ converges to the matrix $D^*_{i,i}$, satisfying (56), as k increases. Thus, we have the bound

$$\|D(x(r))\|_2 \leq a_1/\mu_{\min} \tag{75}$$

where $a_1 \stackrel{\text{def}}{=} 2\|[(\lambda_i^*)^{1-\alpha_\lambda}]_{i=1}^m\|_2$, for all k sufficiently large. It also follows from the convergence of x and $x^{\oplus}$ to x^* and that of s_i to s_i^* that there is an integer $k_2 \geq k_1$ for which

$$0 < \tfrac{1}{2}c_i(x^*) < c_i(x(r)) + s_i^{(l)} < 2c_i(x^*) \quad \text{for all} \quad i \in \mathcal{I}^* \tag{76}$$

and

$$0 < \tfrac{1}{2}\mu_{\min}(\lambda_i^*)^{\alpha_\lambda} < c_i(x(r)) + s_i^{(l)} < 2\mu_{\min}(\lambda_i^*)^{\alpha_\lambda} \quad \text{for all} \quad i \in \mathcal{A}^*, \tag{77}$$

for all k sufficiently large and $l \geq k_2$.

We now consider the starting point $x^{\oplus}$ for the next inner iteration in detail. Firstly, combining (12), (16) and (73), we have that

$$\|x^{\oplus} - x\|_z \leq a_0\|P(x, \nabla_x\Psi(x,\lambda,s))\|_g \leq a_0\omega_0(\mu_{\min})^{\alpha_\omega + k\beta_\omega} \tag{78}$$

for any two-norm-consistent norm $\|.\|_z$.

We may bound the change in $c(x)$, due to the shifted starting point, using the integral mean value theorem (see, eg, [11], page 74), the boundedness of $A(x)$ (assumption AS1 and the definition of $\bar{\Omega}$) and inequalities (17) and (78) to obtain

$$\begin{aligned} |c_i(x^{\oplus}) - c_i(x)| \leq & \ \| \textstyle\int_0^1 A(x(r))dr\|_g \|x^{\oplus} - x\|_g \\ \leq & \ a_0 a_2 \omega_0(\mu_{\min})^{\alpha_\omega + k\beta_\omega} \end{aligned} \tag{79}$$

where $x(r)$ is given by (74) and a_2 is an upper bound on $\|A(x)\|_g$ within $\bar{\Omega}$.

We next bound the differences in gradients of the Lagrangian barrier function at x and $x^{\oplus}$. Using the integral mean value theorem, the convergence of $\bar{\lambda} \equiv \lambda^+$ to λ^* (Theorem 1), the boundedness of the Hessian of the Lagrangian (with bounded multiplier estimates) and the constraint Jacobian within $\bar{\Omega}$ (assumption AS1) and the inequalities (17), (75) and (78), we obtain

$$\begin{aligned} & |\nabla_x\Psi(x^{\oplus},\lambda,s)_j - \nabla_x\Psi(x,\lambda,s)_j| \\ & \quad \leq \|x^{\oplus} - x\|_2 \cdot \|e_j^T \textstyle\int_0^1 [H_\ell(x(r),\lambda) + A(x(r))^T D(x(r))A(x(r))]dr\|_2 \\ & \quad \leq a_0^2(a_3 + a_1 a_2^2/\mu_{\min})\omega_0(\mu_{\min})^{\alpha_\omega + k\beta_\omega} \\ & \quad \leq a_0^2(a_3 + a_1 a_2^2)\omega_0(\mu_{\min})^{\alpha_\omega - 1 + k\beta_\omega}, \end{aligned} \tag{80}$$

where a_3 is an upper bound on the two-norm of the Hessian of the Lagrangian function (with bounded multiplier estimates) within $\bar{\Omega}$. We now use the identity

$$\lambda_i^+ = \frac{\lambda_i s_i}{c_i(x) + s_i} \tag{81}$$

to derive the relationship

$$\begin{aligned}
&\nabla_x \Psi(x^\oplus, \lambda^+, s^+) - \nabla_x \Psi(x^\oplus, \lambda, s) \\
&\quad = \sum_{i=1}^m \left(\frac{\lambda_i s_i}{c_i(x^\oplus)+s_i} - \frac{\lambda_i^+ s_i^+}{c_i(x^\oplus)+s_i^+} \right) a_i(x^\oplus) \\
&\quad = \sum_{i=1}^m \left(\frac{\lambda_i s_i}{c_i(x^\oplus)+s_i} - \frac{\lambda_i s_i}{c_i(x)+s_i} \right) a_i(x^\oplus) \\
&\qquad\quad + \sum_{i=1}^m \left(\frac{\lambda_i s_i}{c_i(x)+s_i} - \frac{\lambda_i^+ s_i^+}{c_i(x^\oplus)+s_i^+} \right) a_i(x^\oplus) \\
&\quad = \sum_{i=1}^m \left(\frac{\lambda_i^+ (c_i(x)-c_i(x^\oplus))}{c_i(x^\oplus)+s_i} + \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus)+s_i^+} \right) a_i(x^\oplus) \\
&\quad = \sum_{i\in\mathcal{A}^*} \left(\frac{\lambda_i^+ (c_i(x)-c_i(x^\oplus))}{c_i(x^\oplus)+s_i} + \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus)+s_i^+} \right) a_i(x^\oplus) \\
&\qquad\quad + \sum_{i\in\mathcal{I}^*} \left(\frac{\lambda_i^+ (c_i(x)-c_i(x^\oplus))}{c_i(x^\oplus)+s_i} + \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus)+s_i^+} \right) a_i(x^\oplus).
\end{aligned} \tag{82}$$

But, considering $i \in \mathcal{A}^*$, picking k sufficiently large so that $|\lambda_i^+| \leq 2|\lambda_i^*|$ and using the integral mean value theorem, the relationship $c(x^*)_{[\mathcal{A}^*]} = 0$, the bounds (77), (78), (79) and the inequalities (18) and (51), we obtain the bounds

$$\left| \frac{\lambda_i^+ (c_i(x) - c_i(x^\oplus))}{c_i(x^\oplus) + s_i} \right| \leq 4 a_0 a_2 \omega_0 (\lambda_i^*)^{1-\alpha_\lambda} (\mu_{\min})^{\alpha_\omega - 1 + k\beta_\omega} \tag{83}$$

and

$$\begin{aligned}
\left| \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus)+s_i^+} \right| &\leq 4(\lambda_i^*)^{1-\alpha_\lambda} (\mu_{\min})^{-1} \| \textstyle\int_0^1 a_i(x^\oplus + r(x^* - x^\oplus)) dr \|_g \|x^\oplus - x^*\|_g \\
&\leq 4(\lambda_i^*)^{1-\alpha_\lambda} (\mu_{\min})^{-1} a_2 (\|x^\oplus - x\|_g + \|x - x^*\|_g) \\
&\leq 4 a_2 (\lambda_i^*)^{1-\alpha_\lambda} (a_0 \omega_0 (\mu_{\min})^{\alpha_\omega - 1 + k\beta_\omega} + a_x (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta})
\end{aligned} \tag{84}$$

and hence

$$\begin{aligned}
&\| \sum_{i\in\mathcal{A}^*} \left(\frac{\lambda_i^+ (c_i(x)-c_i(x^\oplus))}{c_i(x^\oplus)+s_i} + \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus)+s_i^+} \right) a_i(x^\oplus) \|_z \\
&\quad \leq m a_0 a_2 \left(\max_{i\in\mathcal{A}^*} \left| \frac{\lambda_i^+ (c_i(x)-c_i(x^\oplus))}{c_i(x^\oplus)+s_i} \right| + \max_{i\in\mathcal{A}^*} \left| \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus)+s_i^-} \right| \right) \\
&\quad \leq a_{\mathcal{A}} (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta},
\end{aligned} \tag{85}$$

where $a_{\mathcal{A}} = 4 m a_0 a_2^2 (2 a_0 \omega_0 + a_x) \max_{i\in\mathcal{A}^*} (\lambda_i^*)^{1-\alpha_\lambda}$, for any two-norm-consistent norm $\|.\|_z$. Furthermore, the superlinear convergence of λ_i to zero, $i \in \mathcal{I}^*$, (76) and the boundedness of the remaining terms implies a bound

$$\| \sum_{i \in \mathcal{I}^*} \left(\frac{\lambda_i^+ (c_i(x) - c_i(x^\oplus))}{c_i(x^\oplus) + s_i} + \frac{\lambda_i^+ c_i(x^\oplus)}{c_i(x^\oplus) + s_i^+} \right) a_i(x^\oplus) \|_z \leq a_{\mathcal{I}} (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}, \tag{86}$$

for some constant $a_{\mathcal{I}}$ (In fact, this term can be made arbitrarily smaller than (85) by picking k sufficiently large). Thus, combining (82), (85) and (86), we obtain the componentwise bound

$$|\nabla_x \Psi_j^\oplus - \nabla_x \Psi(x^\ominus, \lambda, s)_j| \leq (a_{\mathcal{A}} + a_{\mathcal{I}})(\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta} \tag{87}$$

for all $j \in \mathcal{N}$ where we have abbreviated $\nabla_x \Psi(x^\oplus, \lambda^+, s^+)$ as $\nabla_x \Psi^\oplus$.

Now consider the variables whose indices j lie in $\mathcal{F}_1$ for $k \geq k_2$. Firstly, (65), (67), (68) and (73) show that

$$x_j^\oplus = x_j \geq \frac{x_j^*}{1 + \theta} > 0 \tag{88}$$

if $j \in \mathcal{N}_b$. Secondly, combining (80) and (87), and using (13), (17), (18) and (63), we derive the inequality

$$\begin{aligned} &|\nabla_x \Psi_j^\oplus| \\ &\quad \leq |\nabla_x \Psi_j^\oplus - \nabla_x \Psi(x^\oplus, \lambda, s)_j| + |\nabla_x \Psi(x^\oplus, \lambda, s)_j - \nabla_x \Psi(x, \lambda, s)_j| \\ &\qquad + |\nabla_x \Psi(x, \lambda, s)_j| \\ &\quad \leq (a_{\mathcal{A}} + a_{\mathcal{I}})(\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta} + a_0^2 (a_3 + a_1 a_2^2) \omega_0 (\mu_{\min})^{\alpha_\omega - 1 + k\beta_\omega} \\ &\qquad + a_0 \omega_0 (\mu_{\min})^{\alpha_\omega + k\beta_\omega} \\ &\quad \leq a_4 (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}, \end{aligned} \tag{89}$$

where $a_4 \stackrel{\text{def}}{=} a_{\mathcal{A}} + a_{\mathcal{I}} + a_0 \omega_0 (1 + a_0 (a_3 + a_1 a_2^2))$. As k increases, the right-hand-side of the inequality (89) converges to zero. Thus, from (68) and for k sufficiently large, $x_j^\oplus$ is floating for each $j \in \mathcal{F}_1$, and (13) and (89) imply that

$$|P(x^\oplus, \nabla_x \Psi_j^\oplus)| = |\nabla_x \Psi_j^\oplus| \leq a_4 (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}. \tag{90}$$

Conversely, consider the variables which lie in $\mathcal{D}_1$ for $k \geq k_2$. Then, combining (80) and (87), and using (17) and (18) we obtain the inequality

$$\begin{aligned} &|\nabla_x \Psi_j^\oplus - \nabla_x \Psi(x, \lambda, s)_j| \\ &\quad \leq |\nabla_x \Psi_j^\oplus - \nabla_x \Psi(x^\ominus, \lambda, s)_j| + |\nabla_x \Psi(x^\ominus, \lambda, s)_j - \nabla_x \Psi(x, \lambda, s)_j| \\ &\quad \leq (a_{\mathcal{A}} + a_{\mathcal{I}})(\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta} + a_0^2 (a_3 + a_1 a_2^2) \omega_0 (\mu_{\min})^{\alpha_\omega - 1 + k\beta_\omega} \\ &\quad \leq a_5 (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}, \end{aligned} \tag{91}$$

where $a_5 \stackrel{\text{def}}{=} a_{\mathcal{A}} + a_{\mathcal{I}} + a_0^2\omega_0(a_3 + a_1 a_2^2)$. Thus, for sufficiently large k the right-hand-side of (91) can be made arbitrarily small. Combining this result with (69) and the identity $x_j^{\oplus} = 0$, we see that $x_j^{\oplus}$ is dominated for each $j \in \mathcal{D}_1$, and (12) and (91) imply that

$$P(x^{\oplus}, \nabla_x \Psi_j^{\oplus}) = x_j^{\oplus} = 0. \tag{92}$$

Therefore, using (13), (17), (90) and (92), we have

$$\|P(x^{\oplus}, \nabla_x \Psi^{\oplus})\|_g = \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g \leq a_6(\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}, \tag{93}$$

for all k sufficiently large, where $a_6 \stackrel{\text{def}}{=} a_0 a_4 \|e_{[\mathcal{F}_1]}\|_2$.

We also need to be able to bound the Lagrange multiplier estimates $\bar{\lambda}^+ \equiv \bar{\lambda}(x^{\oplus}, \lambda^+, s^+)$. We have, from (8), that

$$|\lambda_i^+ - \bar{\lambda}_i^+| = \left| \frac{\bar{\lambda}_i^+ c_i^{\oplus}}{c_i^{\oplus} + s_i^+} \right|. \tag{94}$$

But then, recalling (84), when $i \in \mathcal{A}^*$, and the superlinear convergence of λ_i^+ to zero, when $i \in \mathcal{I}^*$, together with (18), we obtain a bound

$$\|\lambda^+ - \bar{\lambda}^+\|_g \leq a_{\lambda^+}(\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}, \tag{95}$$

for some constant a_{λ^+}. Thus, combining (51) and (95), we see that $\bar{\lambda}^+$ converges to λ^*, $i \in \mathcal{A}^*$, and, because λ_i^+ converges superlinearly to zero when $i \in \mathcal{I}^*$,

$$\|\bar{\lambda}^+ - \lambda^*\|_g \leq a_{\lambda^{\oplus}}(\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}, \tag{96}$$

for some constant $a_{\lambda^{\oplus}}$.

(ii) The generalized Cauchy point. We consider the Cauchy arc emanating from $x^{\oplus}$. We have shown that the variables in $\mathcal{D}_1$ are on their bounds; the relationships (66), (68), (69) and (91) imply that $\nabla_x \Psi_j^{\oplus} > 0$ for all sufficiently large k and hence that $p^{\oplus}(t)_j = 0$ for all $t > 0$ and $j \in \mathcal{D}_1$. Thus the variables in $\mathcal{D}_1$ remain fixed on the bounds throughout the first inner iteration and

$$p^{\oplus}_{[\mathcal{D}_1]} = 0 \tag{97}$$

for all k sufficiently large.

The remaining variables, those indexed by $\mathcal{F}_1$, are free from their bounds. Because of Assumption 7 the set $\mathcal{J}_1$ in assumption AS9 is identical to $\mathcal{F}_1$ and thus the matrix (57) is positive definite with extreme eigenvalues $0 < \pi_{\min} \leq \pi_{\max}$, say. Using (73) and inequalities (12), (13) and the first part of (21), we deduce that $x^{\oplus}$ converges to x^*. Thus the matrix

$$H^{\oplus}_{[\mathcal{F}_1, \mathcal{F}_1]} = B^{\oplus}_{[\mathcal{F}_1, \mathcal{F}_1]} + A(x^{\oplus})^T_{[\mathcal{F}_1]} D^+(x^{\oplus}) A(x^{\oplus})_{[\mathcal{F}_1]} \tag{98}$$

is also positive definite with extreme eigenvalues satisfying

$$0 < \tfrac{1}{2}\pi_{\min} \leq \pi^{\oplus}_{\min} \leq \pi^{\oplus}_{\max} \leq 2\pi_{\max}, \tag{99}$$

say, for all sufficiently large k. Hence the model (27) is a strictly convex function in the subspace of free variables during the first inner iteration.

We now show that the set

$$\mathcal{L} \stackrel{\text{def}}{=} \{p_{[\mathcal{F}_1]} \mid m^{\oplus}(x^{\oplus} + p) \leq m^{\oplus}(x^{\oplus}) \quad \text{and} \quad p_{[\mathcal{D}_1]} = 0\} \tag{100}$$

lies strictly interior to the set $\mathcal{C}(1)$ (defined in the Model-reduction Algorithm) for all k sufficiently large. The diameter d of $\mathcal{L}$, the maximum distance between two members of the set (measured in the two norm), can be no larger than twice the distance from the center of the ellipsoid defined by $\mathcal{L}$ to the point on $\bar{\mathcal{L}}$ (the boundary of $\mathcal{L}$) furthest from the center. The center of $\mathcal{L}$ is the Newton point,

$$p^*_{[\mathcal{F}_1]} = -(H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]})^{-1}\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}. \tag{101}$$

Let $p_{[\mathcal{F}_1]} \in \bar{\mathcal{L}}$ and $p^*_{[\mathcal{D}_1]} = 0$ and define $v \stackrel{\text{def}}{=} p - p^*$. Then, combining (27), (98), (100) and (101), we have that

$$\begin{aligned}
&\tfrac{1}{2} v^T_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} v_{[\mathcal{F}_1]} \\
&\quad = \tfrac{1}{2} p^{*T}_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} p^*_{[\mathcal{F}_1]} + (m^{\oplus}(x^{\oplus} + p^* + v) - m^{\oplus}(x^{\oplus})) \\
&\qquad -(p^* + v)^T_{[\mathcal{F}_1]}(H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} p^*_{[\mathcal{F}_1]} + \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}) \\
&\quad = \tfrac{1}{2} p^{*T}_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} p^*_{[\mathcal{F}_1]} = \tfrac{1}{2}\nabla_x \Psi^{\oplus T}_{[\mathcal{F}_1]}(H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]})^{-1}\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}.
\end{aligned} \tag{102}$$

Hence, using the extremal properties of the Rayleigh quotient and (102), we have

$$\begin{aligned}
d^2 \stackrel{\text{def}}{=}\ & 4\|v^*_{[\mathcal{F}_1]}\|_2^2 \leq 4 v^{*T}_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} v^*_{[\mathcal{F}_1]}/\pi^{\oplus}_{\min} \leq 8 v^{*T}_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} v^*_{[\mathcal{F}_1]}/\pi_{\min} \\
=\ & 8\nabla_x \Psi^{\oplus T}_{[\mathcal{F}_1]}(H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]})^{-1}\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}/\pi_{\min} \leq 16\|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2^2/\pi^2_{\min}
\end{aligned} \tag{103}$$

where $\|v^*_{[\mathcal{F}_1]}\|_2 = \max_{p^*_{[\mathcal{F}_1]} + v_{[\mathcal{F}_1]} \in \bar{\mathcal{L}}} \|v_{[\mathcal{F}_1]}\|_2$. Thus, using (17), (93) and (103), any step within $\mathcal{L}$ satisfies the bound,

$$\|p_{[\mathcal{F}_1]}\|_2 \leq d \leq 4\|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2/\pi_{\min} \leq 4a_0 a_6 (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}/\pi_{\min}, \tag{104}$$

for sufficiently large k.

The inequality (88) shows that $x^{\oplus}_j$, $j \in \mathcal{F}_1 \cap \mathcal{N}_b$, is separated from its bound for all k sufficiently large while (104) shows that all steps within $\mathcal{L}$ become arbitrarily small. Thus the problem bounds are excluded from $\mathcal{L}$. Moreover (16), (34), (93), (97) and (104) combine to give

$$\|p\|_t = \|p_{[\mathcal{F}_1]}\|_t \leq a_0^{\frac{1}{2}} \|p_{[\mathcal{F}_1]}\|_2 \leq \Delta^{\oplus} \frac{4a_0 \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g^{1-\zeta}}{\pi_{\min}\kappa}. \tag{105}$$

for all steps within, or on the boundary of, $\mathcal{L}$. Inequality (93) then combines with (105) to show that any such step is shorter than the distance to the trust region boundary for all k sufficiently large.

Thus $\mathcal{L}$ lies strictly interior to $\mathcal{C}(1) \subseteq \mathcal{C}(\beta_2)$ for all k sufficiently large. But, as all iterates generated by the Model-reduction Algorithm satisfy (41) and thus lie in $\mathcal{L}$, it follows that both the generalized Cauchy point and any subsequent improvements are not restricted by the boundaries of $\mathcal{C}$ or $\mathcal{C}(\beta_2)$.

It remains to consider the Cauchy step in more detail. The Cauchy arc starts in the steepest descent direction for the variables in $\mathcal{F}_1$. The minimizer of the model in this direction occurs when

$$t = t^* = \frac{\nabla_x \Psi^{\oplus T}_{[\mathcal{F}_1]} \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}}{\nabla_x \Psi^{\oplus T}_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}}. \tag{106}$$

and thus, from the above discussion, gives the generalized Cauchy point proposed by Conn *et al.* [5]. We use the definition of t^*, (16), (99) and the extremal property of the Rayleigh quotient to obtain

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{C\oplus}) = \tfrac{1}{2} t^* \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2^2 \geq \frac{\|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g^2}{4 a_0 \pi_{\max}} \tag{107}$$

for this variant of the generalized Cauchy point. Alternatively, if Moré's (1988) variant is used, the requirement (37) and the definition of the Cauchy arc imply that

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{C\oplus}) \geq \mu_1 t^{C\oplus} \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2^2. \tag{108}$$

If the first alternative of (38) holds, (108) implies that

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{C\oplus}) \geq \mu_1 \nu_1 \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2^2. \tag{109}$$

Otherwise, we may use the same arguments as above to show that it is impossible for $t^{L\oplus}$ to satisfy (40) when k is sufficiently large. Therefore, $t^{L\oplus}$ must satisfy (39). Combining (27), (39), (98) and the definition of the Cauchy arc, we have that

$$\tfrac{1}{2}(t^{L\oplus})^2 \nabla_x \Psi^{\oplus T}_{[\mathcal{F}_1]} H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]} \geq (1 - \mu_2) t^{L\oplus} \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2^2. \tag{110}$$

Hence, combining (99) and (110) with the extremal properties of the Rayleigh quotient, we have that $t^{L\oplus} \geq (1 - \mu_2)/\pi_{\max}$. Thus, when the second alternative of (38) holds, this result and (108) give that

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{C\oplus}) \geq [\mu_1 \nu_2 (1 - \mu_2)/\pi_{\max}] \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_2^2. \tag{111}$$

Therefore, (17), (109) and (111) give the inequality

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{C\oplus}) \geq (\mu_1/a_0) \min(\nu_1, \nu_2(1 - \mu_2)/\pi_{\max}) \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g^2. \tag{112}$$

We shall make use of these results in (iv) below.

(iii) Improvements beyond the generalized Cauchy point. We have that $x^{\oplus}_{[\mathcal{D}]} = 0$, and, as a consequence of (93), $\|P(x^{\oplus}, \nabla_x \Psi^{\oplus})\|_g^{\xi} \leq \nu$ for all k sufficiently large. Hence, because we have shown that any p in $\mathcal{L}$ lies strictly interior to $\mathcal{C}$, a single pass of Step 2 of the Model-reduction Algorithm is required. We must pick p to satisfy (45) and (41) by determining $p^{\oplus}_{[\mathcal{F}_1]}$ so that

$$\|H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} p^{\oplus}_{[\mathcal{F}_1]} + \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g \leq \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g^{1+\xi}. \tag{113}$$

and

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{\oplus}) \geq \beta_3 [m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{C\oplus})] \tag{114}$$

for some $\beta_3 \leq 1$. The set of values which satisfy (113) and (114) is non-empty as the Newton step (101) satisfies both inequalities.

It remains to consider such a step in slightly more detail. Suppose that $p^{\oplus}_{[\mathcal{F}_1]}$ satisfies (113). Let

$$r^{\oplus}_{[\mathcal{F}_1]} = H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} p^{\oplus}_{[\mathcal{F}_1]} + \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]} \tag{115}$$

Then combining (16), (99), (113) and (115), we have

$$\begin{aligned} \|p^{\oplus}_{[\mathcal{F}_1]}\|_g &\leq a_0 \|H^{\oplus -1}_{[\mathcal{F}_1,\mathcal{F}_1]}\|_2 (\|r^{\oplus}_{[\mathcal{F}_1]}\|_g + \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g) \\ &\leq 2a_0 \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g (1 + \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g^{\xi})/\pi_{\min}. \end{aligned} \tag{116}$$

Thus, combining (93) and (116), and picking k sufficiently large so that

$$\|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\| \leq 1, \tag{117}$$

we obtain the bound

$$\|p^{\oplus}_{[\mathcal{F}_1]}\|_g \leq 4a_0 a_6 (\mu_{\min})^{\alpha_\eta - 1 + k\alpha_\lambda \beta_\eta}/\pi_{\min}. \tag{118}$$

(iv) Acceptance of the new point. We have seen that

$$p^{\oplus}_{[\mathcal{D}_1]} = 0 \tag{119}$$

and $p^{\oplus}_{[\mathcal{F}_1]}$ satisfies (113). As $p^{\oplus}$ can be made arbitrarily small, it follows (as in (76) and (77)) from the convergence of $x^{\oplus}$ to x^* and that of s_i^+ to s_i^* that there is an integer k_3 for which

$$0 < \tfrac{1}{2} c_i(x^*) < c_i(x^{\oplus} + p^{\oplus}) + s_i^+ < 2c_i(x^*) \quad \text{for all} \quad i \in \mathcal{I}^* \tag{120}$$

and

$$0 < \tfrac{1}{2} \mu_{\min} (\lambda_i^*)^{\alpha_\lambda} < c_i(x^{\oplus} + p^{\oplus}) + s_i^+ < 2\mu_{\min} (\lambda_i^*)^{\alpha_\lambda} \quad \text{for all} \quad i \in \mathcal{A}^*. \tag{121}$$

for all k sufficiently large and $l \geq k_3$. Thus

$$c_i(x^{\oplus} + p^{\oplus}) + s_i^+ > 0 \tag{122}$$

for all $1 \leq i \leq m$ and k sufficiently large.

We now wish to show that the quantity

$$|\rho^{\oplus} - 1| = \frac{|\Psi(x^{\oplus} + p^{\oplus}, \lambda^+, s^+) - m^{\oplus}(x^{\oplus} + p^{\oplus})|}{|m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{\oplus})|} \tag{123}$$

converges to zero, ensuring that the new point will prove acceptable in Step 4 of the Inner-iteration Algorithm.

Consider first the denominator on the right-hand-side of (123). Combining (107), (112) and (114), we have

$$m^{\oplus}(x^{\oplus}) - m^{\oplus}(x^{\oplus} + p^{\oplus}) \geq a_7 \|\nabla_x \Psi(x^{\oplus}, \lambda^+, s^+)_{[\mathcal{F}_1]}\|_g^2, \tag{124}$$

where $a_7 = \beta_3 \min(1/(4a_0\pi_{\max}), \mu_1 \min(\nu_1, \nu_2(1-\mu_2)/\pi_{\max})/a_0)$. Turning to the numerator on the right-hand-side of (123), we use the integral mean value theorem to obtain

$$\begin{aligned}
&\Psi(x^{\oplus} + p^{\oplus}, \lambda^+, s^+) \\
&\quad = \Psi(x^{\oplus}, \lambda^+, s^+) + p_{[\mathcal{F}_1]}^{\oplus T} \nabla_x \Psi_{[\mathcal{F}_1]}^{\oplus} \\
&\qquad\quad + \tfrac{1}{2} \textstyle\int_0^1 p_{[\mathcal{F}_1]}^{\oplus T} \nabla_{xx} \Psi(x^{\oplus}(t), \lambda^+, s^+)_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus} dt \\
&\quad = \Psi(x^{\oplus}, \lambda^+, s^+) + p_{[\mathcal{F}_1]}^{\oplus T} \nabla_x \Psi_{[\mathcal{F}_1]}^{\oplus} \\
&\qquad\quad + \tfrac{1}{2} \textstyle\int_0^1 p_{[\mathcal{F}_1]}^{\oplus T} [\nabla_{xx} \Psi(x^{\oplus}(t), \lambda^+, s^+) - \nabla_{xx} \Psi^{\oplus}]_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus} dt \\
&\qquad\quad + \tfrac{1}{2} p_{[\mathcal{F}_1]}^{\oplus T} [\nabla_{xx} \Psi^{\oplus} - H^{\oplus}]_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus} + \tfrac{1}{2} p_{[\mathcal{F}_1]}^{\oplus T} H_{[\mathcal{F}_1, \mathcal{F}_1]}^{\oplus} p_{[\mathcal{F}_1]}^{\oplus} \\
&\quad = m^{\oplus}(x^{\oplus} + p^{\oplus}) + \tfrac{1}{2} p_{[\mathcal{F}_1]}^{\oplus T} [\nabla_{xx} \Psi^{\oplus} - H^{\oplus}]_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus} \\
&\qquad\quad + \tfrac{1}{2} \textstyle\int_0^1 p_{[\mathcal{F}_1]}^{\oplus T} [\nabla_{xx} \Psi(x^{\oplus}(t), \lambda^+, s^+) - \nabla_{xx} \Psi^{\oplus}]_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus} dt,
\end{aligned} \tag{125}$$

where $x^{\oplus}(t) = x^{\oplus} + tp^{\oplus}$ and we have abbreviated $\nabla_{xx} \Psi(x^{\oplus}, \lambda^+, s^+)$ as $\nabla_{xx} \Psi^{\oplus}$.

Considering the last two terms in (125) in turn, we have the bounds

$$\begin{aligned}
&|\tfrac{1}{2} p_{[\mathcal{F}_1]}^{\oplus T} [\nabla_{xx} \Psi^{\oplus} - H^{\oplus}]_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus}| \\
&\quad \leq \tfrac{1}{2} a_0 (v \|p_{[\mathcal{F}_1]}^{\oplus}\|_g^{\varsigma} + \|[\nabla_{xx} \ell(x^{\oplus}, \bar{\lambda}^+) - \nabla_{xx} \ell(x^*, \lambda^*)]_{[\mathcal{F}_1, \mathcal{F}_1]}\|_g) \|p_{[\mathcal{F}_1]}^{\oplus}\|_g^2,
\end{aligned} \tag{126}$$

using (16), (42), the definition of the Hessian of the Lagrangian barrier function and AS8, and

$$|\tfrac{1}{2} \int_0^1 p_{[\mathcal{F}_1]}^{\oplus T} [\nabla_{xx} \Psi(x^{\oplus}(t), \lambda^+, s^+) - \nabla_{xx} \Psi^{\oplus}]_{[\mathcal{F}_1, \mathcal{F}_1]} p_{[\mathcal{F}_1]}^{\oplus} dt| \leq \tfrac{1}{4} a_0 a_8 \|p_{[\mathcal{F}_1]}^{\oplus}\|_g^3, \tag{127}$$

using (16), the convergence (and hence boundedness) of the Lagrange multiplier estimates and the Lipschitz continuity of the second derivatives of the problem functions (assumption AS3) with some composite Lipschitz constant a_8. Thus, combining (116), (123), (124), (125), (126) and (127), we obtain

$$|\rho^{\oplus} - 1| \leq 2a_0^3(1 + \|\nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|^{\xi})^2_g \cdot \frac{\frac{1}{2}a_8\|p^{\oplus}_{[\mathcal{F}_1]}\|_g + v\|p^{\oplus}_{[\mathcal{F}_1]}\|^{\varsigma}_g + \|[\nabla_{xx}\ell(x^{\oplus}, \bar{\lambda}^+) - \nabla_{xx}\ell(x^*, \lambda^*)]_{[\mathcal{F}_1,\mathcal{F}_1]}\|_g}{a_7\pi^2_{\min}}. \quad (128)$$

As the right-hand-side of (128) converges to zero as k increases, $x^+ = x^{\oplus} + p^{\oplus}$ for all k sufficiently large.

(v) Convergence of the inner iteration at the new point. The relationship (122) ensures that x^+ satisfies the feasibility test of the second part of (21). We now show that x^+ satisfies the inner-iteration convergence test (26).

Firstly, in the same vein as (80) , for $j \in \mathcal{D}_1$ we have that

$$\begin{aligned} &|\nabla_x \Psi(x^+, \lambda^+, s^+)_j - \nabla_x \Psi^{\oplus}_j| \\ &\quad \leq \|p^{\oplus}\|_2 \cdot \|e_j^T \textstyle\int_0^1 [H_\ell(x^{\oplus}(t), \lambda) + A(x^{\oplus}(t))^T D^+(x^{\oplus}(t)) A(x^{\oplus}(t))]_j dt\|_2 \\ &\quad \leq a_0(a_3 + a_1 a_2^2/\mu_{\min})\|p^{\oplus}\|_2, \end{aligned} \quad (129)$$

where $x^{\oplus}(t) = x^{\oplus} + tp^{\oplus}$ and where we use the bound

$$\|D^+(x^{\oplus}(t))\| \leq a_1/\mu_{\min} \quad (130)$$

for all $0 \leq t \leq 1$. This latter follows from the definition (28) and the convergence of $x^{\oplus}$ and, because of (119) and (118), the convergence of $x^{\oplus} + p^{\oplus}$ to x^*. Thus, as the right-hand-side of (129) can be made arbitrarily small, by taking k sufficiently large, (69) and the identity $x_j^+ = x_j^{\oplus} = 0$ for each $j \in \mathcal{D}_1$, imply that x_j^+ is dominated for each $j \in \mathcal{D}_1$ while (12) and (92) imply that

$$P(x^+, \nabla_x \Psi(x^+, \lambda^+, s^+))_j = x_j^+ = 0. \quad (131)$$

We now consider the components of $P(x^+, \nabla_x \Psi(x^+, \lambda^+, s^+))_j$ for $j \in \mathcal{F}_1$. Using the integral mean value theorem, we have

$$\begin{aligned} &\nabla_x \Psi(x^+, \lambda^+, s^+)_{[\mathcal{F}_1]} \\ &\quad = \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]} + \textstyle\int_0^1 \nabla_{xx} \Psi(x^{\oplus}(t), \lambda^+, s^+)_{[\mathcal{F}_1,\mathcal{F}_1]} p^{\oplus}_{[\mathcal{F}_1]} dt \\ &\quad = [H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]} p^{\oplus}_{[\mathcal{F}_1]} + \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}] + [\nabla_{xx}\Psi^{\oplus} - H^{\oplus}]_{[\mathcal{F}_1,\mathcal{F}_1]} p^{\oplus}_{[\mathcal{F}_1]} \\ &\qquad + \textstyle\int_0^1 [\nabla_{xx}\Psi(x^{\oplus}(t), \lambda^+, s^+) - \nabla_{xx}\Psi^{\oplus}]_{[\mathcal{F}_1,\mathcal{F}_1]} p^{\oplus}_{[\mathcal{F}_1]} dt \end{aligned} \quad (132)$$

where $x^{\oplus}(t) = x^{\oplus} + tp^{\oplus}$. We observe that each of the three terms on the right-hand-side of (132) reflects a different aspect of the approximations made. The first corresponds

to the approximation to the Newton direction used, the second to the approximation of a nonlinear function by a quadratic and the third to the particular approximation to the second derivatives used. We now bound each of these terms in turn.

The first term satisfies the bound (113). Hence, combining (93) and (113), we obtain

$$\|H^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]}p^{\oplus}_{[\mathcal{F}_1]} + \nabla_x \Psi^{\oplus}_{[\mathcal{F}_1]}\|_g \leq a_6^{1+\xi}(\mu_{\min})^{(\alpha_\eta-1)(1+\xi)+k\alpha_\lambda\beta_\eta(1+\xi)}. \tag{133}$$

The same arguments as those used to establish (126) imply that the second term on the right-hand-side of (132) satisfies the bound

$$\begin{aligned}
&\|[\nabla_{xx}\Psi^{\oplus} - H^{\oplus}]_{[\mathcal{F}_1,\mathcal{F}_1]}p^{\oplus}_{[\mathcal{F}_1]}\|_g \\
&\quad\leq (\upsilon\|p^{\oplus}_{[\mathcal{F}_1]}\|_g^{\varsigma} + \|(\nabla_{xx}\ell(x^{\oplus},\bar{\lambda}^+) - \nabla_{xx}\ell(x^*,\lambda^*))_{[\mathcal{F}_1,\mathcal{F}_1]}\|_g)\|p^{\oplus}_{[\mathcal{F}_1]}\|_g \\
&\quad\leq (\upsilon\|p^{\oplus}_{[\mathcal{F}_1]}\|_g^{\varsigma} + a_9\|x^{\oplus} - x^*\|_g + a_{10}\|\bar{\lambda}^+ - \lambda^*\|_g)\|p^{\oplus}_{[\mathcal{F}_1]}\|_g,
\end{aligned} \tag{134}$$

for some composite Lipschitz constants a_9 and a_{10}. We may then combine (17), (51), (63), (78), (96), (118) and (134) to obtain the bound

$$\begin{aligned}
&\|[\nabla_{xx}\Psi^{\oplus} - H^{\oplus}]_{[\mathcal{F}_1,\mathcal{F}_1]}p^{\oplus}_{[\mathcal{F}_1]}\|_g \\
&\quad\leq [\upsilon[(4a_0a_6/\pi_{\min})(\mu_{\min})^{\alpha_\eta-1+k\alpha_\lambda\beta_\eta}]^{\varsigma} \\
&\qquad\quad + a_9[a_x(\mu_{\min})^{\alpha_\eta+k\alpha_\lambda\beta_\eta} + a_0\omega_0(\mu_{\min})^{\alpha_\omega+k\beta_\omega}] \\
&\qquad\quad + a_{10}a_{\lambda^{\oplus}}(\mu_{\min})^{\alpha_\eta-1+k\alpha_\lambda\beta_\eta}](4a_0a_6/\pi_{\min})(\mu_{\min})^{\alpha_\eta-1+k\alpha_\lambda\beta_\eta}
\end{aligned} \tag{135}$$

for all sufficiently large k. Lastly, the third term on the right-hand-side of (132) satisfies the bound

$$\|\int_0^1 [\nabla_{xx}\Psi(x^{\oplus}(t),\lambda^+,s^+) - \nabla_{xx}\Psi^{\oplus}_{[\mathcal{F}_1,\mathcal{F}_1]}p^{\oplus}_{[\mathcal{F}_1]}]dt\|_g \leq \tfrac{1}{2}a_0a_8\|p^{\oplus}_{[\mathcal{F}_1]}\|_g^2. \tag{136}$$

by the same arguments we used to establish inequality (127). We may then combine (118) and (136) so that

$$\begin{aligned}
&\|\int_0^1 [\nabla_{xx}\Psi(x^{\oplus}(t),\lambda^+,s^+) - \nabla_{xx}\Psi^{\oplus}]_{[\mathcal{F}_1,\mathcal{F}_1]}p^{\oplus}_{[\mathcal{F}_1]}dt\| \\
&\quad\leq 8a_0^3a_6^2a_8(\mu_{\min})^{2\alpha_\eta-2+k2\alpha_\lambda\beta_\eta}/\pi_{\min}^2
\end{aligned} \tag{137}$$

for all k sufficiently large.

We now combine equation (132) with the inequalities (133), (137) and (135), the condition $\xi < 1$ and the definitions of $\alpha_\eta(< 1)$ and $\beta_\eta(> 0)$ to obtain the bound

$$\|\nabla_x\Psi(x^+,\lambda^+,s^+)_{[\mathcal{F}_1]}\| \leq a_{11}(\mu_{\min})^{\bar{\alpha}+k\bar{\beta}}. \tag{138}$$

where

$$\bar{\alpha} = (\alpha_\eta - 1)(1 + \max(1,\varsigma)), \quad \bar{\beta} = \alpha_\lambda\beta_\eta(1 + \min(\xi,\varsigma)) \tag{139}$$

and

$$a_{11} = a_6^{1+\xi} + 8a_0^3a_6^2a_8/\pi_{\min}^2 + (4a_0a_6/\pi_{\min})(\upsilon((4a_0a_6/\pi_{\min})^\varsigma) + a_9(a_x + a_0\omega_0) + a_{10}a_\lambda). \tag{140}$$

Firstly, observe that the right-hand-side of (138) may be made arbitrarily small. Therefore, (13), (131) and (138) imply that

$$\|P(x^+, \nabla_x\Psi(x^+, \lambda^+, s^+))\|_g = \|\nabla_x\Psi(x^+, \lambda^+, s^+)_{[\mathcal{F}_1]}\|_g \leq a_{11}(\mu_{\min})^{\bar{\alpha}+k\bar{\beta}}. \tag{141}$$

Secondly, define $\delta = \log_{\mu_{\min}}(a_{11}/\omega_0)$. Now let k_1 be any integer for which

$$k_1 \geq \frac{\alpha_\omega + \beta_\omega - \bar{\alpha} - \delta}{\bar{\beta} - \beta_\omega}. \tag{142}$$

Then (58), (141) and (142) imply that

$$\|P(x^+, \nabla_x\Psi(x^+, \lambda^+, s^+))\|_g \leq a_{11}(\mu_{\min})^{\bar{\alpha}+k\bar{\beta}} \leq \omega_0(\mu_{\min})^{\alpha_\omega+(k+1)\beta_\omega} = \omega^+ \tag{143}$$

for all sufficiently large $k \geq k_1$. Thus, the iterate x^+ satisfies the inner iteration first convergence test of (21) for all k sufficiently large and we have $x^{(k+1)} = x^{(k+1,1)} \equiv x^+$.

(vi) Redundancy of the shifted starting point. Finally, we observe that all the variables $x_j^{(k)}$, $j \in \mathcal{D}$, lie on their bounds for sufficiently large k. Therefore, $x^{(k+1,0)} = x^{(k)}$ and the perturbed starting point is redundant. ■

5. The general case

We now turn briefly to the more general problem (1)—(3). The presence of the more general bounds (3) does not significantly alter the conclusions that we are able to draw. The algorithms of section 3 are basically unchanged. We now use the region $\mathcal{B} = \{x \in \Re^n \mid l \leq x \leq u\}$ — and hence $\mathcal{N}_b = \mathcal{N}$ — and replace $P(x, v)$ by $P(x, v, l, u)$ where appropriate. The concept of floating and dominated variables stays essentially the same. For each iterate in $\mathcal{B}$ we have three mutually exclusive possibilities, namely, (i) $0 \leq x_j^{(k)} - l_j \leq (\nabla_x\Psi^{(k)})_i$, (ii) $(\nabla_x\Psi^{(k)})_i \leq x_j^{(k)} - u_j \leq 0$ or (iii) $x_j^{(k)} - u_j < (\nabla_x\Psi^{(k)})_i < x_j^{(k)} - l_j$, for each component $x_j^{(k)}$. In case (i) we then have that $P(x^{(k)}, \nabla_x\Psi^{(k)}, l, u)_i = x_j^{(k)} - l_j$ while in case (ii) $P(x^{(k)}, \nabla_x\Psi^{(k)}, l, u)_i = x_j^{(k)} - u_j$ and in case (iii) $P(x^{(k)}, \nabla_x\Psi^{(k)}, l, u)_i = (\nabla_x\Psi^{(k)})_i$. The variables that satisfy (i) and (ii) are said to be the dominated variables, the ones satisfying (i) are *dominated above* while those satisfying (ii) are *dominated below*. Consequently, the sets corresponding to (14) are straightforward to define. $\mathcal{D}_1$ is now made up as the union of two sets $\mathcal{D}_{1l}$, whose variables are dominated above for all k sufficiently large, and $\mathcal{D}_{1u}$, whose variables are dominated below for all k sufficiently large. $\mathcal{F}_1$ contains

variables which float for all k sufficiently large and which converge to values interior to $\mathcal{B}$. Similarly $\mathcal{F}_2$ is the union of two sets, $\mathcal{F}_{2l}$ and $\mathcal{F}_{2u}$, whose variables are floating for all k sufficiently large but which converge to their lower and upper bounds respectively. We also replace (32) by

$$\hat{x}_j^{(k-1)} = \begin{cases} l_j & \text{if } 0 \leq x_j^{(k-1)} - l_j \leq \theta(\nabla_x \Psi^{(k-1)})_j \\ u_j & \text{if } \theta(\nabla_x \Psi^{(k-1)})_j \leq x_j^{(k-1)} - u_j \leq 0 \\ x_j^{(k-1)} & \text{otherwise.} \end{cases} \tag{144}$$

With such definitions, we may reprove the results of section 4, extending AS4, AS7—AS9 in the obvious way. The only important new ingredient is that Conn *et al.* [9] indicate that the non-degeneracy assumption AS7 ensures that the iterates are asymptotically isolated in the three sets $\mathcal{F}_1$, $\mathcal{D}_{1l}$ and $\mathcal{D}_{1u}$.

6. Conclusions

We have shown that, under suitable assumptions, a single inner iteration is needed for each outer iteration of the Lagrangian barrier algorithm. We anticipate that such an algorithm may prove to be an important ingredient of release B of the LANCELOT package.

7. Acknowledgement

The work reported here has been partly supported by the NATO travel grant CRG 890867.

References

1. D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, London, 1982.
2. J. V. Burke and J. J. Moré. On the identification of active constraints. *SIAM Journal on Numerical Analysis*, 25(5):1197–1211, 1988.
3. J. V. Burke, J. J. Moré, and G. Toraldo. Convergence properties of trust region methods for linear and convex constraints. *Mathematical Programming, Series A*, 47(3):305–336, 1990.
4. P. H. Calamai and J. J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39:93–116, 1987.
5. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25:433–460, 1988. See also same journal 26:764–767, 1989.
6. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430, 1988.
7. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.
8. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds. In D.F Griffiths and G.A. Watson, editors, *Proceedings of the 14th Biennal Numerical Analysis Conference Dundee 1991*, pages 49–68. Longmans, 1992.

9. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Mathematics of Computation*, volume 66, pages 261-288, 1997.
10. J. E. Dennis and J. J. Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Mathematics of Computation*, 28(126):549–560, 1974.
11. J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall, Englewood Cliffs, USA, 1983.
12. A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. J. Wiley and Sons, New York, 1968. Reprinted as *Classics in Applied Mathematics 4*, SIAM, 1990.
13. N. I. M. Gould. On the accurate determination of search directions for simple differentiable penalty functions. *IMA Journal of Numerical Analysis*, 6:357–372, 1986.
14. N. I. M. Gould. On the convergence of a sequential penalty function method for constrained minimization. *SIAM Journal on Numerical Analysis*, 26:107–128, 1989.
15. J. J. Moré. Trust regions and projected gradients. In M. Iri and K. Yajima, editors, *System Modelling and Optimization*, volume 113, pages 1–13, Berlin, 1988. Springer Verlag. Lecture Notes in Control and Information Sciences.
16. Ph. L. Toint. Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space. *IMA Journal of Numerical Analysis*, 8:231–252, 1988.
17. M. H. Wright. Interior methods for constrained optimization. volume 1 of *Acta Numerica*, pages 341–407. Cambridge University Press, New York, 1992.

Computational Optimization and Applications, 7, 71–87 (1997)

Numerical Experiences with New Truncated Newton Methods in Large Scale Unconstrained Optimization*

STEFANO LUCIDI lucidi@iasi.rm.cnr.it
MASSIMO ROMA roma@dis.uniroma1.it
Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy

Abstract. Recently, in [12] a very general class of truncated Newton methods has been proposed for solving large scale unconstrained optimization problems. In this work we present the results of an extensive numerical experience obtained by different algorithms which belong to the preceding class. This numerical study, besides investigating which are the best algorithmic choices of the proposed approach, clarifies some significant points which underlies every truncated Newton based algorithm.

Keywords: Large scale unconstrained optimization, Truncated Newton methods, negative curvature direction, curvilinear linesearch, Lanczos method.

1. Introduction

This work deals with a new class of algorithms for solving large scale unconstrained problems. We consider the minimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1}$$

where $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is a real valued function and we assume that the gradient $g(x) = \nabla f(x)$ and the Hessian matrix $H(x) = \nabla^2 f(x)$ exist and are continuous. We are interested in solving Problem (1) when the dimension n is large. This interest derives from the fact that problems with larger and larger number of variables are arising very frequently from real world applications. Moreover, besides its own interest, the definition of efficient algorithms for solving Problem (1) may be also considered an essential starting point to tackle large scale constrained minimization problems.

As well known, the knowledge of the Hessian matrix enables to significantly exploit more information about the problem than those available merely from the gradient. This is clearly evidenced by the efficiency of Newton-type methods. For wide classes of unconstrained optimization problems the Hessian matrix is available, but, unfortunately, when the dimension is large, it can not be stored and the exact computation of the Newton direction can be too expensive. For these classes of problems it is appropriate to use a truncated Newton approach. In fact, the algorithms which follow this approach use an approximate solution of the Newton equation

* This work was partially supported by Agenzia Spaziale Italiana, Roma, Italy.

$$H(x_k)s = -g(x_k) \tag{2}$$

and require only the storage of matrix vector product of $H(x_k)$ with a suitable vector. Moreover, they present good convergence properties.

The most popular method used as iterative scheme for computing an approximate Newton-type direction is the conjugate gradient algorithm [4, 5, 19]. In most of all these truncated Newton methods, the inner conjugate gradient iterates are terminated when the desired accuracy is obtained or whenever a negative curvature direction (i.e. a vector v such that $v^T Hv < 0$) is detected.

Recently, a different strategy has been used by the truncated Newton method implemented in the LANCELOT software package [2]. In fact, as usual, the inner iterates are terminated whenever a negative curvature direction is detected, but, unlike the other truncated algorithms, this direction is exploited by performing a significant movement along it. This strategy enables this algorithm to take into account the local nonconvexity of the objective function and its beneficial effects are evidenced by the numerical behaviour of the algorithm.

However, the choice of terminating the inner conjugate gradient iterations whenever a negative curvature is found can present the following drawbacks:

- the accuracy of the solution of (2) could be very poor when the iterates of the conjugate gradient method are terminated since a negative curvature has been detected;
- the first negative curvature direction detected by the iterative scheme could not to have sufficient information on the local nonconvexity contained in the Hessian matrix.

An attractive iterative method to compute an approximate solution of the Newton equation (2), alternative to the conjugate gradient method, is the Lanczos algorithm (see [15]). In fact this algorithm presents the following features [3, 8]:

- it does not break down when the Hessian matrix is indefinite and hence, whenever system (2) is solvable, an accurate Newton-type direction can be computed;
- it enables to compute efficiently a good approximation of the smallest eigenvalue (and the corresponding eigenvector) of the Hessian matrix and hence a negative curvature direction with significant information on the local nonconvexity of the function can be obtained.

In [12], by drawing inspiration from [10, 13, 14], a new class of the truncated Newton methods has been proposed. A common feature of the methods belonging to this class, is the use of the Lanczos algorithm for determining both a Newton-type direction s_k and a negative curvature direction d_k. This pair of directions is used for defining a curvilinear search path

$$x(\alpha) = x_k + \alpha^2 s_k + \alpha d_k.$$

Then, the new point is computed along this path by means of a very general nonmonotone stabilization strategy.

Many different algorithms can be derived from the class proposed in [12] and all these algorithms have, of course, the same theoretical properties. However, from the computational point of view, they can have very different behaviours and therefore only an extensive numerical experience can clarify which are the best algorithms within the proposed class.

The numerical results reported in [12] have already evidenced which general strategy should be adopted in order to define efficient algorithms. In fact, the conclusions drawn from those results have indicated that the best strategy is that of defining algorithms which are based on the joint use of negative curvature directions and nonmonotone stabilization techniques. Furthermore, the comparison between the results obtained in [12] and those obtained by different versions of the truncated Newton method implemented in the LANCELOT package, have evidenced that the approach proposed in [12] is very promising from the computational point of view.

In this work we continue the numerical investigation on the algorithmic choices concerning the class of algorithms proposed in [12]. First of all, we focus our attention on the most effective way of computing a good Newton-type direction by adopting different termination criteria of the inner Lanczos iterates. Then we analyse different strategies for calculating negative curvature direction and study different ways of evaluating the resemblance of the negative curvature direction to the eigenvector corresponding to the smallest eigenvalue of the Hessian matrix.

We point out that these numerical investigations, besides allowing us to understand better which are the best algorithmic choices within the class proposed in [12], give interesting answers to some open questions which are behind the truncated Newton approach and hence they give helpful hints for defining any new truncated Newton method.

The paper is organized as follows: in Section 2 the new method proposed in [12] is briefly reviewed. In Section 3 we report the results of our numerical experiences.

2. Description of the method

In this section we review some of the proposals of [12]. First of all, after having briefly recalling the truncated Newton methods proposed in that paper, we describe the particular algorithm which showed the best numerical behaviour in the computational testing reported in [12]. We refer to [12] for a detailed description and a rigorous analysis of this algorithm.

2.1. The algorithm model

Now we briefly recall the various parts that constitute the algorithm model proposed in [12].

Curvilinear search path

Following the curvilinear search approach of [13, 14], a sequence of point $\{x_k\}$ is produced according to the rule

$$x_{k+1} = x_k + \alpha_k^2 s_k + \alpha_k d_k \tag{3}$$

where s_k and d_k are search directions and α_k is a step length.

Computation of the search directions

The SYMMLQ algorithm by Paige and Saunders [17] is used for computing a truncated Newton direction s_k by approximately solving the system (2). This routine implements the Lanczos algorithm and, in particular, it produces a sequence of *Lanczos basis vector* v_i together with the scalars α_i and β_i. At the $i-$th step it can be also defined the matrix V_i whose columns are the vectors v_i and the tridiagonal matrix T_i such that $(T_i)_{h,l} = 0$ if $|h-l| > 1$, $(T_i)_{l,l} = \alpha_l$, for $l = 1, \ldots, i$ and $(T_i)_{l+1,l} = (T_i)_{l,l+1} = \beta_{l+1}$, for $l = 1, \ldots, i-1$.

As well known (see [3, 8, 18]), the minimum eigenvalue μ_i of the tridiagonal matrix T_i and the corresponding eigenvector w_i present interesting properties. In fact, as the number of Lanczos iterations increases, μ_i approximates better and better an eigenvalue λ_k of the Hessian matrix $H(x_k)$ and, under suitable assumptions, λ_k is just the smallest eigenvalue of the Hessian matrix. Correspondingly the vector $\widetilde{d}_i = V_i w_i$ bears better and better resemblance to the eigenvector associated to the eigenvalue λ_k.

The precise manner in which the directions s_k and d_k can be computed is outlined in the following scheme:

Lanczos based iterative Truncated Scheme (LTS)

Step 1: Initialization
Choose $-g(x_k)$ as the Lanczos starting vector and set $i = 1$.

Step 2: Lanczos iterations
Iterate the SYMMLQ algorithm until the *termination criterion* is satisfied.

Step 3: Computation of direction s_k

- If the system (2) is solvable and the current estimate $\widetilde{s}_i$ of the solution of the system (2) produced by SYMMLQ routine satisfies the *convergence conditions*, then set $s_k = \widetilde{s}_i$
- otherwise set $s_k = -P_k g(x_k)$ where P_k is a strictly positive definite matrix.

Step 4: Computation of direction d_k
Compute the smallest eigenvalue μ_i of the tridiagonal matrix T_i.

- If $\mu_i < 0$ then compute the corresponding eigenvector w_i and set

$$d_k = -\rho_k \text{sgn}\left[g(x_k)^T \widetilde{d}_i\right] \widetilde{d}_i \quad \text{where} \quad \widetilde{d}_i = V_i w_i \tag{4}$$

- otherwise set $d_k = 0$.

A key point of this truncated scheme LTS is the convergence criterion at Step 2. In fact, this test should ensure that the Lanczos algorithm is terminated at the $i-$th iteration provided that both the estimate of the solution of the Newton system $\widetilde{s}_i$ is sufficiently accurate and the smallest eigenvalue μ_i of the tridiagonal matrix T_i is a good approximation of an eigenvalue of $H(x_k)$. The accuracy of the solution of the Newton system (2) can be evaluated by the magnitude of the residual at each step. As regards μ_i, its difference from an eigenvalue of $H(x_k)$ is bounded by the scalar β_{i+1} which, hence, can be used as a measure of the accuracy of μ_i in approximating an eigenvalue of the Hessian matrix [18].

Stabilization algorithm

As regards the computation of step length α_k, the nonmonotone globalization strategy proposed in [7, 10] is adopted. The motivation of this choice is to try to accept as many times as possible the unit stepsize whenever the iterates of the algorithm are in a region where Newton method present strong convergence properties. This stabilization algorithm is now recalled.

Nonmonotone Stabilization Algorithm (NSA)

Data: x_0, $\Delta_0 > 0$, $\delta \in (0,1)$, $N \geq 1$, $M \geq 0$, $\sigma \in (0,1)$ and $\delta \in (0, \frac{1}{2})$.

Step 1: Set $k = \ell = j = 0$, $\Delta = \Delta_0$. Compute $f(x_0)$ and set $Z_0 = F_0 = f(x_0)$, $m(0) = 0$.

Step 2: Compute $g(x_k)$. If $\|g(x_k)\| = 0$ stop.

Step 3: If $k \neq \ell + N$ compute directions s_k and d_k by LTS algorithm; then:

(a) if $\|s_k\| + \|d_k\| \leq \Delta$, set $\alpha_k = 1$, compute x_{k+1} according to (3), set $k = k + 1$, $\Delta = \delta\Delta$ and go to Step 2;

(b) if $\|s_k\| + \|d_k\| > \Delta$, compute $f(x_k)$;
if $f(x_k) \geq F_j$, replace x_k by x_ℓ, set $k = \ell$ and go to Step 5;
otherwise set $\ell = k$, $j = j + 1$, $Z_j = f(x_k)$ and update F_j according to

$$F_j = \max_{0 \leq i \leq m(j)} Z_{j-i}, \quad \text{where} \quad m(j) \leq \min[m(j-1) + 1, M] \tag{5}$$

and go to Step 5.

Step 4: If $k = \ell + N$ compute $f(x_k)$; then:

(a) if $f(x_k) \geq F_j$, replace x_k by x_ℓ, set $k = \ell$ and go to Step 5;

(b) if $f(x_k) < F_j$, set $\ell = k$, $j = j + 1$, $Z_j = f(x_k)$ and update F_j according to (5). Compute directions s_k and d_k by LTS algorithm;
if $\|s_k\| + \|d_k\| \leq \Delta$, set $\alpha_k = 1$, compute x_{k+1} according to (3), set $k = k + 1$, $\Delta = \delta\Delta$ and go to Step 2;
otherwise go to Step 5.

Step 5: Compute $\alpha_k = \sigma^h$ where h is the smallest nonnegative integer such that

$$f(x_k + \alpha_k^2 s_k + \alpha_k d_k) \leq F_j + \gamma \alpha_k^2 \left[g(x_k)^T s_k + \frac{1}{2} d_k^T H(x_k) d_k \right], \tag{6}$$

compute x_{k+1} according to (3), set $k = k + 1$, $\ell = k$, $j = j + 1$, $Z_j = f(x_k)$, update F_j according to (5) and go to Step 2.

The usefulness of a nonmonotone strategy in solving "difficult" problems (such as highly nonlinear and/or ill conditioned problems) has been evidenced in many papers. For example, [10, 11, 21] in the context of the linesearch algorithms and [6, 22, 23] in the context of trust region algorithms.

Without going into details the key idea of a nonmonotone based algorithm consists in not forcing the decrease of the objective function values sequence $f(x_k)$ at each iteration; in fact a step could be automatically accepted without evaluating the objective function at the new point (see Step 3) provided that a test on the reduction rate of the norms of the directions is verified. Such test is performed by means of a decreasing parameter Δ, which represents a prescribed bound of the actual steplength, with the aim to evaluate whether the iterates are converging or if convergence has to be enforced by linesearch procedure. In fact, if the test is not satisfied, a nonmonotone modified Armjio-tpye curvilinear linesearch is used (Step 5). The rationale behind this strategy is that the directions s_k and d_k convey significant information on the objective function and that the decrease of their magnitude is, usually, an indication that the algorithm is converging towards a critical point.

As regards the nonmonotone linesearch procedure, in the generalized Armjio-type condition (6), instead of using $f(x_k)$ as reference value, an objective function value F_j corresponding to a previous iterate is considered. This value F_j is given by (5) and represents the maximum function value over a prescribed number of previous iterations corresponding to accepted points where the objective function has been evaluated (these objective function values are denoted by Z_j). The adjustable reference value F_j is also used to evaluate if the new iterate leads to regions where the function is poorly behaved; in fact, in this case, a backtracking scheme is incorporated to restart the algorithm from the iterate corresponding to the last accepted point.

2.2. *Convergence analysis*

The convergence properties of this algorithm have been studied in [12]. In particular, assuming that for a given x_0, the level set $\Omega_0 = \{x \in \mathbb{R}^n | f(x) \leq f(x_0)\}$ is compact, the algorithm is globally convergent towards a stationary point if the direction s_k and d_k are bounded and satisfy the following *convergence conditions*:

$$g(x_k)^T s_k \leq 0, \qquad g(x_k)^T d_k \leq 0, \qquad d_k^T H(x_k) d_k \leq 0,$$
$$g(x_k)^T s_k \to 0 \quad \text{implies} \quad g(x_k) \to 0 \quad \text{and} \quad s_k \to 0,$$
$$\|s_k\| + \|d_k\| \to 0 \quad \text{implies} \quad g(x_k) \to 0.$$

If, in addition, the direction d_k satisfies also

$$\begin{cases} d_k^T H(x_k) d_k \to 0 \\ g(x_k) \to 0 \end{cases} \quad \text{implies} \quad \begin{cases} \min\left[0, \lambda_{\min}\left(H(x_k)\right)\right] \to 0 \\ d_k \to 0 \end{cases}$$

(where $\lambda_{\min}\left(H(x_k)\right)$ is the minimum eigenvalue of the Hessian matrix) then it is possible to show that the preceding algorithm is globally convergent towards stationary points where the Hessian matrix is positive semidefinite [12, Theorem 2.1]. We point out that the directions s_k and d_k used in the algorithm (NSA) are computed (as described in Section 2.1) in such a way as to satisfy the conditions for convergence.

2.3. *Implementation details*

In [12] a preliminary computational experience has been performed in order to investigate the numerical behaviour of some algorithms belonging to the proposed class. The aim of that numerical testing has been to understand the effect of the use of the negative curvature direction and of the nonmonotone stabilization strategy. The results obtained confirmed that the joint use of negative curvature directions and nonmonotone stabilization strategy can be considered a very promising tool for defining new efficient truncated Newton algorithms.

In particular, the algorithm called **NMonNC** has showed the best computational behaviour in the numerical investigations performed in [12] and it is characterized by the following algorithmic choices (we refer to [12] for a discussion on the rationale behind these choices). First of all, we consider the parameters which appear in the truncated scheme LTS: the upper limit on the number of the SYMMLQ iterations is set to n and the user-specified tolerance required by the SYMMLQ routine is set to the value

$$rtol = \tau \min\left\{1, \|g(x_k)\| \max\left(\frac{1}{k+1}, \frac{1}{\exp\left(\frac{k}{\tau n}\right)}\right)\right\} \tag{7}$$

where $\tau = 10^{-1}$. Moreover, as regards the *termination criterion* at Step 2 of LTS, the SYMMLQ routine has been slightly modified in order to continue to perform Lanczos iterations until the i–th iteration where both one of the original stopping criteria of SYMMLQ routine is fulfilled and one of this additional criteria is satisfied:

$$\beta_{i+1} < \epsilon_k \qquad \text{or} \qquad i > L \tag{8}$$

where $\epsilon_k = 10^3 \max\left\{\|g(x_k)\|, \frac{1}{k+1}\right\}$ and $L = 50$ is an upper bound on the number of the vectors stored in the matrix V_i. Furthermore the following value of the scaling factor ρ_k in (4) is used

$$\rho_k = \sqrt{-\mu_i}\ \min\left\{1, \frac{1}{\|g_k\|}\right\} \tag{9}$$

where μ_i is the smallest eigenvalue of the matrix T_i produced by the Lanczos algorithm.

In the nonmonotone stabilization algorithm, the following values for its parameters have been adopted: $\Delta_0 = 10^3$, $N = 20$, $M = 20$, $\delta = 0.9$, $\sigma = 0.5$; moreover whenever a backtracking is performed (see Step 3 (b) and Step 4 (a)), then the current values of Δ and M are modified as follows: $\Delta = 10^{-1}\Delta$ and $M = M/5 + 1$.

3. Numerical experiences

In this section we continue the numerical investigation started in [12] with the aim to further study the effectiveness of different algorithmic choices concerning the class of algorithms proposed. As we said in the introduction, the results of this numerical study can be of interest also in a more wide context. In fact, some of the indications drawn from this investigation, can be useful for defining, in general, any truncated Newton method.

As concerns the reported numerical experiences, we have considered as default choices those of algorithm **NMonNC** described in Section 2.3. The following numerical investigation has been based on the use of a large set of test problems. In particular we have used all the large scale unconstrained test problems available from the CUTE collection [1]; all the problems with a number of variables which ranges between 930 and 10000 have been selected providing us with a test set of 98 problems.

All tests have been performed on an IBM RISC System/6000 375 under AIX using Fortran in double precision with the default optimization compiling option. All the runs have been terminated when the convergence criterion $\|g\| \leq 10^{-5}$ has been fulfilled.

In the comparisons between different algorithms which will be reported in the sequel, we consider all the test problems coherently solved by all these algorithms, namely all the problems where these algorithms converge to the same point. Moreover, we consider equal the results of two runs if they differ by at most of 5%. Finally, we consider as failure all the runs which need more than 5000 iterations or 5000 seconds of CPU time.

Since the results of all the runs consist in many tables, in the sequel we report only the summaries of this extensive numerical testing together with some statistical comparisons.

3.1. Computation of the Newton-type direction s_k

First of all, we perform a numerical investigation on the influence of the accuracy of the Newton-type direction within a general truncated Newton algorithm. More in particular, we investigate on the following questions:

1. is it worthwhile to continue the inner iterations of a truncated scheme whenever a direction of negative curvature is detected ?

2. which is the best value for the tolerance $rtol$ used by the SYMMLQ routine ?

Question 1. In most of the classical truncated Newton methods the conjugate gradient algorithm is used for computing an approximate Newton-type direction, and the conjugate gradient iterations are terminated when a desired accuracy is achieved or whenever a negative curvature direction is detected. In this second case the estimate generated at the previous iteration is often accepted as approximate solution of (2) even if a sufficiently accuracy has been not reached. Hence, a possible drawback of this use of the conjugate gradient method is that the accuracy of the solution of (2) could be very poor when the iterates of the conjugate gradient method are terminated as a negative curvature has been found. In particular, in [9] was pointed out that it could be of beneficial effect to try to continue the

inner conjugate gradient iterations even if the Hessian matrix is not positive definite, and hence to try to compute, also in this case, a sufficiently good solution of (2).

Since the Lanczos based iterative scheme LTS does not break down when the Hessian matrix is not positive definite, our algorithm model, represents a useful and flexible tool for investigating on this aspect which is, of course, one the most important for the definition of an effective truncated Newton algorithm. In particular, we consider some modifications of default implementation of NSA (algorithm **NMonNC**) *which do not use the negative curvature direction* (i.e. we set $d_k = 0$) in order to focus our attention only on the influence on the efficiency of the algorithm with respect to different strategies adopted for computing the truncated Newton direction. In particular, we consider three different versions of NSA where the only difference consists in the termination criterion at the Step 2 of the LTS scheme. In fact, the following different criteria are used:

Algorithm **A**: the inner iterations are terminated whenever the original stopping criteria of SYMMLQ routine are satisfied or a negative curvature direction is detected.

Algorithm **B**: the inner iterates continue until the termination criteria of SYMMLQ routine are satisfied.

Algorithm **C**: the inner iterates continue until the original tests of SYMMLQ routine and one of the criteria (8) are satisfied.

In the algorithm **A** the criterion commonly used in the truncated Newton method is adopted. The termination criterion of algorithm **B** differs from that adopted by algorithm **A** since the inner iterates are not terminated whenever a direction of negative curvature is detected and this should enable to compute a more accurate approximation of the Newton direction. The choice of algorithm **C** is the same used in the default implementation described in Section 2.3. In that implementation this test is needed to ensure the goodness of the negative curvature direction; here, since negative curvature directions are not used, the numerical behaviour of algorithm **C**, should indicate the influence on the computation of the truncated Newton direction s_k of continuing the inner iterates even if a sufficiently small residual in the solution of (2) has been obtained. The summary of the results obtained by these three algorithms on the whole test set is summarized in the following tables. In particular, in Table 1 we report the cumulative results of this comparison that is the total number of iterations, function and gradient evaluations, CPU time needed to solve all the problems considered. Table 2 reports how many times each of the three algorithms is the best, second and worst in terms of number of iterations, function and gradient evaluations and CPU time.

By comparing the results obtained by algorithm **A** and those obtained by algorithms **B** and **C** it appears clear that the computation of an accurate Newton-type direction also when the Hessian is not positive definite can improve significantly the efficiency of the algorithm. This is confirmed also by observing the results obtained by algorithms **B** and **C**. In fact, the best results have been obtained by algorithm **C** where the estimate $\widetilde{s}_i$ is considered a good Newton-type direction only when, besides being a good approximation of the solution of the system (2), it conveys sufficient information on the curvature of the objective function. This is obtained by using the additional criterion (8) which ensures that the iterates of SYMMLQ routine continue until a small scalar β is produced by the Lanczos algorithm or until the

Table 1. Cumulative results with different termination criteria

	Iter	Funct	Grad	Time
A	7371	12438	7475	11487.56
B	5466	5324	5562	4545.54
C	4918	4804	5014	4424.14

Table 2. Comparative ranking for the algorithms **A**, **B**, and **C**

	algorithm	1st	2nd	3rd
ITERATIONS	**A**	76	5	15
	B	84	12	0
	C	87	7	2
FUNCTIONS	**A**	72	4	20
evaluations	**B**	81	14	1
	C	85	9	2
GRADIENTS	**A**	76	5	15
evaluations	**B**	84	12	0
	C	87	7	2
CPU time	**A**	74	6	16
	B	78	18	0
	C	78	12	6

number of inner iterations is greater than the prefixed upper limit L. Therefore, this last consideration shows that this additional test (8) needed in the default algorithm **NMonNC** to compute a sufficiently good negative curvature direction d_k, has a beneficial influence also in the computation of the Newton-type direction.

Question 2. As well known, the value of the tolerance in approximately solving the Newton equation (2) is a key point for the efficiency of every truncated Newton method and, hence, an empirical tuning of the parameter $rtol$ is surely needed. Of course, since there are so many different choices for this parameter, it is out of the scope of this work to give any conclusive answer. Here we have only performed some numerical experiences for testing different choices of the parameter $rtol$ with respect to the value (7) used in algorithm **NMonNC**. In particular, we have investigated on the use of some values which draw their inspiration from some proposals widely used in literature. More specifically, in algorithm **D**, the inner iterates are interrupted when the original stopping criteria of SYMMLQ are satisfied (same strategy as algorithm **B**) and, following [5], the tolerance parameter is set to the value

$$rtol = \min \|g(x_k)\| \left\{ \|g(x_k)\|, \frac{1}{k+1} \right\}.$$

Similarly, in algorithm **E**, following [16], we use for the parameter $rtol$ the value

$$rtol = 10^{-8} \|g(x_k)\|$$

together with a bound on the maximum number of the iterates of LTS scheme set to $\min\{n, 500\}$. In both the algorithms **D** and **E** the negative curvature direction is not considered. A comparison between the results obtained by algorithm **B** which uses the

Table 3. Cumulative results for the Algorithms **D** and **B**

	Iter	Funct	Grad	Time	*failures*
D	6257	6913	6350	13925.2	3
B	5415	5301	5508	4490.7	0

Table 4. Number of times each algorithm **D** and **B** performs the best

	Iter	Funct	Grad	Time
D	15	9	15	15
B	36	32	36	49
tie	42	52	42	29

value given by (7) and algorithm **D** is reported in Table 3 and in Table 4 is reported the number of times each algorithm performs the best. Note that algorithm **D** showed three failures which are not considered in these cumulative results. Now we reports a summary of the comparison between the results obtained by algorithms **B** and **E**. Table 5 reports the

Table 5. Cumulative results for the Algorithms **E** and **B**

	Iter	Funct	Grad	Time	*failures*
E	3862	6906	3953	14079.83	4
B	4376	3878	4467	1825.66	0

cumulative results and Table 6 the number of times each algorithm performs the best. Note that algorithm **E** showed four failures. By observing Table 3, Table 4, Table 5 and Table 6, it appears clear that the best choice is that one adopted by algorithm **B** both in terms of efficiency and in terms of robustness. More in details, Table 3 and Table 4 clearly show that algorithm **B** outperforms algorithm **D** while Table 5 and Table 6 indicate that algorithm **E** is efficient in terms of number of iterations and also in terms of gradient evaluations.

Table 6. Number of times each algorithm **E** and **B** performs the best

	Iter	Funct	Grad	Time
E	35	26	35	15
B	27	30	27	54
tie	29	35	29	22

3.2. *Computation of the negative curvature direction d_k*

In this section, we perform a numerical study on the effect that different negative curvature directions have on the behaviour of a truncated Newton algorithm. Since the use of negative curvature directions in large scale optimization is relatively new, up to now, few investigations have been carried out on the sensitivity of an algorithm as the computation of negative curvature direction varies. The aim of the numerical experiences reported in this section is to shed some light on the following questions:

1. is it convenient to compute a "good" negative curvature direction ?
2. how can the "goodness" of a negative curvature direction be tested ?
3. which is the influence of the bound L on the Lanczos based vectors stored ?

Question 1. In the field of large scale minimization methods, roughly speaking, two alternative strategies for computing negative curvature directions have been proposed. The first one derives from the LANCELOT algorithm [2] which, as we said in the introduction, uses a negative curvature which is very cheap to compute but that could not to have significant information on the local nonconvexity of the objective function. The other one is based on the use of negative curvature directions which are more expensive to compute but that can be considered "better" than the previous ones in the sense that they have a "good" resemblance to the eigenvectors of the Hessian matrices corresponding to the most negative eigenvalues. This last strategy has been followed by algorithm **NMonNc** proposed [12] where the iterates of SYMMLQ algorithm are continued until a sufficient information on the smallest eigenvalue of the Hessian matrix (which can be tested by controlling the magnitude of β_{i+1}) is obtained or until the upper bound on number of Lanczos basis vectors stored is achieved (see criterion (8)). The numerical results reported in [12] seem to indicate that the second strategy is more efficient. However, this conclusion is influenced by the fact that the LANCELOT algorithm and the algorithm **NMonNc** are very different: in fact the method implemented in the LANCELOT package is a trust region Newton method where the search directions are computed by means of the conjugate gradient algorithm while algorithm **NMonNC** follows a curvilinear linesearch approach and uses the Lanczos algorithm for computing the search directions. Therefore, in order to investigate better on the effect of computing "accurate" search directions, we have implemented an algorithm (denoted by algorithm **F**) which, within the class of methods proposed in [12], draws its

inspiration from the strategy adopted by the LANCELOT algorithm. More in particular, algorithm **F** is characterized as follows:

Algorithm **F**: the inner Lanczos iterations are terminated whenever the original stopping criteria of SYMMLQ are satisfied or a negative curvature direction is detected and *the direction of negative curvature is used.*

Therefore, algorithm **NMonNC** described in Section 2.3 differs from algorithm **F** only in the termination criterion of the inner iterates. In Table 7 we report the cumulative results obtained by these two algorithms and in Table 8 we report the number of times each algorithm **F** and **NMonNC** performs the best. Table 7 and Table 8 clearly show that algo-

Table 7. Cumulative results for the Algorithms **F** and **NMonNC**

	Iter	Funct	Grad	Time
F	7439	11033	7536	10394.87
NMonNC	4964	5209	5061	3840.17

Table 8. Number of times each algorithm **F** and **NMonNC** performs the best

	Iter	Funct	Grad	Time
F	9	7	9	15
NMonNC	19	23	19	18
tie	69	67	69	64

rithm **NMonNC** ouperforms algorithm **F** in terms of number of iterations, function and gradient evaluations and in CPU time.

However, the different strategies adopted by algorithms **F** and **NMonNC** to terminate the SYMMLQ routine influence the computation of both the search directions, and hence both a "better" Newton type directions and a "more accurate" negative curvature directions are computed in algorithm **NMonNC**. As already outlined in Section 3.1, part of the efficiency of algorithm **NMonNC** is due to the use of more effective Newton directions. Now, in order to evaluate only the effect of using more accurate negative curvature directions we have implemented another algorithm **G** with the following features:

Algorithm **G**: the inner iterates continue until the original criteria of SYMMLQ routine are satisfied and one of the criteria (8) is fulfilled but the negative curvature used is *the first negative curvature direction detected.*

Therefore, algorithm **G** and algorithm **NMonNC** uses the same Newton type direction and they differ only in the negative curvature direction. We have compared the numerical

behaviour of these algorithms, on 14 test problems which are the only ones where negative curvature directions are detected and the two algorithms perform differently. In Table 9 we report the cumulative results obtained by both the algorithms and in Table 10 we report the number of times each algorithm **G** and **NMonNC** performs the best on these 14 problems. Table 9 and Table 10 seem to indicate that the use of "better" negative curvature direction has a clear beneficial effect as regards number of iterations, gradient evaluations and as regards CPU time. More questionable is the comparison between these two algorithms in

Table 9. Cumulative results for the Algorithms **G** and **NMonNC**

	Iter	Funct	Grad	Time
G	2955	3597	2969	2711.76
NMonNC	2854	3661	2868	2508.05

Table 10. Number of times each algorithm **G** and **NMonNC** performs the best.

	Iter	Funct	Grad	Time
G	0	3	0	0
NMonNC	9	8	9	9
tie	5	3	5	5

terms of function evaluations. In fact, Table 9 and Table 10 show that algorithm **NMonNC** performs better in most of the test problems but, on the other hand, in few test problems algorithm **G** allows a considerable saving in terms of function evaluations. Moreover, by observing more in detail the results obtained on the whole test set we note that in the only test problem (BROYDN7D) where the two algorithms converge towards different critical points, algorithm **NMonNC** is able to locate a point where the objective function value is lower as reported in Table 11.

Table 11. Detailed results of the problem BROYDN7D.

	Iter	Funct	Grad	Time	*function values*
G	87	168	88	6.89	.378021D+03
NMonNC	90	160	91	6.82	.368223D+03

Question 2. As concerns algorithms which use negative curvature directions, an important point is how to evaluate when a negative curvature direction is a "good" negative curvature direction. In the algorithm **NMonNC**, on the basis of the properties of the Lanczos algorithm (see [18]), we control the "goodness" of the direction of negative curvature by monitoring

the magnitude of the scalar β_{i+1} generated by the Lanczos algorithm until enough room is available. In literature a different criterion for evaluating the effectiveness of a negative curvature direction has been proposed in [20]. In order to compare these two criteria, we have implemented another algorithm (denoted by algorithm **H**) which, instead of using the criterion adopted in algorithm **NMonNC**, uses the test proposed in [20] which in our notation can be written

$$\frac{\left\|H(x_k)\widetilde{d}_i - \mu_i \widetilde{d}_i\right\|}{|\mu_i|\left\|\widetilde{d}_i\right\|} = \frac{\beta_{i+1}\left|e_i^T w_i\right|}{|\mu_i|\,\|w_i\|} \leq 0.1 \tag{10}$$

where $\widetilde{d}_i$ is given by (4) and w_i is the eigenvector of the tridiagonal matrix T_i corresponding to the smallest eigenvalue. In particular, algorithm **H** has the following features:

Algorithm **H**: the termination criterion of the inner iterates is the same of the algorithm **NMonNC** but the test on β_{i+1} in (8) is replaced by the test (10).

In Table 12 we reports the cumulative results obtained by algorithm **H** together with those obtained algorithm **NMonNC**. Note that algorithm **H** shows one failure. In Table 13 we

Table 12. Cumulative results for the Algorithms **H** and **NMonNC**

	Iter	Funct	Grad	Time	*failures*
H	4990	5952	5084	9893.90	1
NMonNC	4729	5060	4823	3357.71	0

report the number of times each algorithm **H** and **NMonNC** performs the best. On the

Table 13. Number of times each algorithm **H** and **NMonNC** performs the best.

	Iter	Funct	Grad	Time
H	24	14	23	10
NMonNC	12	21	12	48
tie	59	60	60	37

basis of Table 12 algorithm **NMonNC** appears the most effective. However, Table 13 shows that algorithm **H** is superior as regards the number of wins in terms of iterations and gradient evaluations. Therefore, the obtained results indicate that, probably, the best way to test the "goodness" of direction d_k is to use a criterion which a compromise between the criterion of algorithm **NMonNC** and the one proposed in [20].

Question 3. The computation of the negative curvature direction d_k (see (4)) requires the use of the matrix V_i whose columns are the Lanczos basis vectors. As the number of

the Lanczos vectors stored is larger as the direction d_k is a better approximation of an eigenvector of the Hessian matrix corresponding to a negative eigenvalue (see [8, 12, 18]). Due to the requirement of limited storage room, in algorithm **NMonNC** only L Lanczos vectors are stored. Therefore, we have performed a numerical study of the sensitivity of the algorithm as the parameter L varies in order to find suitable values of L which ensures a good efficiency of the algorithm without requiring an excessive storage room. In Table 14 we report the cumulative results of the algorithm **NMonNC** ($L = 50$) together with those obtained by the same algorithm with different value of the parameter L.

Table 14. Cumulative results with different values of L

L	ITER	FUNCT	GRAD	TIME
5	6210	6343	6308	5802.66
10	6206	6378	6304	4934.05
15	5922	6181	6020	4017.40
20	5783	6027	5881	4072.14
25	5644	5879	5742	4192.49
30	5371	5579	5469	3880.57
50	5054	5369	5152	3846.99
60	5001	5310	5099	4132.38
65	4945	5222	5043	3663.62
70	4950	5345	5848	4436.17
75	4922	5222	5020	3625.28
85	4903	5261	5001	4174.32
100	4884	5243	4982	4067.20
200	4897	5488	4995	6131.46
500	4786	5303	4884	8319.56

By observing this Table 14, it is clear that as L increases there is a substantial improvement of the behaviour of the algorithm in terms of number of iterations, function and gradient evaluations. On the other hand, when L is greater than 100, an excessive increase of CPU time is needed without producing a substantial improvement of the behaviour of the algorithm. In conclusion, the best choices seem to be $L \in [30, 75]$.

As concluding remark, we believe that a suitable scaling of the negative curvature direction d_k could play an important role for improving the efficiency of the algorithm. Therefore, this topic is worthy of an extensive study and will be the subject of future work.

Acknowledgments

We would like to thank Ph.L. Toint for helpful discussions and for many useful comments and suggestions.

References

1. I. Bongartz, A. Conn, N. Gould, and P. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Transaction on Mathematical Software*, 21:123–160, 1995.

2. A. Conn, N. Gould, and P. Toint. *LANCELOT: A Fortran package for Large-Scale Nonlinear Optimization (Release A)*. Springer Verlag, Heidelberg, Berlin, 1992.
3. J. Cullum and R. Willoughby. *Lanczos alghorithms for large symmetric eigenvalue computations*. Birkhauser, Boston, 1985.
4. R. Dembo, S. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19:400–408, 1982.
5. R. Dembo and T. Steihaug. Truncated-Newton methods algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.
6. N. Deng, Y. Xiao, and F. Zhou. Nonmonotonic trust region algorithm. *Journal of Optimization Theory and Applications*, 76:259–285, 1993.
7. M. Ferris, S. Lucidi, and M. Roma. Nonmonotone curvilinear linesearch methods for unconstrained optimization. *Computational Optimization and Applications*, 6: 117-136, 1996.
8. G. Golub and C. Van Loan. *Matrix Computations*. The John Hopkins Press, Baltimore, 1989.
9. L. Grippo, F. Lampariello, and S. Lucidi. A truncated Newton method with nonmonotone linesearch for unconstrained optimization. *Journal of Optimization Theory and Applications*, 60:401–419, 1989.
10. L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59:779–805, 1991.
11. G. Liu and J. Han. Convergence of the BFGS algorithm with nonmonotone linesearch. Technical report, Institute of Applied Mathematics, Academia Sinica, Beijing, 1993.
12. S. Lucidi, F. Rochetich, and M. Roma. Curvilinear stabilization techniques for truncated Newton methods in large scale unconstrained optimization: the complete results. Technical Report 02.95, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1995.
13. G. McCormick. A modification of Armijo's step-size rule for negative curvature. *Mathematical Programming*, 13:111–115, 1977.
14. J. Moré and D. Sorensen. On the use of directions of negative curvature in a modified Newton method. *Mathematical Programming*, 16:1–20, 1979.
15. S. Nash. Newton-type minimization via Lanczos method. *SIAM Journal on Numerical Analysis*, 21:770–788, 1984.
16. S. Nash and J. Nocedal. A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization. *SIAM Journal on Optimization*, 1:358–372, 1991.
17. C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.
18. B. Parlett. *The symmetric eigenvalue problem*. Prentice-Hall series in Computational Mathematics, Englewood Cliffs, 1980.
19. T. Schlick and A. Fogelson. TNPACK - A truncated Newton package for large-scale problems: I. algorithm and usage. *ACM Transaction on Mathematical Software*, 18:46–70, 1992.
20. G. Shultz, R. Schnabel, and R. Byrd. A family of trust-region-based algorithms for unconstrained minimization. *SIAM Journal on Numerical Analysis*, 22:47–67, 1985.
21. P. Toint. An assesment of non-monotone linesearch techniques for unconstrained optimization. SIAM Journal of Scientific Computing, 17: 725-739, 1996.
22. P. Toint. A non-monotone trust-region algorithm for nonlinear optimization subject to convex constraints. Technical Report 94/24, Department of Mathematics, FUNDP, Namur, Belgium, 1994.
23. Y. Xiao and F. Zhou. Nonmonotone trust region methods with curvilinear path in unconstrained minimization. *Computing*, 48:303–317, 1992.

Computational Optimization and Applications, 7, 89–110 (1997)

Sparse Linear Least Squares Problems in Optimization

PONTUS MATSTOMS pomat@math.liu.se
Department of Mathematics, Linköping University, S-581 82 Linköping, Sweden.

Abstract. Numerical and computational aspects of direct methods for large and sparse least squares problems are considered. After a brief survey of the most often used methods, we summarize the important conclusions made from a numerical comparison in MATLAB. Significantly improved algorithms have during the last 10-15 years made sparse QR factorization attractive, and competitive to previously recommended alternatives. Of particular importance is the multifrontal approach, characterized by low fill-in, dense subproblems and naturally implemented parallelism. We describe a Householder multifrontal scheme and its implementation on sequential and parallel computers. Available software has in practice a great influence on the choice of numerical algorithms. Less appropriate algorithms are thus often used solely because of existing software packages. We briefly survey software packages for the solution of sparse linear least squares problems. Finally, we focus on various applications from optimization, leading to the solution of large and sparse linear least squares problems. In particular, we concentrate on the important case where the coefficient matrix is a fixed general sparse matrix with a variable diagonal matrix below. Inner point methods for constrained linear least squares problems give, for example, rise to such subproblems. Important gains can be made by taking advantage of structure. Closely related is also the choice of numerical method for these subproblems. We discuss why the less accurate normal equations tend to be sufficient in many applications.

Keywords: Least squares problem, Sparse matrices, QR factorization, Multifrontal method

1. Introduction

Many scientific applications lead to the solution of large and sparse unconstrained linear least squares problems,

$$\min_x ||Ax - b||_2.$$

Typical areas of applications include chemistry, structural analysis and image processing. Sparse least squares problems are also common subproblems in large scale optimization.

Above, the coefficient matrix $A \in \mathbf{R}^{m \times n}$ is large and sparse with at least as many rows as columns ($m \geq n$), $b \in \mathbf{R}^m$ is a right-hand side vector and $x \in \mathbf{R}^n$ is the solution. Moreover, we assume that A has full column rank and that the nonzero entries are nonstructured. The full rank assumption of A makes $A^T A$ positive definite and the least squares solution uniquely determined. Much attention has during the last years been concentrated to sparse rank deficient problems. By rank revealing QR factorization (RRQR), the columns of A are permuted in such a way that the orthogonal factorization

$$Q^T(A\Pi) = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

makes $R_{11} \in \mathbf{R}^{r \times r}$ well-conditioned and $||R_{22}||_2$ small. Here, π is a permutation matrix and r is the numerical rank of A. Details are, for example, given by Björck [5].

In real applications sparse matrices usually have more than 10^5 rows and columns, but often less than 0.1% of nonzero entries. An extreme application, described by Kolata [34], is the adjustment of coordinates of North American geodetic stations. It results in mildly non-linear least squares problems of about 6.5 million equations and 540,000 unknowns. It should, however, be emphasized that the complexity of sparse matrix algorithms not only is determined by the dimension and degree of sparsity; also the sparsity pattern plays an important role.

The method of least squares dates back almost exactly 200 years. It was proposed as an algebraic procedure by Legendre [36] in 1805, and was at that time used for applications in astronomy. Gauss [18] later justified the method as a statistical procedure. He even claimed to have used the method of least squares since 1795, and should therefore be the "legitimate inventor". Later examinations of Gauss' processing of astronomical data suggest that he was right in his claim.

Let $Ax = b$ denote a linear system with more equations than unknowns. The method of least squares finds a "solution" x that minimizes the distance between the range of A and the right-hand side b. For consistent problems, where b belongs to the range of A, the residual $r = b - Ax$ becomes zero. A general residual r, corresponding to an optimal solution x, satisfies the orthogonality relation $A^T r = 0$.

The standard direct methods for least squares problems can, with respect to their underlying theoretical foundations, be classified into two groups: (a) those based on the orthogonality relation $A^T r = 0$ and (b) those based on the orthogonal invariance of the 2-norm. The first group includes the method of normal equations and the augmented system method, while methods based on QR factorization and SVD belong to the second group.

Sparse linear least squares problems are, as already mentioned, frequent in large scale optimization. Typical sources include constrained linear least squares problems, interior point methods for linear programming and nonlinear least squares problems. Often the underlying optimization problem is solved by the repeated solution of linear least squares problems, where the coefficient matrices are defined by a fixed sparse matrix and a variable diagonal matrix below. Let A be a given sparse matrix, for different values of the real parameter λ we then consider the solution of

$$\min_x \left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} x - \begin{pmatrix} b \\ c \end{pmatrix} \right\|_2. \tag{1}$$

The repeated problems can, by taking advantage of their special structure, be solved using previously computed information. If QR factorization is used, then we notice that the structure of the upper triangular factor $R(\lambda)$ is invariant under the choice of λ. Much of the overall computation (the symbolic analysis phase) do therefore not need to be repeated for each new subproblem.

Another interesting subproblem from optimization arises in the interior point solution of linear programs. Repeated large and sparse linear least squares problems of the type

$$\min_x ||D(Ax - b)||_2$$

are then solved. Here, the diagonal matrix D may contain large elements making DA ill-conditioned. Despite this ill-conditioning, the method of normal equations is common

and in the literature often recommended as a standard method for solving these problems. The coefficient matrix $A^T D^2 A$ is then explicitly formed and its Cholesky factor computed. It is, however, interesting that the disparaged method of normal equations in practice turns out to work well in this application. One explanation could be that the computed solutions are search directions in Newton's method, wherefore high accuracy not should be that important. This is, however, not a sufficient motivation. We suggest that the observed attractive properties also can be explained as an effect of implicit iterative refinement.

The outline the paper is as follows. Section 2 surveys the most often used direct methods for sparse linear least squares problems. The numerical properties of methods for dense problems obviously carry over to the sparse case. Stable methods for general dense matrices are thus equally accurate when used for solving sparse problems. Dense methods may, however, be less appropriate from a sparsity point of view. Section 3 deals with sparse QR factorization in general. We sketch the different alternatives that have been used during the last 25 years. The most recent advance for efficient sparse QR factorization, multifrontal methods, are considered in Section 4. We describe the algorithm and briefly some details related to the implementation on sequential and parallel computers. In Section 5, we discuss and summarize some often used software packages for solving sparse linear least squares problems. The last section is devoted to applications in optimization where sparse least squares problems arise as repeated subproblems. In particular we focus on the case mentioned above, where the coefficient matrix is a fixed general sparse matrix with a variable diagonal matrix below.

2. Direct methods

Execution times and memory usage are central issues in sparse matrix computation. Many large and sparse problems can thus only be solved if sparsity is well utilized. The introduction of fill-in must in particular be avoided, making it a determining criteria in the evaluation of sparse methods. However, the question of numerical stability must also be taken into account. There are often inherent incompatibilities between sparsity and stability. One such example is the solution of weighted least squares problem by QR factorization. High accuracy may require a column ordering that is less suitable with respect to fill-in. Another example is sparse LU factorization, where a column ordering chosen for sparsity reasons often must be modified to ensure numerical stability. In many implementations of sparse LU factorization pivots are chosen by a threshold criterion, that balances sparsity and numerical stability.

The method of normal equations, based on the orthogonality relation $A^T r = 0$, is the classical and because of its simplicity probably the most common method for solving linear least squares problems. It was derived and used already by Gauss. The solution is computed simply by forming and solving the normal equations,

$$A^T A x = A^T b.$$

The full rank assumptions of A makes $A^T A$ positive definite, wherefore the symmetric linear system can be solved without pivoting using Cholesky factorization, $A^T A = R^T R$.

Important savings in memory usage and execution times are made by appropriately ordering the columns of A. Savings are achieved both in the factorizartion step and in the subsequent solution of triangular systems in R. Often used methods for finding low fill-in column orderings of A (symmetric orderings of A^TA) include the Minimum degree ordering, Nested dissection and Reverse Cuthill-McKee. In many scientific applications it is, however, also possible to formulate the practical problem in such a way that the natural ordering gives low fill-in. With this remark in mind, it is clear that general purpose software in general not can compete with software designed for particular application. The explicit forming of A^TA gives rise to two potential problems: (a) loss of accuracy due to the squared condition number, $\kappa(A^TA) = \kappa^2(A)$, and (b) fill-in. Dense rows in an otherwise sparse matrix A make A^TA filled.

Two of the most reliable and accurate direct numerical methods for solving sparse least squares problems are based on the QR factorization of A,

$$Q^TA = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

The matrix A, with columns permuted for sparsity in R, is decomposed into an orthogonal matrix $Q \in \mathbf{R}^{m\times m}$ and an upper triangular matrix $R \in \mathbf{R}^{n\times n}$. Using the *corrected seminormal equations* (CSNE), a solution $\bar{x} \in \mathbf{R}^n$ is first computed from

$$R^TR\bar{x} = A^Tb,$$

and then corrected by one step of iterative refinement, $x \leftarrow \bar{x} + \delta x$. Here, the correction vector δx is computed in fixed precision from $R^TR\delta x = A^Tr$. By the correction step it can be shown (Björck [2]) that computed solutions normally are of high accuracy.

It is easily shown that the Cholesky factor of A^TA and the upper triangular factor in the QR factorization of A mathematically are the same. They have the same sparsity patterns and, except from possible sign differences of the rows, the same numerical values. It follows that the seminormal equations can be seen as an alternative way of solving the normal equations. The factor R is computed by QR factorization of A instead of Cholesky factorization of A^TA. One could believe that due to a "better" matrix R, a more accurate solution then can be expected. This is, however, not a sufficient explanation. It should be noticed that the solution $\bar{x}$, obtained without the refinement step, generally is not more accurate than a solution computed by the normal equations. Only a careful error analysis can explain why the CSNE works better than the normal equations. One important difference is, however, that the rate of convergence in iterative refinement is much better when the seminormal equations are used.

Golub's method [31] is another approach based on QR factorization. It computes the least squares solution by factorizing A and then solving the triangular system

$$Rx = (Q^Tb)_{1:n}.$$

This method and the CSNE have essentially the same attractive numerical properties. An advantage of the CSNE method is that it only uses the factor R. Subsequent right-hand sides can be handled without the extra cost for storing Q. This is important since Q normally is

much more costly to store than than the matrix R. For details related to the storage and use of a large and sparse matrix Q, we refer to Lu and Barlow [40] and Puglisi [53].

A potential drawback of the normal equations and the methods based on QR factorization of A, is that R is assumed to be sparse. This is, however, not the case when using the *augmented system method*. The least squares solution $x \in \mathbf{R}^n$ and the corresponding residual vector $r = b - Ax \in \mathbf{R}^m$ are computed from a symmetric but indefinite linear system of order $m + n$,

$$\begin{pmatrix} \alpha I_m & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} \alpha^{-1} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

The scaling factor $\alpha \in \mathbf{R}$ is introduced in order to reduce the effect of roundoff errors in the computed solution. With a sufficiently large value of α, the m first pivots in Gaussian elimination are chosen from the (1,1)-block. The resulting system then reduces to the system of normal equations which, as indicated above, is unsatisfactory for less well-conditioned problems. Björck [4] shows how α can be chosen to minimize the condition number of the augmented matrix or, alternatively, to minimize an upper bound for the introduced roundoff error. Since the expressions for these optimal values both include the smallest singular value of A, they are expensive to compute. Cheaper approximative values of α have therefore been proposed. Arioli et al. [1] and Gilbert et al. [27] use $\alpha = \max_{i,j} |a_{ij}|/1000$, while Matstoms [46] approximates the smallest singular value of A by one step of inverse iteration on $A^T A$.

A small α improves stability but may also introduce more fill-in. The required number of floating point operations is then also increased. It should be noticed that Björck's optimal choices of α are derived only with respect to numerical stability. Notice, however, that iterative refinement often compensates for a large value of α.

Efficient solution of the augmented system requires that structure and symmetry is utilized and preserved. However, ordinary symmetric Gaussian elimination, with 1×1 pivots chosen from the diagonal, may be unstable. A combination of 1×1 and 2×2 pivots should instead be used. The use of 2×2 pivots may also reduce the fill-in (Duff and Reid [13]). The MA27 software (Duff and Reid [14]) factorizes sparse matrices by a pivot strategy similar to the one proposed by Bunch and Kaufman [6] for dense matrices. The balance between stability and sparsity is controlled by a threshold criterion in the choice of pivots.

To evaluate the described four methods, Matstoms [45] compares accuracy and execution times in the sparse extention of MATLAB. The experiments are carried out on nine of the matrices from the widely used Harwell-Boeing test collection (Duff et al. [12]), together with five matrices formed by the merging of two Harwell-Boeing matrices. Following Arioli et al. [1] a second set of more ill-conditioned matrices is formed by a row scaling of the Harwell-Boeing matrices. Rows from index $n - 1$ to m are multiplied by a factor 16^{-5}. A set of *consistent* sparse linear least squares problems is defined by choosing the exact solutions $x = (1, \ldots, 1)^T$ and then setting the right-hand sides to $b = Ax$.

Matstoms' conclusions from the numerical experiments can be summarized as follows: The current MATLAB implementation of the augmented system method (build-in) works well for well-conditioned problems of moderate sizes. For general sparse problems a better choice of the scaling parameter α or iterative refinement must be used to get accurate

solutions. However, unacceptable execution times and memory requirements is still a problem. A more appropriate factorization (general sparse LU is now used) would probably make the method more attractive.

The two QR based methods solve all the problems in the two sets with high accuracy. Golub's method also succeeds in solving very ill-conditioned problems in a third set. The two methods based on orthogonal factorization are also much faster than the augmented system method, in particular for large problems.

Due to unsatisfactory numerical properties for less well-conditioned problems, the normal equations must be used with care. In particular, the normal equations may cause problems whenever $\kappa(A) \approx \sqrt{u}$, while the corresponding condition for the QR based methods is $\kappa(A) \approx u$. However, superb execution times in MATLAB make the CNE to an attractive candidate for well-conditioned problems, in particular if combined with iterative refinement.

It follows that methods based on QR factorization are most appropriate. For the solution of general least squares problems in Matlab-like software, Matstoms recommends the use of Golub's method. An efficient implementation of sparse QR factorization is then assumed. It is also assumed that the premultiplication of Q to the right-hand side b is made during the factorization process. The extra cost for storing Q can then be avoided.

3. Sparse QR factorization

This section briefly summarizes the methods that during the last 20-30 years have been used for sparse QR factorization. Algorithms based on Householder transformations have traditionally, until the recent introduction of multifrontal methods, suffered costly (intermediate) fill-in, and have therefore been rejected. See, for example, Duff and Reid [13], Gill and Murray [28] or Heath [33]. They all conclude that "*Givens rotations are a much more appropriate tool in this context because of their ability to introduce zeros more selectively and in a more flexible order*" (Heath [33]).

Givens rotations, either based on row or column-wise elimination, have instead been preferred and used. The column-wise strategy uses the same elimination order as used in the Householder method. Thus, the kth major step eliminates the subdiagonal elements in the kth column, and computes the kth row of R. This variant of sparse Givens QR factorization has, for example, been considered by Duff [10] and Duff and Reid [13]. The introduction of intermediate fill-in can be controlled by an appropriate row ordering. Duff [10] considers different strategies for finding a row ordering that minimizes the introduction of intermediate fill-in.

An alternative strategy, *variable pivot rows*, was suggested by Gentleman [21],[22] (see also Duff [10]). Instead of using a fixed pivot row within each major step, any two rows with nonzero entries in the pivot column can be rotated. A proper choice of row combinations may, compared with the fixed pivot strategies, lead to savings in operation count and memory requirements.

Row-oriented Givens schemes have, for example, been considered by Gentleman [19], [20], Gill and Murray [29], and George and Heath [23]. The input matrix A is processed by rows, in such a way that the kth major step eliminates the subdiagonal elements of the kth row. This is made by rotations of rows into the partially computed factor. In contrast to

column-oriented algorithms, the factor R is not computed such that new rows or columns of R in each major step are definitely computed. The "partially computed factor" therefore just refers to an intermediate result.

Row-oriented schemes have, compared with column-oriented schemes, two important advantages. First, since the algorithm at the same time only operates on a single working row of A and on the partially computed factor, out-of-core implementations follows naturally. Only the partially computed factor R need to be held in memory. Rows of A can sequentially be read from a secondary storage and merged into R. Second, the method is well suited for updates. New observations, in terms of new rows of A, can easily be handled when the original R has been computed. In this case, new rows are processed in exactly the same way as the original rows of A. Finally, it should be noticed that *local* heuristics for finding an appropriate row and column ordering often are based on column wise annihilation of nonzeros. Such algorithms can therefore not be used together with row-oriented schemes. *A priori* heuristics must instead be used; but then the out-of-core argument vanishes.

The algorithm proposed by George and Heath [23] was an important advance. It made QR factorization a useful alternative for the solution of sparse linear least squares problems. QR factorization was previously much slower and more memory consuming than the alternative direct methods. The main contribution by George and Heath was the proposed a priori strategy for finding an appropriate column permutation P_c of A, and a scheme for efficiently handling intermediate fill-in. In the symbolic phase they utilized the previously mentioned relation between QR factorization of A and Cholesky factorization of A^TA. A sparse Cholesky factor of $P_c(A^TA)P_c^T = (AP_c)^T(AP_c)$ guarantees a sparse factor R in QR factorization AP_c. Standard symmetric strategies, for example minimum degree and nested dissection orderings, can therefore be used for computing a column ordering of A.

Variable row pivoting may, as already pointed out, decrease the required number of rotations in sparse Givens QR factorization. It reduces the propagation of intermediate fill-in from previously rotated rows, and decreases the number of new nonzero elements to be annihilated. From this idea, Liu [38] generalizes the row-oriented algorithm by George and Heath [23] to handle more than one simultaneously active triangular matrix. In the George and Heath algorithm, the rows of A are sequentially rotated into a triangular matrix that finally, when all rows are processed, defines the factor R. The operation of rotating a sparse row into a upper triangular matrix is called a *row rotation* (George and Ng [25]). In Liu's algorithm [38], rows are instead annihilated by rotations with one of many triangular structures. Such triangular matrices are then pairwise rotated together into new upper triangular structures (*Generalized row merging* ((Liu [38])). The resulting upper triangular matrix equals the factor R.

We finally comment on some important implementation details mentioned by Liu [38]. First, the submatrix rotations can be performed as dense matrix operations. Rows and columns identically zero remain zero during the triangularization, and can therefore be removed in advance. A simple mapping between local and global column indices is used to match the resulting dense matrix with the overall sparse problem. Second, by visiting the nodes in *depth-first order*, the simultaneously active triangular matrices can be stored and retrieved in first-in/last-out manner. Efficient data representation can therefore be obtain by a *stack* data structure. The use of depth-first ordered trees was in this context first

proposed by Duff and Reid [14]. George and Liu [24] generalizes Liu's algorithm by, instead of using Givens rotations, performing the dense triangularizations by Householder transformations. All these modifications of George and Heath's algorithm are used in the multifrontal Householder algorithm discussed in the next section.

4. Multifrontal QR factorization

Columns of sparse matrices are often structurally independent. In terms of QR factorization this means that columns not need to be eliminated in a strict order from the left to the right, which is the case for dense matrices. A more flexible order can instead be used, where independent columns can be eliminated in any order or in parallel.

A compact way of describing existing column dependencies is by an *elimination tree* (see Liu [39]). It is a rooted tree determined from the nonzero structure of R. The parent p of a child node i is defined to be

$$p = \min\{j > i | r_{ij} \neq 0\}.$$

As already mentioned, it is easily shown that the factor R, except from possible sign differences of the rows, equals the Cholesky factor of $A^T A$. The nonzero structure of R can therefore be predicted by symbolic Cholesky factorization of $A^T A$. Elimination trees can also be computed directly from the structure of A. For the important class of matrices with bipartite graphs having the *Strong Hall property*, symbolic Cholesky factorization correctly predicts the nonzero structure of R (Coleman et al. [8]). Only existing column dependencies are then prescribed by the elimination tree. Each tree node corresponds to the elimination of certain columns in the matrix to be factorized. Nodes in the same subtree are structurally dependent and must be visited from the bottom and upwards. Other nodes are independent and may thus be visited in any order, or in parallel. The elimination tree of a general sparse matrix may be a forest. The order rule is, however, the same; child nodes first.

To factorize a given matrix A, the elimination tree is traversed in such a way that all nodes are visited in an appropriate order. For each node i, a *frontal matrix* F_i is formed by the merging of certain rows of A (those with certain leading entry columns) and the *update matrices* of the child nodes. Dense QR factorization of the frontal matrix F_i,

$$Q^T F_i = \begin{pmatrix} R_i \\ 0 \end{pmatrix},$$

then defines a contribution to the matrix factor R and the dense update matrix of the node itself. The recursive way of computing update matrices motivates the tree traverse rule given above. QR factorization has no effect on columns identically zero, wherefore such columns can be removed in advance. The condensation makes dense methods possible to use in the frontal factorization. A number of further improvements have been proposed and studied by Matstoms [43] and Puglisi [53].

Multifrontal algorithms are basicly parallelized along two different lines. First, the elimination tree can be traversed in parallel. Columns associated with nodes in different branches

are independent and can be simultaneously processed. We refer to this approach as *tree parallelism*. In the alternative approach, called *node level parallelism*, the tree is sequentially traversed but the dense factorization problems of each node are solved in parallel. The efficiency of the two approaches is determined both by the structure of the sparse matrix to be factorized, and by the computer used. In particular factors such as the size of the frontal matrices and the structure of the underlying elimination tree are of great importance. On shared memory architecture, where there is no cost for communication between processors, also the machine dependent parameter $n_{1/2}$ is of importance. It is defined to be the smallest matrix dimension (square matrices), required to achieve half the asymptotic performance of a certain matrix operation. We use it as a measure of how fast the performance increases under increasing matrix dimensions. A large value of $n_{1/2}$ means that large matrices are required to achieve high performance on the computer used. The performance on small matrices may then be unsatisfactory. Mostly large frontal matrices and a small value of $n_{1/2}$ indicates that node level parallelism should be used. Small frontal matrices and a large value of $n_{1/2}$ make, one the other hand, tree parallelism more attractive. In the latter case, it is also important that the elimination tree has a suitable structure. Ideally, the tree should be short and bushy.

In the shared memory implementation (see Matstoms [46],[47] and Puglisi [53]) a pool-of-tasks is initially set to all leaf nodes of the elimination tree. During the computation, processes ask the pool manager for new tasks. The contribution block from A and the update matrices of the child nodes are then merged into a frontal matrix. Dense factorization of the frontal matrix gives a contribution to R and an update matrix of the node itself. A parent node is ready to be processed, and consequently moved to the pool, when all its children are processed. Since nodes in this parallel setting not are visited in a strict depth first order, the stack storage of update matrices can no longer be used. A more general form of dynamic memory allocation (a buddy system) is instead used. Semaphores are also required to prevent processes from simultaneously writing to shared memory blocks.

In a message passing implementation the main problem is to obtain good load balancing and, at the same time, minimize the communication overhead. This can only be achieved by dividing the elimination tree in a number of independent subtrees of essentially the same computational complexity. Each processor is then assigned a subtree. Like in the shared memory implementation, the rather sequential upper half of the tree must be treated in a special way to make full use of parallelism.

5. Software for sparse least squares problems

In this section, we survey some often used software packages for general sparse least squares problems. Many problem related issues, such as problem size, sparsity and structure, determine whether direct or iterative methods should be used. Very large problems with structured nonzero patterns and easily computed nonzero entries are, for example, often solved by iterative methods. Direct methods are, on the other hand, preferable in statistical modeling when the covariance matrix is required. Other problem related details, such as possible rank deficiency, occurrence of weighted rows and the number of right-hand sides, also influence on the choice of method and software. Special software packages, designed

for the particular problems of interest, may sometimes give better performance than the general software packages described here.

MATLAB has been extended to include sparse matrix storage and operations. The included operations and algorithms are described in Gilbert et al. [27]. A minimum degree preordering algorithm and a sparse Cholesky decomposition have, for example, been included. By using these and other built-in routines, new sparse algorithms are relatively easily implemented. There is also a built-in sparse least squares solver in MATLAB. This currently uses the augmented system formulation with the scaling parameter chosen to be $\alpha = 10^{-3} \max |a_{ij}|$. The solution is computed using the minimum degree ordering and the built in sparse LU decomposition.

Matstoms [45] has developed a multifrontal sparse QR decomposition to be used with MATLAB. This is implemented as four m-files, which are available from *netlib*. The main routine is called `sqr` and the statements `[R,p,c]=sqr(A,b)` will compute the factor R in a sparse QR decomposition of A, and $c = Q^T b$. For further details we refer to [45].

More recently C. Sun, Advanced Computing Research Facility, Cornell University, has developed another software package for computing a sparse QR decomposition This package is implemented in C and also designed to be used within the MATLAB environment. C. Sun [55] has also developed a parallel multifrontal algorithm for sparse QR factorization on distributed-memory multiprocessors.

Pierce and Lewis [51] at Boeing have implemented a multifrontal sparse rank revealing QR decomposition/least squares solution module. This code has some optimization for vector computers in general, but it also works very well on a wide variety of scientific workstations. It is included in the commercial software package BCSLIB-EXT from Boeing Information and Support Services, Seattle. This library of FORTRAN callable routines is also given to researchers in laboratories and academia for testing, comparing and as a professional courtesy.

The Harwell Subroutine Library (HSL) has a subroutine MA45 to solve the normal equations. If the least squares problem is written in the augmented matrix form, then the multifrontal subroutine MA27 for solving symmetric indefinite linear system can be used. However, the MA27 code does not exploit the special structure of the augmented system. There is also a new routine MA47 which is designed to efficiently solve this kind of systems.

Closely related to the Harwell MA27 code is the QR27 code, that has been developed by Matstoms [44]. It is a Fortran-77 implementation of the Householder multifrontal algorithm for sparse QR factorization that was described in a previous section. To solve sparse least squares problems it uses the corrected seminormal equations (CSNE). The code is available for academic research and can be ordered by e-mail to *qr27@math.liu.se*. A parallel version of QR27 has been developed for shared memory MIMD computers, see Matstoms [47].

SPARSPAK is a collection of routines for solving sparse systems of linear systems developed at University of Waterloo. It is divided into two portions; SPARSPAK-A deals with sparse symmetric positive definite systems and SPARSPAK-B handles sparse linear least squares problems, including linear equality constraints. For solving least squares both A and B parts are needed. SPARSPAK-B has the feature that dense rows of A, which would cause R to fill, can be withheld from the decomposition and the final solution updated to

incorporate them at the end. Only the upper triangular factor is maintained, and the Givens rotations are not saved.

Zlatev and Nielsen [57] have developed a Fortran subroutine called LLSS01 which uses fast Givens rotations to perform the QR decomposition. The orthogonal matrix Q is not stored, and elements in R smaller than a user specified tolerance are dropped. The solution is computed using fixed precision iterative refinement, or alternatively preconditioned conjugate gradient, with the computed matrix R as preconditioner, see [58]. The table below summarizes the considered software packages.

Table 1. Software packages for sparse linear least squares problems.

Package	Purpose	Author	Distribution
MATLAB	LU and Cholesky	Gilbert et al. [27]	The Mathworks Inc.
SQR (MATLAB)	Householder QR	Matstoms [45]	*netlib*
MA27	LDL^T	Duff and Reid [14]	HSL
MA47	LDL^T	Duff et al. [11]	HSL
QR27	Householder QR	Matstoms [46, 47]	*qr27@math.liu.se*
SPARSPAK-A	Cholesky	Chu et al. [7]	Univ. of Waterloo
SPARSPAK-B	Givens QR	George and Ng [26]	Univ. of Waterloo
YSMP	Cholesky	Eisenstat et al.[15]	Yale
LLSS01	Incomplete QR	Zlatev and Nielsen [57]	Tech. Univ. Denm.

6. Applications in optimization

Sparse linear least squares problems are frequent in large scale optimization. Typical sources are constrained linear least squares problems and interior point methods for linear programming. Also the solution of nonlinear least squares problems give rise to a sequence of sparse linear least squares problems. We discuss these applications and show how linear least squares problems arise as subproblems. Of particular interest is the solution of problems of regularization type. In this case the coefficient matrix is a fixed general sparse matrix with a variable diagonal matrix below. Another interesting topic is the repeated solution of certain linear least squares problems arising in interior point methods for linear programming. The coefficient matrices may here be rather ill-conditioned, and it has therefore been somewhat surprising that the normal equation method works well and usually turns out to give sufficiently accurate solutions.

Our object is far from giving a complete survey of the considered optimization methods. We rather aim to define some important problems in optimization and then, with technical details suppressed, show how the solution leads to repeated unconstrained linear least squares problems. For more exhaustive treatments we refer to Gill et al. [30] and Dennis and Schnabel [9] (nonlinear least squares), Lawson and Hanson [35] and Björck [5] (con-

strained least squares), and Gonzaga [32] and Wright [56] (interior point methods for linear programming).

6.1. Nonlinear least squares problems

Let $f : \mathbf{R}^n \to \mathbf{R}^m$ be a vector-valued real function. The unconstrained nonlinear least squares problem is to find a vector $x \in \mathbf{R}^n$ such that the sum of squares of the $m \geq n$ functions $f_i(x)$ is minimized,

$$\min_x F(x) \quad \text{for} \quad F(x) = \frac{1}{2}||f(x)||_2^2. \tag{2}$$

Let us first consider the standard method of Gauss-Newton for the iterative solution of (2). If $J(x) \in \mathbf{R}^{m\times n}$ is the Jacobian of $f(x)$ and $G_i(x) \in \mathbf{R}^{n\times n}$ the Hessian of the ith component $f_i(x)$ of f, then the Jacobian and Hessian of the objective function $F(x)$ equal

$$g(x) = J(x)^T J(x)$$

and

$$H(x) = J(x)^T J(x) + Q(x), \quad Q(x) = \sum_{i=1}^{m} f_i(x) \cdot G_i(x),$$

respectively. By using Newton's method with the approximation $Q(x) = 0$, one obtains the Gauss-Newton method

$$J(x_k)^T J(x_k) p_k = -J_k^T f_k. \tag{3}$$

Here, x_k is the kth approximation to the solution and p_k the increment defining x_{k+1},

$$x_{k+1} = x_k + p_k.$$

The Jacobian $J(x_k)$ is in practice often ill-conditioned and sometimes even rank deficient. Explicit formation of the coefficient matrix in (3) is therefore unsuitable. A better alternative is to consider (3) as the *normal equations* for the linear least squares problem,

$$\min_p ||J(x_k)p + f_k||_2, \tag{4}$$

which instead can be solved by QR factorization. Rank detection is in the dense case easily implemented by Golub's column pivoting, and for sparse matrices a sparse RRQR algorithm can be used. Search directions are in the rank deficient case determined as minimum norm solutions of (4).

An often used alternative to the Gauss-Newton approach, avoiding critical ill-conditioning of the Jacobian $J(x)$, is given by the *Levenberg-Marquardt* strategy (Levenberg [37] and Marquardt [42]). To guard against inaccurate and unpropitious search directions, a restriction $||p_k||_2 \leq \Delta$ is imposed upon (4). For a parameter $\lambda \geq 0$, related to the restriction Δ, new search directions p_k are defined by

$$(J(x_k)^T J(x_k) + \lambda_k I)p = -J(x_k)^T f(x_k). \tag{5}$$

With this interpretation of the λI term, the approach can be considered as a trust region strategy. Otherwise, (5) can be seen as a pure regularization of (4). The key point is, however, that the added diagonal matrix makes $J^T J + \lambda I$ nonsingular and a unique solution can be guaranteed. Many strategies for choosing λ have been proposed. Algorithm and implementation details are discussed by Moré [48]. The system (5) can obviously be interpreted as the normal equation solution of the linear least squares problem

$$\min_p \left|\left| \begin{pmatrix} J(x_k) \\ \lambda^{1/2} I \end{pmatrix} p + \begin{pmatrix} f(x_k) \\ 0 \end{pmatrix} \right|\right|_2. \tag{6}$$

Alternative methods, like Golub's method and the CSNE, can then be used. We remark that each iteration normally requires the solution of (6) for different values of λ. The most recent approximation x_k and the associated Jacobian $J(x_k)$ are then fixed. Moré [48] introduces a scaling matrix D and thus replaces λI with λD.

6.2. *Constrained least squares problems*

As an example of constrained linear least squares problems, we consider the nonnegativity problem

$$\min_x ||Ax - b||_2 \quad \text{subject to} \quad x \geq 0. \tag{7}$$

Here, $A \in \mathbf{R}^{m \times n}$, $m \geq n$, is assumed to be a large and sparse matrix of full rank. Compared with the previously considered unconstrained least squares problem, a non-negativity requirement is imposed on the solution vector x. Underlying physical restrictions in the mathematical model often motivate such restrictions. More general constraints include upper and lower bounds, $l \leq Cx \leq u$, which for $C = I$ reduces to the simple bounds, $l \leq x \leq u$.

By writing the vector norm in (7) as an inner product, the quadratic problem

$$\min_x (A^T b)^T x + \frac{1}{2} x^T A^T A x \quad \text{subject to} \quad x \geq 0,$$

equivalent to (7), is obtained. The full rank assumption of A makes this problem strictly convex and its unique solution defined by the *linear complementarity problem* (LCP)

$$y = A^T A x - A^T b, \quad x \geq 0, \quad y \geq 0, \quad x^T y = 0. \tag{8}$$

In this section, we consider an interior point method based on the above formulation. Other strategies, also based on the LCP, include *active set* methods (Björck [3] and Oreborn [49]) and *principal pivoting algorithms* (Portugal et al. [52]). These two approaches are based on the same idea, namely that a certain subset of the constraints in optimum must be active and satisfied with equality. To find these active variables, and implicitly the values of the non-active variables, the active set method makes element-wise modifications of a

candidate set. Block principal methods instead exchange more than one variable at a time. A given set of active variables can be eliminated and the other variables computed. The computational effort is in both cases restricted to the solution of unconstrained linear least squares problems in the non-active variables. Thus, the coefficient matrices are defined by a subset of columns from the original matrix A in (7).

Let us now focus on the interior point solution and how it leads to unconstrained least squares problems of the considered regularization form. Technical details are skipped and instead we refer to Portugal et al. [52]. The basic idea is to apply Newton's method to the complementarity problem (8). It can directly be formulated as a system of nonlinear equations,

$$\begin{cases} XYe = 0 \\ A^TAXe - Ye - A^Tb = 0 \end{cases}. \tag{9}$$

The component-wise complementarity conditions $x_iy_i = 0, i = 1, \ldots, n$, are expressed in terms of the diagonal matrices $X = \mathrm{diag}(x)$ and $Y = \mathrm{diag}(y)$. In (9), e is the unit vector with ones in all elements. The Jacobian of (9) is easily computed and the search directions in Newton's methods are defined by a square $2n \times 2n$ sparse linear system,

$$\begin{pmatrix} Y_k & X_k \\ A^TA & -I_n \end{pmatrix} \begin{pmatrix} u^k \\ v^k \end{pmatrix} = \begin{pmatrix} -X_kY_ke + \mu_ke \\ -A^TAX_ke + Y_ke + A^Tb \end{pmatrix}. \tag{10}$$

A centralization parameter μ_k (Lustig et al. [41]) is added to the first block of the right-hand side. With given search directions $(u^k, v^k)^T$, new iterates are computed under a damped update of X_k and Y_k. By block elimination of (10) we obtain

$$\begin{pmatrix} Y_k & X_k \\ A^TA + X_k^{-1}Y_k & 0 \end{pmatrix} \begin{pmatrix} u^k \\ v^k \end{pmatrix} = \begin{pmatrix} -X_kY_ke + \mu_ke \\ -A^TAX_ke + A^Tb + \mu_kX_k^{-1}e \end{pmatrix},$$

and the u^k component is thus defined by

$$(A^TA + X_k^{-1}Y_k)u^k = X_k^{-1}\mu_ke - A^TAX_ke + A^Tb.$$

As in the discussion of nonlinear least squares problems, we identify the above equation as the normal equations for the unconstrained least squares problem

$$\min_u \left\| \begin{pmatrix} A \\ X_k^{-1/2}Y_k^{1/2} \end{pmatrix} u - \begin{pmatrix} b - AX_ke \\ X_k^{-1/2}Y_k^{-1/2}\mu_ke \end{pmatrix} \right\|_2. \tag{11}$$

Remember that X_k and Y_k are diagonal matrices. The coefficient matrix in the above problem therefore consists of a *fixed* sparse matrix A and, in each iteration, *different* diagonal blocks. The v^k component of the search direction is, for a given u_k, defined by

$$v^k = -Y_ke + X_k^{-1}\mu_ke - X_k^{-1}Y_ku^k.$$

A more stable way of computing v^k is given by

$$v^k = A^T(Au^k - r^k) + Y_ke, \qquad r^k = b - Ax_ke.$$

6.3. *Linear programming*

As a last example of applications in optimization, we consider the standard linear program

$$\min_x c^T x \quad \text{subject to} \quad Ax = b, \quad x \geq 0, \tag{12}$$

where $Ax = b$ is a consistent underdetermined system of m equations and n unknowns. For the interior point solution of (12), a barrier term is introduced as follows

$$\min_x \underbrace{c^T x - \mu \sum_{i=1}^{n} \ln x_i}_{B(x,\mu)} \quad \text{subject to} \quad Ax = b. \tag{13}$$

The non-negativity constraints are then implicitly handled by the objective function. The original problem (12) is solved by the repeated solution of (13) for decreasing values of μ. Let $x^{(i)} = x(\mu_i)$ be the solution for $\mu = \mu_i$. Then it can be shown that $x^{(i)} \to x^*$ for $\mu_i \to 0_+$, where x^* is an exact solution of (12). If the starting vector $x^{(0)}$ is feasible, i.e. $Ax^{(0)} = b$ and $x^{(0)} > 0$, then also the subsequent vectors will be feasible. To derive an iterative method for solving the subproblem (13), we first briefly consider the solution of more general convex constrained minimization problems,

$$\min_x f(x) \quad \text{subject to} \quad Ax = b. \tag{14}$$

Feasibility and the first-order optimality conditions requires that a solution x^*, for some vector y^*, satisfies

$$\begin{cases} Ax^* = b, \\ \nabla f(x^*) = A^T y^* \end{cases} \tag{15}$$

Written as a single system of nonlinear equations, a solution x^* and the Lagrange multiplier y^* must satisfy

$$\begin{pmatrix} \nabla f(x) - A^T y \\ Ax - b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

In the Newton solution of this nonlinear system of equations, search directions (p_k, q_k),

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \begin{pmatrix} p_k \\ q_k \end{pmatrix},$$

are computed from the *KKT-system*

$$\begin{pmatrix} \nabla^2 f(x_k) & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p_k \\ -q_k \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) + A^T y_k \\ b - Ax_k \end{pmatrix}. \tag{16}$$

Returning to the linear program (12), we now specialize (16) for the gradient and Hessian of $B(x, \mu)$,

$$\nabla B(x, \mu) = c - \mu X^{-1} e$$

and

$$\nabla^2 B(x,\mu) = \mu X^{-2}.$$

Here, $X = \mathrm{diag}(x)$ and $e = (1,\ldots,1)^T$. Given a solution $x_0 = x^{(i)}$, the next iterate is computed by the inner iteration

$$x_{k+1} = x_k + \alpha_k p_k \tag{17}$$

for

$$\begin{pmatrix} \mu X_k^{-2} & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p_k \\ -q_k \end{pmatrix} = \begin{pmatrix} \mu X_k^{-1} e - c + A^T y_k \\ b - Ax_k \end{pmatrix}. \tag{18}$$

Thus, major iterates $x^{(i)}$ are computed as $x^{(i+1)} = \lim_{k\to\infty} x_k$. The damping factor α_k is introduced to guarantee feasibility. Pure Newton steps may violate the non-negativity constraints. The Lagrange multiplier is simultaneously computed by the iteration

$$y_{k+1} = y_k + q_k.$$

If the last iterate x_k is feasible and satisfies $Ax_k = b$, then (18) becomes

$$\begin{pmatrix} \mu X_k^{-2} & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p_k \\ -q_k \end{pmatrix} = \begin{pmatrix} \mu X_k^{-1} e - c + A^T y_k \\ 0 \end{pmatrix}. \tag{19}$$

The second equation prescribes $Ap_k = 0$, wherefore also the subsequent solution x_{k+1} becomes feasible with respect to the linear constraints $Ax = b$. By observing that a general augmented system,

$$\begin{pmatrix} D & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

is equivalent to the weighted least squares problem $\min_x ||D^{-1/2}(Ay - b)||_2$, the square linear system (19) can equivalently be formulated as a weighted unconstrained linear least squares problem:

$$\min_q ||X_k(A^T q - (c - A^T y_k - \mu X_k^{-1} e))||_2. \tag{20}$$

For a given vector q_k, the search directions p_k are explicitly given by

$$p_k = Xe - X^2(c + A^T y_k - A^T q_k)/\mu.$$

6.4. Discussion

Both the solution of nonlinear and constrained linear least squares problems give rise to subproblems of regularization type. The Levenberg-Marquardt algorithm for nonlinear problems leads to the solution of (6), while the interior point solution of linear constrained problems leads to (11). In both cases we have a fixed general sparse matrix A merged by a varying diagonal block. In the dense case, such problems are normally solved by

bidiagonalization. Orthogonal transformations are then applied both from the right and left to transform A into bidiagonal form (Eldén [16]). If the matrix A instead is large and sparse, then Lanczos bidigonalization can be used. See Paige and Saunders [50].

Let the QR factorization of the coefficient matrix in (1) be given by

$$Q(\lambda)^T \begin{pmatrix} A \\ \lambda I_n \end{pmatrix} = \begin{pmatrix} R(\lambda) \\ 0 \end{pmatrix}. \tag{21}$$

The upper triangular factor $R(\lambda)$ then satisfies

$$R(\lambda)^T R(\lambda) = A^T A + \lambda^2 I_n,$$

and since the second term to the right only affects the diagonal, it is clear that the nonzero structure of R only depends on the structure of A. The analysis phase, which solely is based on the predicted nonzero structure of R, need therefore not to be repeated for new values of λ. In particular the minimum fill-in ordering and the elimination tree, both computed in the analysis phase, are constant and do not need to be recomputed.

Moreover, it is also clear that A in (21) can be replaced by the triangular factor $R = R(0)$ of A. Thus, the factor $R(\lambda)$ can equivalently be computed by the QR factorization

$$Q(\lambda)^T \begin{pmatrix} R \\ \lambda I_n \end{pmatrix} = \begin{pmatrix} R(\lambda) \\ 0 \end{pmatrix}. \tag{22}$$

An appropriate row ordering is required in order to minimize the introduction of intermediate fill-in. For the row-oriented Givens algorithm, George and Heath [23] sort the rows in leading entry order. If ℓ_i is the leading entry column of the ith row in the reordered matrix, then $\ell_1 \leq \ldots \leq \ell_m$. The same row ordering strategy should be used in Householder QR factorization, and is therefore used in the presented multifrontal method. Reid [54] and Lawson and Hanson [35] show how the Householder algorithm can be modified to take advantage of a possible resulting band or block upper triangular structure. The multifrontal algorithm can, in fact, be considered as a generalization of Reid's algorithm for band matrices.

Another row ordering is used if the matrix R from the beginning is considered as the working matrix in the George and Heath method. The diagonal block is then annihilated by rotations with rows in that matrix. However, this approach may cause a serious introduction of intermediate fill-in and should therefore not be used. If P_i is the number of ancestor nodes (a node is here considered to be an ancestor of itself) to the ith node in the elimination tree $T(A^T A)$, then it can be shown that the number of Givens rotations using this bad row ordering equals $\sum_{i=1}^{n} P_i$. Figure 1 illustrates by an example the effect of an appropriate row ordering. The upper triangular factor R is in this case chosen to be bidiagonal. In the example, M_1 is the unpermuted matrix that corresponds to the above bad strategy. Since the elimination tree is a chain of 5 nodes, it follows that 15 Givens rotations are required. The two matrices M_2 and M_3 illustrate the importance of an appropriate tie-breaking strategy in the leading entry order. If the elements from R are ordered first as in M_2, 12 rotations are required. For M_3, where the diagonal entries are placed first, only 9 rotations are required.

$$M_1 = \begin{pmatrix} x & x & & & & \\ & x & x & & & \\ & & x & x & & \\ & & & x & x & \\ & & & & & x \\ x & & & & & \\ & x & & & & \\ & & x & & & \\ & & & x & & \\ & & & & & x \end{pmatrix} \quad M_2 = \begin{pmatrix} x & x & & & & \\ x & & & & & \\ & x & x & & & \\ & x & & & & \\ & & & x & x & \\ & & & x & & \\ & & & & & x \\ & & & & & x \end{pmatrix} \quad M_3 = \begin{pmatrix} x & & & & & \\ x & x & & & & \\ & x & & & & \\ & x & x & & & \\ & & x & & & \\ & & x & x & & \\ & & & x & & \\ & & & x & x & \\ & & & & x & \\ & & & & x & \end{pmatrix}$$

Figure 1. Three alternative row orderings for Givens QR factorization.

Let us now consider the multifrontal factorization of the matrix in (22). The reduction in row dimension compared with (21) also reduces the row dimension of the resulting frontal matrices. Multifrontal factorization of (22) should therefore be faster. However, different effects of vectorization and the structure of the frontal matrices also influence on the performance. It turns out that the number of nonzero entries (nnz) in R is important. By numerical experiments we are lead to the conclusion that (22) should be used only if $nnz(R) < nnz(A)$. Some examples are given in Table 2. Execution times are based on the MATLAB implementation SQR on a *Sun SparcStation*.

Table 2. Factorization times in MATLAB.

Matrix	nnz(A)	nnz(R)	$\begin{pmatrix} A \\ \lambda I \end{pmatrix}$	$\begin{pmatrix} R \\ \lambda I \end{pmatrix}$
WELL1033	4732	2582	10.5	8.5
WELL1850	8755	7691	19.0	15.0
ASH958	1916	2926	5.0	4.2
NFAC30	6728	17670	22.2	25.3
NFAC40	12170	35040	87.6	140

The method of normal equations, applied on the above problem, gives the square system

$$(A^T A + \lambda^2 I_n)x = A^T b. \tag{23}$$

The fixed structure of R makes it possible also for the Cholesky factorization to reuse information from the analysis phase. However, since the diagonal is modified under different values of λ, it is necessary to *recompute* the Cholesky factorization. Previously computed factors $R(\lambda)$ give no computational advantages. However, efficient methods for sparse Cholesky factorization makes this approach in general faster by a factor of 2-3 than the above methods based on QR factorization. A drawback to the Cholesky method is, on the other hand, that the diagonal term in (23) may vanish due to the squared λ. In the

regularization of ill-conditioned least squares problems, this effect may require a larger value of λ in order to stabilize a solution.

In the previous section, it was shown how the interior point solution of linear programs leads to the weighted least squares problem (20). The coefficient matrix XA^T is in practice often ill-conditioned and almost rank deficient, wherefore the method of normal equations should be less useful. However, (20) is in practice solved by the normal equations. "*The small number of observed numerical difficulties with the normal-equation approach has therefore been a continuing surprise. A careful error analysis is likely to explain this phenomenon, but it remains slightly mysterious at this time*" (Wright [56]).

For the solution vector x^*, the second equation of (15) can be considered as a consistent overdetermined system in the unknown vector y. In terms of the original linear program (14), one obtains

$$A^T y = c - \mu X^{-1} e.$$

The consistency makes it possible to introduce any non-singular scaling matrix $X_k \in \mathbf{R}^{m \times m}$ without affecting the solution,

$$X_k A^T y = X_k (c - \mu X^{-1} e). \tag{24}$$

This system is considered and solved as the weighted least squares problem,

$$\min_y ||X_k (A^T y - (c - \mu X^{-1} e))||_2.$$

The accuracy in a computed solution $\bar{y}$ can then be improved by *iterative refinement*. Corrections q are then computed from

$$\min_q ||X_k (A^T q - (c - A^T \bar{y} - \mu X^{-1} e))||_2,$$

and used to refine the approximate solutions, $\bar{y} \leftarrow \bar{y} + q$.

The interior point method for linear programming leads to inner iterations (20) that, except for a non-fixed ($X_k \rightarrow X$ for $x \rightarrow \infty$) matrix X_k in the right-hand side, equals the above procedure. The good behaviour of the normal equations in the solution of (20) should therefore be explained by the asymptotic equivalence to iterative refinement of the underlying linear system (24). Foster [17] shows by numerical experiments that full accuracy normally can be achieved if the normal equations are combined with iterative refinement.

Only a careful error analysis can fully explain the good behaviour of the normal equation method for solving the inner subproblem (20). However, the above discussion indicates that the inner iterations should be equivalent to iterative refinement of solutions to a consistent overdetermined system. A more general analysis would also cover a larger set of subproblems arising in interior point methods.

References

1. M. Arioli, I. S. Duff, and P. de Rijk, *On the augmented system approach to sparse least-squares problems*, Numer. Math., 55 (1989), pp. 667–684.

2. Å. BJÖRCK, *Stability analysis of the method of semi-normal equations for least squares problems*, Linear Algebra Appl., 88/89 (1987), pp. 31–48.
3. Å. BJÖRCK, *A direct method for sparse least squares problems with lower and upper bounds*, Numer. Math., 54 (1988), pp. 19–32.
4. Å. BJÖRCK, *Pivoting and stability in the augmented system method*, Technical Report LiTH-MAT-R-1991-30, Department of Mathematics, Linköping University, June 1991.
5. Å. BJÖRCK, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.
6. J. R. BUNCH AND L. KAUFMAN, *Some stable methods for calculating inertia and solving symmetric linear systems*, Mathematics of Computation, 31 (1977), pp. 162–179.
7. E. C. H. CHU, J. A. GEORGE, J. LIU, AND E. NG, *SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-A*, Research Report CS-84-36, Dept. of Computer Science, University of Waterloo, 1984.
8. T. F. COLEMAN, A. EDENBRANDT, AND J. R. GILBERT, *Predicting fill for sparse orthogonal factorization*, J. ACM, 33 (1986), pp. 517–532.
9. J. DENNIS AND R. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, N.J., 1983.
10. I. S. DUFF, *Pivot selection and row orderings in Givens reduction on sparse matrices*, Computing, 13 (1974), pp. 239–248.
11. I. S. DUFF, N. I. M. GOULD, J. K. REID, J. A. SCOTT, AND K. TURNER, *The factorization of sparse symmetric indefinite matrices*, IMA J. Numer. Anal., 11 (1991), pp. 181–204.
12. I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Softw., 15 (1989), pp. 1–14.
13. I. S. DUFF AND J. K. REID, *A comparison of some methods for the solution of sparse overdetermined systems of linear equations*, J. Inst. Maths. Applics., 17 (1976), pp. 267–280.
14. I. S. DUFF AND J. K. REID, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
15. S. C. EISENSTAT, M. H. SCHULTZ, AND A. H. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 225–237.
16. L. ELDÉN, *Algorithms for the regularization of ill-conditioned least squares problems*, BIT, 17 (1977), pp. 134–145.
17. L. V. FOSTER, *Modifications of the normal equations method that are numerically stable*, in Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms, G. H. Golub and P. V. Dooren, eds., NATO ASI Series, Berlin, 1991, Springer-Verlag, pp. 501–512.
18. C. F. GAUSS, *Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections*, Dover, New York (1963), 1809. C. H. Davis, Trans.
19. W. M. GENTLEMAN, *Basic procedures for large, sparse, or weighted linear least squares problems*, Research report CSRR 2068, University of Waterloo, Waterloo, Ontario, Canada, July 1972.
20. W. M. GENTLEMAN, *Least squares computations by Givens transformations without square roots*, J. Inst. Maths. Applics., 12 (1973), pp. 329–336.
21. W. M. GENTLEMAN, *Error analysis of QR decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189–197.
22. W. M. GENTLEMAN, *Row elimination for solving sparse linear systems and least squares problems*, in Proceedings the 6th Dundee Conference on Numerical Analysis, G. A. Watson, ed., Springer Verlag, 1976, pp. 122–133.
23. J. A. GEORGE AND M. T. HEATH, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra Appl., 34 (1980), pp. 69–83.
24. J. A. GEORGE AND J. W.-H. LIU, *Householder reflections versus Givens rotations in sparse orthogonal decomposition*, Linear Algebra Appl., 88/89 (1987), pp. 223–238.
25. J. A. GEORGE AND E. G. NG, *On row and column orderings for sparse least squares problems*, SIAM J. Numer. Anal., 20 (1981), pp. 326–344.
26. J. A. GEORGE AND E. G. NG, *SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-B*, Research Report CS-84-37, Dept. of Computer Science, University of Waterloo, 1984.
27. J. R. GILBERT, C. MOLER, AND R. SCHREIBER, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

28. P. E. GILL AND W. MURRAY, *Nonlinear least squares and nonlinearly constrained optimization*, in In Proceedings Dundee Conference on Numerical Analysis 1975, Lecture Notes in Mathematics No. 506, Springer Verlag, 1976.
29. P. E. GILL AND W. MURRAY, *The orthogonal factorization of a large sparse matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., New York, 1976, Academic Press, pp. 201–212.
30. P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London and New York, 1981.
31. G. H. GOLUB, *Numerical methods for solving least squares problems*, Numer. Math., 7 (1965), pp. 206–216.
32. C. C. GONZAGA, *Path-following methods for linear programming*, SIAM Review, 34 (1992), pp. 167–224.
33. M. T. HEATH, *Numerical methods for large sparse linear least squares problems*, SIAM J. Sci. Statist. Comput.., 5 (1984), pp. 497–513.
34. G. B. KOLATA, *Geodesy: Dealing with an enormous computer task*, Science, 200 (1978), pp. 421–422.
35. C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice Hall, Englewood Cliffs, New Jersey, 1974.
36. A. M. LEGENDRE, *Nouvelle méthodes pour la détermination des orbites des comètes*, Courcier, Paris, 1805.
37. K. LEVENBERG, *A method for the solution of certain non-linear problems in least squares*, Quart. Appl. Math., 2 (1944), pp. 164–168.
38. J. W.-H. LIU, *On general row merging schemes for sparse Givens transformations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1190–1211.
39. J. W.-H. LIU, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.
40. S. LU AND J. L. BARLOW, *Multifrontal computation with the orthogonal factors of sparse matrices*, SIAM J. Matr. Anal. 3(1996), pp. 658-679.
41. I. LUSTIG, R. MARSTEN, AND D. SHANNO, *Computational experience with a primal-dual interior point method for linear programming*, Linear Algebra Appl., (1991), pp. 191–222.
42. D. W. MARQUARDT, *An algorithm for least-squares estimation of nonlinear parameters*, Journal of the Society for Industrial and Applied Mathematics, 11 (1963), pp. 431–441.
43. P. MATSTOMS, *The multifrontal solution of sparse linear least squares problems*, Licentiat thesis, Linköping University, 1991.
44. P. MATSTOMS, *QR27—Specification sheet*, Tech. Report March 1992, Department of Mathematics, 1992.
45. P. MATSTOMS, *Sparse QR factorization in MATLAB*, ACM Trans. Math. Software, 20 (1994), pp. 136–159.
46. P. MATSTOMS, *Sparse QR Factorization with Applications to Linear Least Squares Problems*, PhD thesis, Linköping University, 1994.
47. P. MATSTOMS, *Parallel sparse QR factorization on shared memory architectures*, Parallel Computing, 21 (1995), pp. 473–486.
48. J. J. MORÉ, *The Levenberg-Marquardt algorithm: Implementation and theory*, in G.A. Watson, Lecture Notes in Math. 630, Berlin, 1978, Springer Verlag, pp. 105–116.
49. U. OREBORN, *A Direct Method for Sparse Nonnegative Least Squares Problems*, licentiat thesis, Linköping University, 1986.
50. C. C. PAIGE AND M. A. SAUNDERS, *LSQR. an algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.
51. D. J. PIERCE AND J. G. LEWIS, *Sparse multifrontal rank revealing QR factorization*, Technical Report MEA-TR-193-Revised, Boeing Information and Support Services, 1995.
52. L. F. PORTUGAL, J. J. JÚDICE, AND L. N. VICENTE, *Solution of large scale linear least-squares problems with nonnegativ variables*, technical report, Departamento de ciêcias da terra, Universidade de Coimbra, 3000 Coimbra, Portugal, 1993.
53. C. PUGLISI, *QR factorization of large sparse overdetermined and square matrices with the multifrontal method in a multiprocessor environment*, PhD thesis, CERFACS, 42 av. G. Coriolis, 31057 Toulouse Cedex, France.
54. J. K. REID, *A note on the least squares solution of a band system of linear equations by Householder reductions*, Comput J., 10 (1967), pp. 188–189.
55. C. SUN, *Parallel sparse orthogonal factorization on distributed-memory multiprocessors*, SIAM J. Sci. Comput., 17 (1996), p. to appear.

56. M. H. WRIGHT, *Interior methods for constrained optimization*, Acta Numerica 1992, Cambridge University Press, 1992, pp. 341–407.
57. Z. ZLATEV AND H. NIELSEN, *LLSS01 - a Fortran subroutine for solving least squares problems (User's guide)*, Technical Report 79-07, Institute of Numerical Analysis, Technical University of Denmark, Lyngby, Denmark, 1979.
58. Z. ZLATEV AND H. NIELSEN, *Solving large and sparse linear least-squares problems by conjugate gradient algorithms*, Comput. Math. Applic., 15 (1988), pp. 185–202.

Computational Optimization and Applications, 7, 111–126 (1997)

Simulated Annealing and Genetic Algorithms for the Facility Layout Problem: A Survey

THELMA D. MAVRIDOU. AND PANOS M. PARDALOS pardalos@ufl.edu
Center for Applied Optimization, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611-6595 USA

Abstract. The facility layout problem (FLP) has many practical applications and is known to be $\mathcal{NP}$-hard. During recent decades exact and heuristic approaches have been proposed in the literature to solve FLPs. In this paper we review the most recent developments regarding simulated annealing and genetic algorithms for solving facility layout problems approximately.

Keywords: Heuristics, Simulated Annealing, Genetic Algorithms, Facility Layout Problem, Parallel Algorithms, Combinatorial Optimization.

1. Introduction

The facility layout problem deals with the physical arrangement of a given number of departments or machines within a given configuration. In the context of manufacturing the objective is to minimize the total material handling cost of moving the required material between the departments. The importance of material handling is stated by Tompkins and White [67] who claimed that 20-50 % of the total operating expenses within manufacturing are attributed to it.

The facility layout problem is one of the best-studied problems in the field of combinatorial optimization. A number of formulations have been developed for the problem. More particularly the FLP has been modeled as [36] :

1. quadratic assignment problem (QAP),
2. quadratic set covering problem,
3. linear integer programming problem,
4. mixed integer programming problem,
5. graph theoretic problem.

The quadratic assignment formulation has been traditionally used to model the facility layout problem. QAP was first introduced by Koopmans and Beckmann [33] in 1957 as a mathematical model for locating a set of indivisible economic activities. Consider the problem of allocating a set of facilities to a set of locations, with the objective to minimize the cost associated not only with the distance between locations but with the flow also. More specifically, given two $n \times n$ matrices $F = (f_{ij})$ and $D = (d_{kl})$ where f_{ij} is the flow

between the facility i and the facility j, and d_{kl} is the distance between the location k and the location l, and a set of integers $N = \{1, 2, \ldots, n\}$, the QAP can be written as follows:

$$\min_{p \in \Pi_N} \sum_{i}^{n} \sum_{j}^{n} f_{ij} d_{p(i)p(j)} \tag{1}$$

where Π_N is the set of all permutations of N, and n is the number of facilities and locations [56].

Exact algorithms for solving the FLP include *branch and bound* ([39], [30]) and *cutting plane algorithms* ([7], [9]). These approaches require rather high computational time as the problem size increases, resulting in practice in the solution of only moderately sized problem instances. Therefore, a number of heuristic algorithms, such as *construction, improvement* and *hybrid algorithms*, have been developed for sub-optimally solving large-size instances in a reasonable amount of CPU time and computer memory. Recent survey papers on the facility layout problem and its solution approaches can be found in [22], [36], and in [43].

In this paper we focus on the work that has been done to date for solving the facility layout problem using simulated annealing (SA) and genetic algorithms (GA). Both heuristic approaches are stochastic search techniques modeled on processes found in nature (thermodynamic process and natural evolution). These heuristic methods have been used to solve a wide variety of combinatorial optimization problems ([12], [19], [37], [47]).

2. Simulated Annealing for the Facility Layout Problem

Simulated annealing was first proposed by Kirkpatrick et al. [31] as a method for solving combinatorial optimization problems. The name of the algorithm derives from an analogy between the simulation of the annealing of solids first proposed by Metropolis et al. [44], and the strategy of solving combinatorial optimization problems. Annealing refers to a process of cooling material slowly until it reaches a stable state. Starting from an initial state, the system is perturbed at random to a new state in the neighborhood of the original one, for which a change of ΔE in the objective function value (OFV) takes place. In a minimization process if the change ΔE is negative then the transformation to the new state is accepted. If $\Delta E \geq 0$ the transformation is accepted with a certain probability of $p(\Delta E) = e^{\frac{-\Delta E}{k_b T}}$, where T is a control parameter corresponding to the temperature in the analogy and k_b is Boltzmann' s constant. The change ΔE in the OFV corresponds to the change in the energy level (in the analogy) that occurs as the temperature T decreases. SA gives us a mechanism for accepting small increases in the objective function value, controlling though the probability of acceptance $p(\Delta E)$ through the temperatures. Kirkpatrick et al. [31] argue that allowing "hill climbing" moves, one can avoid configurations that lead to locally optimal solutions and eventually higher quality solutions can be obtained. So the main advantage of the simulated annealing method is its ability to escape from local optima.

The main features of the SA method are [14] :

- the *temperature* T, which is the parameter that controls the probability $p(\Delta E)$ of accepting a cost-increasing interchange. During the course of the algorithm T is decreased

in order to steadily reduce the probability of acceptance of interchanges that increase the value of the objective function,

- the *equilibrium*, i.e. the condition in which a further improvement in the solution using additional interchanges is highly unlikely to occur,
- the *annealing schedule* that determines when and by how much the temperature is to be reduced.

A pseudo-code of the simulated annealing procedure is given in Figure 1 [54].

Simulated Annealing Procedure

Input: *A problem instance*

Output: *A (sub-optimal) solution*

1. Generate an initial solution at random and initialize the temperature T.
2. **while** $(T > 0)$ **do**
   (a) **while** (thermal equilibrium not reached) **do**
      (i) Generate a neighbor state at random and evaluate the change in energy level ΔE.
      (ii) If $\Delta E < 0$ update current state with new state.
      (iii) If $\Delta E \geq 0$ update current state with new state with probability $e^{\frac{-\Delta E}{k_b T}}$, where k_b is a constant.
   (b) Decrease temperature T according to annealing schedule.
3. Output the solution having the lowest energy.

Figure 1. Simulated Annealing Procedure

Several implementations of the simulated annealing algorithm have been proposed for the facility layout problem. We will present the main concepts of the most recent approaches and comment on the computational results.

Heragu and Alfa in [21], present an extensive experimental analysis of two simulated annealing based algorithms, implementing them on two patterns of layout, the *single-row*

and *multi-row facility layouts*. The first algorithm uses the standard techniques of the SA heuristic. In the main step the algorithm examines the random exchange of the positions of two facilities. The new solution is accepted if the exchange results in a lower OFV. Otherwise, the difference ΔE between the OFV of the best solution obtained so far and the current solution is computed. This solution is accepted with probability $e^{\frac{-\Delta E}{T}}$. This step is repeated $100n$ times or until the number of new solutions accepted is equal to $10n$, where n is the number of facilities in the layout problem. Next, the algorithm decreases the value of temperature T by multiplying it by the *cooling ratio* r and repeats the main step. The stopping criterion is a fixed maximum number of temperature change steps. The initial temperature T is set as a number sufficiently larger than the largest ΔE encountered for problems tested with other heuristics. Guidelines for setting the parameters can be found in [57].

The second algorithm presented in the same paper is a *hybrid SA algorithm* (HSA), which uses a "core" algorithm to generate a "good" initial solution, and then improves it using the SA algorithm described before. The core algorithm is a modified penalty algorithm (MP) presented in [22]. Eight test problems of size up to 30 (available in the literature) are used for the single-row case. Each test problem is solved 10 times using the same initial solution. For six of the problems the HSA algorithm produces optimal or best-known solutions. For the remaining two problems, the solutions are better than those previously reported in the literature. A comparison between the HSA and the SA algorithms is presented, as well as with three other heuristic algorithms (a 2-way exchange, a 3-way exchange and a Wilhem-Ward version of simulated annealing [70]) using 15 equal-area multi-row FLPs. The HSA in terms of solution quality, performed better than all the other algorithms though requiring more computational time than the SA algorithm. Also as the number of annealing runs increases, SA seems to produce similar quality solutions with HSA with less computational effort.

Another implementation of the SA algorithm applied to the *cellular layout problem* can be found in [27]. This problem involves the determination of the relative positions of n equidimensional manufacturing entities which may represent either the set of machines belonging to a cell (*intra-cell problem*) or the manufacturing cells within a shop (*inter-cell problem*). The objective of both layout problems is to minimize the total material flow (cost) between the manufacturing entities. The method presented in the paper is called CLASS, which stands for Computerized LAyout Solutions using Simulated annealing. The proposed algorithm is a regular simulated annealing algorithm with the following most important elements:

- *Solution space*: The solution space consists of a $n \times n$ grid, i.e. n^2 positions are available to be occupied by the n entities. The distance between all pairs of positions is determined using *geometric* or *Manhattan* distances.

- *Interchanges*: The interchange given a solution can be either a move of an entity from its current position to an unoccupied position or an "exchange" of the positions of two entities. The two positions from the solution space that are exchanged are selected *randomly*.

- *Annealing schedule*: The annealing schedule considers the initial temperature to be sufficiently large so that all interchanges are eventually accepted. The temperature is reduced by multiplying it with a constant that takes values between 0 and 1.

- *Parameters*: The number of interchanges to be attempted at each temperature, the number of accepted interchanges at each step and the total number of temperature change steps are $100n$, $10n$ and 100 respectively, following the guidelines of [57], exactly as in [21].

- *Interchange Acceptance Criterion*: The interchange is accepted if a randomly generated number between 0 and 1 is less than the value of $e^{\frac{-\Delta E}{T}}$, where ΔE and T are respectively the difference in the OFV and the temperature at the current step.

CLASS was compared to twelve other layout methods (discussed in [36]) in terms of both the quality of the solution and the speed of convergence. Eight problems available in the literature were used for the comparison of the algorithms, with sizes between $n = 5$ and $n = 30$. In each case CLASS either equals the performance of, or outperforms each of the other methods. The sensitivity of CLASS to the initial conditions was tested by running each of the test problems of sizes 5,6,7, and 8, five times, each time with a different initial solution. The optimal solution was obtained in each case, indicating the insensitivity of the solution quality to the initial conditions.

For the *inter-cell* problem Tam in [64] describes a SA solution approach which takes into consideration the traffic between cells, the geometric constraints of the individual cells and any occupied regions on the floor plan. The objective is to find a layout that minimizes the weighted flow of parts between the manufacturing cells while satisfying the area and shape constraints of the individual cells. There are several critical points concerning the problem formulation:

- *Layout representation*: The layout takes the form of a slicing structure, which is represented by a *slicing tree*. This is a binary tree representing the recursive partitioning process of a rectangular area, through *cuts*. A cut specifies the relative position of the departments (left, right, below or above each other) through four distinguished *branching operators*.

- *Solution space*: The solution space is defined as the set S which consists of all slicing trees that can be generated by rearranging cuts of a given structure. It is shown that $|S| = 4^{n-1}$, where n is the number of cells and the size of the neighborhood N is $|N| = 4n - 5$.

- *Area constraints*: The location where a rectangular partition is cut, i.e. the cut point, must be chosen so that the split partitions receive their required areas. The cut point is determined in a top-down fashion starting from the "root" of the tree.

- *Shape constraints*: The cell' s shape is described using the *aspect ratio* and the *deadspace ratio*. The first ratio is the height over the width of the partition allocated to a cell. The second ratio is used to measure the amount of unusable space within the partition allocated to a manufacturing cell. Both ratios have lower and upper bounds.

- *Slicing tree construction*: Using numerical *clustering techniques* a slicing tree is constructed in such a way that cells with large inter-cell traffic volume are placed in close proximity with each other.

The attractive element of the algorithm is that it exploits the hierarchical representation of the layout, so that the probability of selecting a neighborhood state is not uniformly distributed (as in a regular SA algorithm), but is dependent on T. More particularly, when T is high at the first steps of the annealing procedure, a cut near the root of the slicing tree will be selected, causing large swings in the cost function value since a large number of cells will have to be relocated. As T decreases during the course of the algorithm, cuts that are located at a lower level in the tree are selected, to generate a neighborhood state. So a guided search in the set of neighboring solutions is adopted. The algorithm was compared to two other local search methods, denoted as HC (a straightforward hillclimbing method) and BC (a modified version of HC). Two test problems of size $n = 20$ and $n = 30$ were constructed for the comparison. Each method was run 10 times with different initial solutions. The computation time was kept the same among the three methods. In terms of solution quality the proposed SA algorithm outperformed the other two methods, both in average and minimum cost.

In [34] Kouvelis and Chiang address the *single row layout problem* (SRLP) in flexible manufacturing systems (FMSs). The problem deals with the optimal arrangement of n machines along a straight track with a material handling device moving jobs from one machine to another. The difficulty of the problem is due to the variety of parts to be processed in different ranges of *operation sequences*. When the sequence of operations of a job is not the same as the sequence of the locations of the machines, the job sometimes has to travel in reverse (backtrack) in order to receive the required operations. The objective of the SRLP is to find the ordering of the machines that minimizes the total backtracking distance of the material handling device. If we consider n machines and n candidate locations for the machines to be placed, the solution to the SRLP is one of the possible permutations of the set $S = \{1, 2, \ldots, n\}$ defined as the set of the *workstation assignment vectors*, each one representing a configuration of the machines in a single row. The *neighborhood* of a configuration is the set N of configurations resulting by the interchange of the locations of two machines. The initial configuration is obtained by randomly assigning machines to locations. For the setting of the parameters of the SA algorithm, i.e. the initial acceptance probability (through which the initial temperature will be calculated), the number of interchanges attempted before the reduction of the temperature, the value of the cooling ratio, and the number of steps to reach the equilibrium, a *sensitivity analysis* was performed with respect to each individual parameter. For each parameter a range of values is tested while all other parameters are held fixed. The best values of the parameters are kept as the final ones to be used in the algorithm. The experimental analysis showed that fine-tuning of the SA parameters with respect to each specific application and the selection of the initial solution is very important for the performance of the algorithm in terms of quality solution.

The same authors and J. Fitzsimmons in [35] describe two distinct implementations of the simulated annealing algorithm for machine layout problems in the presence of *zoning constraints*. These constraints are restrictions on the arrangement of machines. *Positive zoning constraints* require that certain machines have to be placed near each other, while

negative zoning constraints do not allow certain machines to be in close proximity. The problem is formulated as a *restricted quadratic assignment problem.* Assuming that the number of candidate locations is equal to the number of machines, the objective is to assign the machines to the locations in a way that the cost function is minimized with respect to the zoning constraints. The first of the SA algorithms called the *Compulsion Method*, takes into consideration the zoning constraints mostly during the search for a new layout in the neighborhood of the original one. The second algorithm, the *Penalty Method*, takes into account the presence of the zoning constraints in the objective function through the use of appropriate penalty terms. For each layout that violates any of the zoning constraints, corresponding penalty terms are charged in the OFV. The fine-tuning of the parameters for both SA procedures and the interpretation of the configuration, the neighborhood of a configuration and the initial configuration are the same as described for [34]. The two versions are compared on an extensive set of computational experiments using test problems of size ranges from 5 to 30 machines. The results showed that the Compulsion Method outperforms the Penalty Method in terms of CPU time and solution quality. The basic advantage of the Penalty Method is that it can be easily changed to handle the addition of extra zoning constraints.

Meller and Bozer in [42] describe a Simulated Annealing Based Layout Evaluation algorithm (SABLE), which introduces a new generator routine for candidate layout solutions, combined with the use of *spacefilling curves.* The algorithm is implemented on a set of single and multiple floor facility layout problems. For the single-floor case test problems of sizes 11 to 25 are used, and the performance of SABLE is compared to the performance of the algorithms presented in [2], [49], [70], and [8]. An average and a worst-case analysis shows that the proposed algorithm performs the best in terms of solution quality. Additionally, SABLE performed better than Tam's SA algorithm [64] on a data set of 20 and 30-size department single-floor FLPs. Let us note that regarding the department shapes, Tam's algorithm generally assumes rectangular shapes, while the proposed algorithm tends to generate departments with non-rectangular shapes. For the multi-floor case, test problems with up to 4 floors and 40 departments were used to evaluate the performance of SABLE. The results indicate the robustness of the algorithm to changes in the vertical to horizontal ratio.

Other recent Simulated Annealing algorithms for layout problems can be found in [62] and [61].

For the special case of QAP several SA approaches have been proposed. Burkard and Rendl [10] were the first to apply simulated annealing for solving the QAP. They reported on rather favorable computational results indicating that the obtained solutions deviate only $1 - 2$ % from the best known solutions. Wilhelm and Ward [70] also applied the SA algorithm to quadratic assignment problems, by further experimenting on the procedure. They report on the sensitivity of SA to the control parameters, and evaluate the algorithm using problems ranging in size from $n = 5$ to $n = 100$. In particular computational results were provided for the test problems in Nugent et al. [49] and for two test problems they introduced in the paper. In [11], Connolly discusses the implementation of SA on 7 problems. The computational results indicate that examining sequentially generated neighboring solutions, rather than randomly generated ones, makes the SA algorithm more

efficient. In [59] and [60] simulated annealing is used as a tool for interactive facility layout decisions. More recently Laursen [38] investigated the performance of the SA algorithm by varying two parameters: (1) the number of simulations, and (2) the simulation length, while in both cases the algorithm uses the same computational time for a specific instance problem. Laursen concluded that the length of each simulation is optimizable and that a large range of its values generate a near-optimal solution quality.

3. Genetic Algorithms for the Facility Layout Problem

Genetic Algorithms (GA) were first introduced by John Holland et al. [23] at the University of Michigan in 1975. Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. GA try to imitate the development of new and better populations among different species during evolution. Unlike most of the heuristic search algorithms, GA conduct the search through the information of a population consisting of a subset of individuals, i.e. solutions. Each solution is associated with a fitness value, which is the objective function value of the solution. Solutions to optimization problems can often be coded to strings of finite length. The genetic algorithms work on these strings. The encoding is done through the structure named *chromosomes*, where each chromosome is made up of units called *genes*.

There are some determining factors that strongly affect the efficiency of genetic algorithms :

1. The representation of the solutions by strings.
2. The generation of the initial population.
3. The selection of individuals in an old population (*parents*) that will be allowed to affect the individuals of a new population.
4. The *genetic operators* that are used to recombine the genetic heritage from the parents to produce children. The most often-used operators are the *crossover* and the *mutation*.

The selection of individuals that will be allowed to affect the following generation is based on the fitness of the individuals. This is done in such a way that individuals with better fitness are more likely to be chosen to become parents. The *recombination* of the population consists of the following four operations:

- *Crossover.* By combining the coded solution strings of two parents two children are created. If one considers the biological origin of the genetic algorithms it makes sense to denote the coded solution string "genome" and look at this procedure as a result of mating. To avoid chaotic behavior, not all individuals in the new population are generated by this operator. The probability of applying this operator (crossover rate) is denoted by p_c.
- *Mutation.* In order to give the populations new impulses some random changes in the genomes are allowed to occur. The mutation operator changes a "gene" in a solution with a probability (mutation rate) p_m.

- *Local search.* It has proven very efficient to search for locally optimal solutions in the neighborhood of the children [40]. If one is able to find a better solution then it will replace the original child as a member of the new population.

- *Control of new individuals.* It is not unlikely that a child will have worse fitness than its parents. In that case the child might not be accepted in the new generation.

Let us note also that a GA implementation requires the specification of certain parameters such as *population size*, and *number of generations*.

Let P_t denote the population at time t. Then the genetic algorithm procedure can be described as in Figure (2) [54].

Genetic Algorithm Procedure

Input: *A problem instance*

Output: *A (sub-optimal) solution*

1. $t = 0$, initialize P_t, and evaluate the fitness of the individuals in P_t

2. **while** (termination condition is not satisfied) **do**
   (a) $t = t + 1$
   (b) Select P_t, recombine P_t and evaluate P_t

3. Output the best solution in the population as the (sub-optimal) solution.

Figure 2. Genetic Algorithm Procedure

We continue with the description of various implementations of the genetic algorithm for the facility layout problem.

As we have seen in the section of SA for the facility layout problem, Tam [64] uses a simulated annealing approach to solve the *inter-cell problem*. The same author using the same problem formulation and representation of the floorplan layout as a slicing tree, attempts a solution approach to the problem using Genetic Algorithms [63]. In applying a GA an important part of the implementation is the coding of solutions as strings of finite length. For the problem formulation under consideration, a slicing tree can be generated by a string using as its elements the nodes of the tree in a sequence which starts from the bottom level nodes and ends at the root of the tree. The nodes of the tree represent either facility identifications (*operands*) or "cut" symbols (*operators*). The proposed GA uses for the recombination of the population the crossover and mutation operators, as described for

the general genetic algorithm. For the selection of the new population the *reproduction operator* is used. Under this operator the chance of being selected to remain in the new population P_{t+1} is proportional to the fitness value of the individual. This operator assigns to each individual a sampling rate $T(s,t) = \mu(s)/\bar{\mu}(s)$, where function μ measures the fitness level of individual s and $\bar{\mu}(s)$ is the average fitness of P_t. So the individuals with above the average fitness will have a higher survival probability than those below the average fitness. The selection of the parameters population size, crossover rate p_c and mutation rate p_m, is based on previous studies that can be found in the literature ([18], [58]). Four layouts with $12, 15, 20$, and 30 facilities were retrieved from Nugent et al. [49]. Initial solutions were obtained by randomly generating cut operators for 30 slicing trees. For each case the GA was run for 150 generations with 10 different sets of initial solutions. The best and average solution in each generation were gathered. The performance of GA was compared with that of a hillclimbing method (HC), which searches through a neighborhood N, where N is the set of operator sequences generated from changing one operator. GA outperformed HC both in terms of minimum and average costs. For the 30-facility layout GA improved the minimum cost by 10.5% and the average cost by 13%.

Koakutsu and Hirata [32] propose an interesting combined approach called *genetic simulated annealing* (GSA) for the solution of the floorplan design of VLSI (Very Large Scale Integrated) circuits. The problem involves the arrangement of a given set of rectangular modules (with no fixed shapes or dimensions) in the plane, with the objective to minimize: (1) the area of the enclosing rectangle which should contain all the modules, and (2) the total wire length between modules that should be connected in the circuit. The main features of the algorithm are the following:

- *Stochastic Optimization*: GSA uses the stochastic optimization used in simulated annealing so that a neighbor state for which there is an increase of the cost function is accepted with a certain probability.

- *Multiple Search Paths*: A population of solutions corresponding to the population of GAs is used to initialize the search in multiple directions. The stochastic optimization is applied to each solution of the population.

- *Selection of search paths*: The selection operator replaces solutions which have value higher than the average value of the population, with solutions that have lower cost value than the average value of the population. This way, paths which are expected to reach good solutions are selected.

- *Genetic Operators*: A genetic crossover operator is used to generate new solutions.

The formulation of the problem represents the floorplan layout as a slicing tree. The representation of a solution as a string is similar to the one described previously in [63], using in this case, vertical and horizontal cuts with corresponding branching operators. GSA is tested on three floorplan problem instances. The first has 16 modules, each one a fixed square of unit area, having wires connecting to its horizontal and vertical neighbors. The second problem has 16 modules and 25 wires, and the third one has 20 modules and 31 wires. For the last two problems the total module area is 100. The proposed algorithm

was compared to a regular SA algorithm. Both algorithms run 100 times with different initial solutions for each of the above problem instances. The average costs are used for the comparison. The results show that GA improves the average cost by $1.7\% - 9.8\%$ compared to the SA within the same computational time.

More recently Banerjee and Zhou [3] developed a genetic algorithm to solve a variation of Montereuil' s mixed integer programming formulation for the FLP [46], and in particular for the special case of *single loop material flow path configuration.* They introduce a "*knowledge-augmented mutation operator*" to determine the flow path direction, which appears to perform well for the cases where the layout has very low flow path dominance. Previous applications of GA for facilities layout design can be found in [4] from the same authors and Montreuil.

Tate and Smith [65] applied GA using an adaptive penalty function to the *unequal-area facility layout problem* with shape constraints. The shape restrictions are expressed through a flexible bay structure proposed in the literature [68]. The rectangular area in which the facilities are to be located is divided into vertical bays of different width and each bay is divided into rectangular departments of different length. The encoding of the solutions to strings is done with two distinct chromosomes. The first one is the *sequential chromosome* which is represented by a permutation of the set $N = \{1, 2, \ldots, n\}$, where n is the number of departments. The sequence of the permutation starts by reading departments bay to bay, from top to bottom and from left to right at the rectangular area. The second chromosome is the *bay chromosome* where each gene shows for each bay the number of departments contained in the previous bays including the involved one, showing this way the breaks that occur in the sequence between bays. For example, consider 4 bays having 3, 4, 6 and 2 departments respectively starting from the left bay. Then using the bay chromosome the solution encoding is $(3, 7, 13)$. Note that the last breakpoint at 15 is obvious. The proposed GA uses variants of crossover and mutation operators.

- The *variant of the crossover operator* works as follows: using two individuals to be the parents, one offspring (child) is generated by the following rules. For the case of GA encoding using the sequential chromosome, each location in the child' s sequence is the department number in the corresponding location from one of the parents, both having the same probability to be selected. This will force the common locations in the sequences of the parents to be carried over to the child. Also each department must occur only once in the child. For the bay chromosomes, the location and number of bay breaks in the child' s sequence is taken from one of the parents, both having equal probabilities to be selected.

- The mutation uses three different operators. Two of the operators alter the number of bays affecting only the bay chromosome and one operator reverses a subsequence of the departments affecting the sequence chromosome.

The evolution parameters, i.e. the population size, and the crossover and mutation rates are determined after several trial runs. An *adaptive penalty function* is used to find good feasible solutions. The penalty function is adaptive because during the course of the algorithm it uses observed population data to adjust the level of the penalty that is applied to the infeasible solutions. Test problems with size ranges from 10 to 20 departments, already published in

the literature ([6], [69], [2]) were used to evaluate the efficiency of the proposed genetic algorithm. The proposed approach proved to be the best in terms of quality solution when compared with previous published results for the problems under consideration.

Genetic algorithms are inherently parallel in nature. Several implementations of GA in parallel environments have recently appeared, introducing in this way a new group of GA, the *Parallel Genetic Algorithms* (PGA). The population of a parallel genetic algorithm is divided into *subpopulations*. Then an independent GA is locally performed on each of these subpopulations, and the best solutions in each case are transferred to all the other subpopulations. Two types of communication are established among the subpopulations [48]. Either *among all nodes* where the best solution of each subpopulation is broadcasted to all the other subpopulations, or *among the neighboring nodes*, where only the neighboring subpopulations receive the best solutions.

The most important features of PGA, which result in a considerable speedup relative to sequential GAs, are the following [28]:

- *Local selection* : In sequential GAs the selection operation takes place by considering the whole population. In a PGA this operation is performed locally by the selection of an individual in a neighborhood.
- *Asynchronous behavior* : It allows the evolution of different population structures at different speeds, resulting in an overall improvement of the algorithm in terms of computational time.
- *Reliability* in computation performance : The computation performance of one processor does not affect the performance of the other processors.

Several implementations of PGA have been proposed for the solution of the *quadratic assignment problem*. An application of an asynchronous parallel GA called ASPARAGOS has been presented by Muhlenbein [47] for the QAP, introducing a *polysexual voting recombination operator*. The PGA was tested on QAPs of size 30 and 36 with known solutions. The algorithm found a new optimum for the Steinberg's problem (QAP of size 36). The numbers of processors that were used to run this problem were 16, 32 and 64. The 64 processors implementation (on a system with distributed memory) gave by far the best results in terms of computational time. Furthermore, Huntley and Brown [26] developed a parallel hybrid of SA and GA to solve the QAP approximately. A parallel genetic algorithm is used to produce a good initial solution for each population and the SA algorithm is used for improving these solutions. More recently, Battiti and Tecchiolli in [5] developed parallelization schemes of genetic algorithms for quadratic assignment problems presenting indicative experimental results.

4. Concluding Remarks

In this paper we summarized the work that has been done in recent years in implementing simulated annealing and genetic algorithms for solving the facility layout problem. Both heuristic approaches have been successfully used to approximately solve difficult combinatorial optimization problems. For the FLP also, the procedures seem to find sub-optimal

solutions in a reasonable amount of computational time. Considering the latest interest and experience in efficient implementation of search algorithms on *parallel multiprocessing computers* ([41], [50], [51], [52], [54], [53], [55]) that significantly increase the sizes of the problems that can be solved, the use of these heuristics becomes more attractive. It is expected that efficient parallel implementations of simulated annealing and genetic algorithms will be very useful for practitioners dealing with facility layout problems.

5. Acknowledgments

We wish to thank Professor R.L. Francis for his careful review and valuable comments.

References

1. C.R. Aragon, D.S. Johnson, L.A. McGeoch, and C. Shevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning", Operations Research, 39, 3, 1991, pp. 378-406.
2. G.C. Armour, and E.S. Buffa, "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities", Management Science, 9, 1963, pp. 294-309.
3. P. Banerjee, and Y. Zhou, "Facilities Layout Design Optimization with Single Loop Material Flow Path Configuration", International Journal of Production Research, 33, 1995, pp. 183-203.
4. P. Banerjee, Y. Zhou, and B. Montreuil, "Genetically Induced Optimization of Facilities Layout Design", Technical Report TR-92-18, Dept. of Mechanical Engineering, University of Illinois, Chicago, 1992.
5. R. Battiti and G. Tecchiolli, "Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU", Microprocessors and Microsystems, 16, 1992, pp. 351-367.
6. M.S. Bazaraa, "Computerized Layout Design: A Branch and Bound Approach", AIIE Transactions, 7, 1985, pp. 432-438.
7. M.S. Bazaraa, and M.D. Sherali, "Benders' Partitioning Scheme Applied to a New Formulation of the Quadratic Assignment Problem", Naval Research Logistics Quarterly, 27, 1, 1980, pp. 29-41.
8. Y.A. Bozer, R.D. Meller, and S.J. Erlebacher, "An Improvement-type Layout Algorithm for Multiple Floor Facilities", Management Science, 40, 1994, pp. 918-932.
9. R.E. Burkard, and T. Bonniger, "A Heuristic for Quadratic Boolean Program with Applications to Quadratic Assignment Problems", European Journal of Operational Research, 13, 1983, pp. 374-386.
10. R.E. Burkard, and F. Rendl, "A Thermodynamically Motivated Simulation Procedure for Combinatorial Optimization Problems", European Journal of Operational Research, 17, 1984, pp. 169-174.
11. D.T. Connolly, "An Improved Annealing Sceme for the QAP", European Journal of Operational Research, 46, 1990, pp. 93-100.
12. L. Davis, "Job Shop Scheduling with Genetic Algorithms", Proceedings of an International Conference on Genetic Algorithms and their Applications, Pittsburgh, PA, 1985, pp. 136-140.
13. C.A. Floudas, and P.M. Pardalos (eds.), *State of the Art in Global Optimization*, Kluwer Academic Publishers, 1996.
14. R.L. Francis, L.F. McGinnis, and J.A. White, *Facility Layout and Location: An Analytical Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
15. A. Ferreira, and P.M. Pardalos (eds.), *Solving Combinatorial and Optimization Problems in Parallel*, Springer-Verlag, 1996.
16. D.E. Goldberg, *Genetic Algorithms In Search, Optimization, and Machine Learning*, Addison-Wesley,1989.
17. B.L. Golden, and C.C. Skiscim, "Using Simulated Annealing to Solve Routing and Location Problems", Naval Research Logistics Quarterly, 33, 1986, pp. 261-279.
18. J.J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", IEEE Transactions on Systems, Man, and Cybernetics, 16, 1986, pp. 122-128.

19. J.J. Grefenstette, R. Gopal, B.J. Rosmaita, and D. Van Gucht, "Genetic Algorithms for the Traveling Salesperson Problem", Proceedings of an International Conference on Genetic Algorithms and their Applications, Pittsburgh, PA, 1985, pp. 160-168.
20. M.M.D. Hassan, G.L. Hogg, and D.R. Smith, "SHAPE: A Construction Algorithm for Area Placement Evaluation", International Journal of Production Research, 24, 1986, pp. 1283-1295.
21. S.S. Heragu, and A.S. Alfa, "Experimental Analysis of Simulated Annealing based Algorithms for the Layout Problem", European Journal of Operational Research, 57, 1992, pp. 190-202.
22. S.S. Heragu, and A. Kusiak, "Efficient Models for the Facility Layout Problem", European Journal of Operational Research, 53, 1991, pp. 1-13.
23. J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
24. R. Horst, and P.M. Pardalos, *Handbook of Global Optimization*, Kluwer Academic Publishers, 1995.
25. R. Horst, P.M. Pardalos, and N.V. Thoai, *Introduction to Global Optimization*, Kluwer Academic Publishers, 1995.
26. C.L. Huntley, and D.E. Brown, "A Parallel Heuristic for Quadratic Assignment Problem", Computers and OR, 18, 1991, pp. 275-289.
27. S. Jajodia, I. Minis, G. Harhalakis, and J.M. Proth, "CLASS: Computerized Layout Solutions using Simulated Annealing", International Journal of Production Research, 30, 1, 1992, pp. 95-108.
28. P. Jog, J.Y. Suh and D. Van Gucht, "Parallel Genetic Algorithms Applied to the Traveling Salesman Problem", SIAM Journal of Optimization, 1, 1991, pp. 515-529.
29. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Shevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning", Operations Research, 37, 6, 1989, pp. 865-892.
30. B.K. Kaku, and G.L. Thompson, "An Exact Algorithm for the General Quadratic Assignment Problem", European Journal of Operational Research, 23, 1986, pp. 382-390.
31. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing", Science, 220, 1983, pp. 671-680.
32. S. Koakutsu, and H. Hirata, "Genetic Simulated Annealing for Floorplan Design", Chiba University, Japan, 1992.
33. T.C. Koopmans, and M. Beckmann, "Assignment Problems and the Location of Economic Activities", Econometrica 25, 1, 1957, pp. 53-76.
34. P. Kouvelis, and W. -C. Chiang, "A Simulated Annealing Procedure for Single Row Layout Problems in Flexible Manufacturing Systems", International Journal of Production Research, 30, 4, 1992, pp. 717-732.
35. P. Kouvelis, W. -C. Chiang, and J. Fitzsimmons, "Simulated Annealing for Machine Layout Problems in the Presence of Zoning Constraints", European Journal of Operational Research, 57, 1992, pp. 203-223.
36. A. Kusiak, and S.S. Heragu, "The Facility Layout Problem", European Journal of Operational Research, 29, 1987, pp. 229-251.
37. P.J.M. Van Laarhoven, and E.H.L. Aarts, *Simulated Annealing. Theory and Applications*, D. Reidel Publishing Co., 1987.
38. P.S. Laursen, "Simulated Annealing for the QAP-Optimal Tradeoff Between Simulation Time and Solution Quality", European Journal of Operational Research, 69, 1993, pp. 238-243.
39. E.L. Lawler, "The Quadratic Assignment Problem", Management Science, 9, 4, 1963, pp. 586-599.
40. Y. Li, "Heuristic and Exact Algorithms for the Quadratic Assignment Problem", Ph.D Thesis, The Pennsylvania State University, Dept. of Computer Science, 1992.
41. W.G. Macready, A.G. Siapas, and S. A. Kauffman, "Criticality and Parallelism in Combinatorial Optimization", Science, 271, 1996, pp. 56-59.
42. R.D. Meller, and Y.A. Bozer, "A new Simulated Annealing Algorithm for the Facility Layout Problem", Technical Report 91-29, Department of Industrial and Operations Engineering, The University of Michigan, 1991.
43. R.D. Meller, and K.-Y. Gau, "The Facility Layout Problem: A Review of Recent and Emerging Research", Auburn University, Auburn, AL, 1995.
44. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines", Journal of Chemical Physics, 21, 1953, pp. 1087-1092.
45. P.B. Mirchandani, and R.L. Francis (eds.), *Discrete Location Theory*, Wiley-Interscience Series in Discrete Mathematics and Optimization, 1990.

46. B. Montreuil, "A Modeling Framework for Integrating Layout Design and Flow Network Design", Proceedings of the MHI/CICMHE Material Handling Research Colloquium, Hebron KY, 1990, (to appear in IIE Transactions).
47. H. Muhlenbein, "Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization", *Lecture Notes in Computer Science*, 565, 1989, pp. 398-406.
48. H. Muhlenbein, M. Schomisch and J. Born, "The Parallel Genetic Algorithm as Function Optimizer", Proceedings on an International Conference on Genetic Algorithms, 1991.
49. C.E. Nugent, T.E. Vollmann, and J. Ruml, "An Experimental Comparison of Techniques for the Assignment of Facilities to Locations", Operations Research, 16, 1968, pp. 150-173.
50. P.M. Pardalos, Y. Li, and K. A. Murthy, "Computational Experience with Parallel Algorithms for Solving the Quadratic Assignment Problem", *Computer Science and Operations Research: New Developments in their Interface*, O. Balci et al. (eds.), Pergamon Press, 1992, pp. 267–278.
51. P.M. Pardalos, and G. Guisewite, "Parallel Computing in Nonconvex Programming", Annals of Operations Research, 43, 1993, pp. 87–107.
52. P.M. Pardalos, A.T. Phillips, and J.B. Rosen, *Topics in Parallel Computing in Mathematical Programming*, Science Press, 1992.
53. P. M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende, "A Parallel GRASP Implementation for the Quadratic Assignment Problem", *Solving Irregular Problems in Parallel: State of the Art*, A. Ferreira and J. Rolim (eds.), Kluwer Academic Publishers, 1995, pp. 111-128.
54. P. M. Pardalos, L.S. Pitsoulis, T.D. Mavridou, and M.G.C. Resende, "Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP", *Lecture Notes in Computer Science*, 980, Springer-Verlag, 1995, pp. 317-331.
55. P.M. Pardalos, M.G.C. Resende, and K.G. Ramakrishnan (eds.), *Parallel Processing of Discrete Optimization Problems*, DIMACS Series, 22, American Mathematical Society, 1995.
56. P.M. Pardalos, F. Rendl, and H. Wolkowicz, "The Quadratic Assignment Problem: A Survey and Recent Developments", *Quadratic Assignment and Related Problems*, P.M. Pardalos, and H. Wolkowicz (eds.), DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 16, American Mathematical Society, 1994, pp. 1-42.
57. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York, 1986.
58. J.D. Schaffer, R.A. Caruana, L.J. Eshelman, and R. Das, "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization", Proceedings on the Third International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1989, pp. 51-60.
59. R. Sharpe, and B.S. Maksjo, "Facility Layout Optimization Using the Metropolis Algorithm", Environment and Planning B: Planning and Design, 12, 1985, pp. 443-453.
60. R. Sharpe, B.S. Maksjo, J.R. Mitchell, and J.R. Crawford, "An Interactive Model for the Layout of Buildings", Applied Mathematical Modeling, 9, 3, 1985, pp. 207-214.
61. A. Souilah, "Simulated Annealing for Manufacturing Systems Layout Design", European Journal of Operational Research, 82, 1995, pp. 592-614.
62. G. Suresh, and S. Sahu, "Multiobjective Facility Layout Using Simulated Annealing", International Journal of Production Economics, 32, 1993, 239-254.
63. K.Y. Tam, "Genetic Algorithms, Function Optimization and Facility Layout Design", European Journal of Operational Research, 63, 1992, pp. 322-346.
64. K.Y. Tam, "A Simulated Annealing Algorithm for Allocating Space to Manufacturing Cells", International Journal of Production Research, 30, 1, 1992, pp. 63-87.
65. D.M. Tate, and A.E. Smith, "Unequal-Area Facility Layout by Genetic Search", IIE Transactions, 27, 1995, pp. 465-472.
66. M.B. Teitz, and P. Bart, "Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph", Operations Research, 16, 1968, pp. 955-961.
67. J.A. Tompkins, and J.A. White, *Facilities Planning*, Wiley, New York, 1984.
68. X. Tong, "SECOT: A Sequential Construction Technique for Facility Design", Doctoral Dissertation, University of Pittsburgh, 1991.
69. D.J. Van Camp, M.W. Carter, and A. Vannelli, "A Nonlinear Optimization Approach for Solving Facility Layout Problems", European Journal of Operational Research 57, 1991, pp. 174-189.

70. M.R. Wilhelm, and T.L. Ward, "Solving Quadratic Assignment Problems by Simulated Annealing", IIE Transactions, 19, 1987, pp. 107-119.

Computational Optimization and Applications 7, 127–142 (1997)

Sequential Quadratic Programming Methods for Large-Scale Problems

WALTER MURRAY*
Systems Optimization Laboratory, Department of Operations Research, Stanford University

Abstract. Sequential quadratic (SQP) programming methods are the method of choice when solving small or medium-sized problems. Since they are complex methods they are difficult (but not impossible) to adapt to solve large-scale problems. We start by discussing the difficulties that need to be addressed and then describe some general ideas that may be used to resolve these difficulties. A number of SQP codes have been written to solve specific applications and there is a general purposed SQP code called SNOPT, which is intended for general applications of a particular type. These are described briefly together with the ideas on which they are based. Finally we discuss new work on developing SQP methods using explicit second derivatives.

Keywords: nonlinearly constrained minimization, quadratic programming, large-scale optimization

1. Introduction

The problem of interest is the following:

$$\begin{array}{ll} \underset{x\in\Re^n}{\text{minimize}} & F(x) \\ \text{s.t.} & c(x) \geq 0, \end{array} \qquad \text{NP}$$

where $F: \Re^n \to \Re$ and $c: \Re^n \to \Re^m$. If second derivatives are not known, computing x^*, a point satisfying the *first-order KKT conditions* for NP is the best that can be assured. Otherwise, it is possible to assure finding a point satisfying the second-order KKT conditions. Our particular interest here is when n and possibly m are large.

When solving large problems the precise form of what are mathematically equivalent forms of NP is important. In practice the constraints may be a mixture of linear and nonlinear inequality constraints, simple bounds on the variables and equality constraints. Also there may be upper as well as lower bounds on some constraints and some variables may appear only linearly in the problem. Such trivial mathematical considerations may assume quite large proportions in some practical algorithms. Success in solving large problems often depends on attention to a myriad of small details in the definition of the model and algorithm. The interface for software for large problems is more complex since the user

*Research supported by the National Science Foundation Grant DMI-9500668; the Office of Naval Research Grant N00014-96-1-0274.

needs to specify more details of their problem in order for the software to be efficient. The level of description in this paper is such that such details will not be given much consideration, but such details are vital in practice.

2. A basic sequential quadratic programming method (SQP)

Typically SQP methods generate a sequence of points $\{x_k\}$ converging to a solution, by solving at each point, x_k, a quadratic program (QP), which for problems in the form of NP will be of the form

$$\boxed{\begin{array}{ll} \underset{p\in\mathbb{R}^n}{\text{minimize}} & \nabla F(x_k)^T p + \frac{1}{2} p^T H_k p \\ \text{s.t.} & c(x_k) + \nabla c(x_k) p \geq 0 \end{array}} \qquad \text{QP}$$

for some positive definite matrix H_k. Let p_k (referred to as the search direction) denote the unique solution to QP. We define $x_{k+1} \equiv x_k + \alpha_k p_k$, where the steplength α_k is chosen to achieve a reduction in a *merit* function. The matrix H_k is usually an approximation to the Hessian of the Lagrangian function

$$L(x, \lambda) = F(x) - \lambda^T c(x),$$

where λ are estimates to the Lagrange multipliers. Since the Hessian of the Lagrangian function is not positive definite this clearly poses some difficulties although such difficulties arise even when n is small. In most cases the approximation is obtained using quasi-Newton updates to an initial approximation that is usually diagonal.

An often unappreciated feature of SQP methods is that they *automatically* take advantage of linearity in a linear constraint. For example, if the initial estimate satisfies a linear equality or inequality constraint then so do all subsequent iterates. Such constraints are also automatically excluded from the merit functions. Once a linear constraint is satisfied it is never again violated. Nonetheless in both large and small problems there is an advantage to taking specific note of whether a constraint is linear. Although such a distinction may not impact the sequence of iterates it does impact the effort to compute the iterates and is therefore important in the large-scale case.

Merit functions

Merit functions are a means of obtaining the required convergence. Since the iterates may not be feasible some means of defining whether one point is better than another is required. The first merit function proposed (see [4] and [31]) was the quadratic or l_2 penalty function. Later the l_1 penalty function was used (see [27] and [38]). The l_1 penalty function has the advantage (as would any merit function based on a norm) of requiring only a finite penalty parameter. However, such merit functions used alone may inhibit the rate of convergence. It is also our experience that they are not as efficient as smooth merit functions regardless of this deficiency. Smooth merit functions have been defined based on exact penalty functions

(see [14]). However, such an approach requires restrictive assumptions about the rank of the Jacobian matrix and uses problem derivatives in the definition of the merit function. If either of these two restrictions are to be avoided it requires searching in a higher dimensional space. In [22] and [26] the following merit function was proposed

$$M(x, \lambda, s, \rho) = F(x) - \lambda^T(c - s) + \tfrac{1}{2}\rho(c - s)^T(c - s),$$

where the search is now in the triple space of x, λ and s. Interestingly the idea of searching in the space of the slack and dual variables has become common practice in the application of barrier (interior-point) methods to linear programs. The above merit function is used in NPSOL [22] and SNOPT, a new SQP code for large problems (see [20] and [21]).

3. Large problems

SQP algorithms are viewed by many as the best approach (see [24]) to the solution of NP when n is small or moderate (say less than 1000). In this sense "best" means it requires the least number of evaluations of the users functions $F(x)$ and $c(x)$. For small problems the effort to solve the QP subproblems is rarely relevant provided the method of solution is done with reasonable care. To adapt SQP methods to solve large problems requires being able to solve efficiently the resulting large QP subproblems. One approach taken in Murray and Prieto [32] is to show that a suitable search direction may be obtain without the need to solve completely the QP subproblem. They also prove convergence of SQP methods that incorporate a large degree of flexibility in the definition of the algorithm in order to accommodate the type of adaptation that is necessary when designing algorithms for large problems. Nonetheless even to solve the subproblem incompletely is likely to require the use of sparse matrix technology. It is also necessary to store the matrices defining the QP problem in compact form. A basic assumption made is that $\nabla c(x_k)$ is a sparse matrix and hence there are a number of ways of storing this matrix in compact form. In general when H_k is obtained by a quasi-Newton approximation it will not be sparse so some modification to the standard quasi-Newton approach is necessary.

One key to a successful SQP algorithm for large problems is a fast algorithm to solve (or partially solve) the QP subproblems. Solving a sequence of related QP problems is not quite the same as solving a single QP problem. Usually after a few nonlinear iterations the active set of the QP subproblem changes only slightly. The active set of one QP subproblem is then close to that of the following subproblem. It is important that full advantage is taken of this information.

It should be understood that it is unlikely that a single SQP code will prove all powerful. Indeed it is unlikely a single QP code that is efficient under all circumstances will emerge. Efficiency for a QP algorithm is often dependent on the nature of the solution. For example, in many problems the degree of freedom in the problem is small. For such problems methods that assume this property will be more efficient than methods that do not. Moreover the difference is significant. Such methods, however, are likely to perform poorly when the property is not true. It can often be assessed prior to the solution whether or not the property holds. For example, if the number of equality constraints is almost equal to the number

of variables then the degree of freedom in the problem must be small. Similarly it will be known when the number of constraints is small since then m is small. Again one can design an SQP method specifically for that class of problem. Other issues impacting the design are the relative cost of the linear algebra operations compared to that of evaluating the user's functions and derivatives (if derivatives are available). Again such information may be obtained prior to solving a problem.

4. Compact forms for H_k

As we have already mentioned for large problems the standard quasi-Newton approach is no longer adequate. Much of the work done on extending quasi-Newton methods to large problems has been directed at the unconstrained case. Although the need to obtain some compact expression for an approximation to the curvature of a function is the same for both unconstrained and constrained problems the use of this information in constrained problems is more complex. Moreover the nature of curvature for constrained problems is different. Research in this area may be classified under the following general approaches:

(i) Limited memory approximations.
(ii) Sparse quasi-Newton approximations.
(iii) Group partial separability.
(iv) Projected quasi-Newton approximations.

Limited memory approximations

Limited memory approximations are easily understood and have been the subject of considerable research and experimentation (see [6, 8, 9, 16, 18, 28, 36]). Instead of H_k being a result of k updates only a limited number of updates are used. There are various ways this may be done. We could for instance keep the last t updates, where t is small, say less than 25 and typically less than ten. The key point is that H_k is not stored *explicitly* rather the initial approximation is stored (usually a diagonal matrix) together with the information required to perform the updates. Depending on the form of the update the storage requirements are at worst $2nt$ and at best nt. An alternative to retaining information on the last t updates is to restart every t iterations. This has some advantages for problems with linear constraints. Yet a third alternative is to try and retain what is thought to be the t most useful updates.

Much of the work on limited memory methods has been directed at the unconstrained case and it is likely the experience there does not fully carry over to the nonlinearly constrained case (it may be better). Although simple in concept there are many ways to implement a limited memory method. How the updates are stored and in what form impacts the cost of operations with H_k. Since H_k is not stored explicitly this limits the methods of solution of the QP subproblems. This approach has the property that it does not depend on the Hessian of the Lagrangian being sparse. Although one could argue that this is likely it does save a user establishing that fact or possibly the need to cast their problem in a form where it is true.

Sparse quasi-Newton approximations

The success of quasi-Newton methods on small dense unconstrained problems prompted considerable research on methods that could update an approximation to H_k that has the same sparsity pattern as $\nabla^2 F(x)$. For constrained problems we are concerned with the sparsity pattern of $\nabla^2 L(x, \lambda)$. Despite considerable work few useful results have emerged. Moreover, it is computationally costly to perform the updating. A key difficulty is to preserve the correct sparsity pattern and the property of inherited positive definiteness. Recently (see [17]) a number of workers have returned to this line of research so some new more useful results may be forthcoming.

This approach determines an explicit representation of $H(x)$ so it does not limit the methods of solution of the QP subproblem. However, it is only applicable to problems for which $\nabla^2 L(x, \lambda)$ is sparse.

Group partial separability

Another approach advocated for determining a compact approximation to H_k is to use the property of *partial separability* or *group partial separability* (e.g., [10, 12, 13]). Quite possibly this approach may result in H_k not being known explicitly ($\nabla^2 F(x)$ may not be sparse).

A function is said to be group partially separable if it can be written in the form

$$F(x) = \sum_i g_i(y_i(x)), \tag{4.1}$$

where each function $y_i(x)$ has the form $y(x) = a^T x + \sum_j f_j(x)$, with each function $f_j(x)$ involving its own (small) subset of the variables x. Obviously such a representation is not unique for a given function $F(x)$. The basic idea is that instead of approximating the Hessian of $F(x)$ directly the Hessians of g_i with respect to y_i may be approximated and that for $\nabla^2 F(x)$ constructed from these matrices. By assumption the function $g_i(y_i)$ is a function of only a few variables, which implies the approach gives a compact representation of H_k. In general H_k may not be sparse since $\nabla^2 F(x)$ need not be sparse. If that is the case it limits the use of H_k to be indirect when solving the QP subproblem

We have not favored direct use of group partial separability, since it seems likely that many users would not know how to define g_i, $y_i(x)$, etc., and the user interface utilizing this information if available is inevitably complex [11]. It seems to us to be an idea primarily suited to unconstrained or linearly constrained problems. When a *nonlinearly constrained* problem is formulated, such convoluted functions are unlikely to occur (they are often a consequence of eliminating nonlinear constraints) because the modeler always has the option of defining extra variables and constraints. For example, the problem

$$\underset{x}{\text{minimize}} \; F(x) \tag{4.2}$$

could be treated as

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & F(y) = \sum_i g_i(y_i) \\ \text{subject to} & y_i = a_i^T x + \sum_j f_j(x). \end{array} \tag{4.3}$$

Of course the problem is now constrained, but this is of no consequence if there are other nonlinear constraints. It is not that the transformation from (4.2) to (4.3) need be made; rather, the second formulation is the *natural* one. Indeed, forming convoluted functions such as (4.1) is inherently dangerous. For example, consider the impact on the derivatives of eliminating the variable y_j and the constraint $x_j^2 + y_j^2 = 1$ from a problem. It may well be better to solve a constrained problem rather than a difficult unconstrained problem. A good rule of thumb when solving large problems is that it is better to have a formulation of the problem that has more variables and more constraints.

The case for partial separability is much stronger since it requires only identifying that the function can be written in the form

$$F(x) = \sum_i g_i(x),$$

where it is assumed each function $g_i(x)$ is a function of only a few variables. It is possible that the user need not provide the above form for the problem since an attempt to identify such a form can be made using information about the sparsity of the Hessian matrix (this is a non-trivial task and finding the *best* form is even harder).

Consider now problems of the form NP. In the constrained case we wish to approximate the Hessian of the Lagrangian function

$$L(x) = F(x) - \sum_i \lambda_i c_i(x)$$

which is *naturally* in partially separable form provided $F(x)$ and $c_i(x)$ are functions of only a few variables. Let $J(x) = \nabla c(x)$. If $J(x)$ is *sparse* (our basic assumption), then in general each constraint function must involve only a few variables (some of which may appear linearly). The user already supplies the sparsity structure of $J(x)$, and from it we may deduce the set of variables involved in each function $c_i(x)$. It is also reasonable for a user to be asked to specify the variables that appear nonlinearly in the objective. For many problems such as those arising from optimal control, this is a trivial requirement. (Indeed many problems have linear objectives.) Such a requirement is already a feature of the codes MINOS and SNOPT. These codes require the variables that occur nonlinearly in the objective *and* constraints to be specified. If these variables are listed first then it is known that H_k has the form

$$\begin{pmatrix} \bar{H}_k & 0 \\ 0 & 0 \end{pmatrix}. \tag{4.4}$$

For many problems the number of variables occuring nonlinearly in the problem is much smaller than the total number of variables. If it is small enough then $\bar{H}_k$ may be stored as

a dense matrix. It also implies there are lots of constraints active at the solution and this knowledge impacts the choice of method for solving the QP subproblems.

Using a quasi-Newton update procedure we can approximate $H_0 \approx \nabla^2 F(x)$ and $H_i \approx \nabla^2 c_i(x)$ and then form the appropriate linear combination $H_0 - \sum_i \lambda_i H_i$ to obtain an approximation to $\nabla^2 L$.

To illustrate, consider obtaining the approximation $H_i \approx \nabla^2 c_i(x)$. For some permutation P_i, we have

$$P_i \nabla^2 c_i(x) P_i^T = \begin{pmatrix} \tilde{H}_i(x) & 0 \\ 0 & 0 \end{pmatrix},$$

where $\tilde{H}_i(x)$ is a dense matrix of order n_i, the number of variables appearing in $c_i(x)$. Assuming n_i is small, we may implement quasi-Newton updates by maintaining a *dense* approximation $\bar{H}_i \approx \tilde{H}_i(x)$. Since the search directions have no special properties it is likely that the rank-one update will be most suitable. It may be that a reasonable approximation to each individual Hessian will be obtained in only a few iterations. For example, if c_i is a quadratic function of n_i variables, its Hessian approximation will be correct after n_i iterations. Typically we expect n_i to be less than 10.

An important feature of this approach is that the individual Hessian approximations are independent of the Lagrange multipliers estimates. If the multiplier estimates are poor at some stage, then any quasi-Newton method approximating $\nabla^2 L$ directly will take many iterations to recover even if the multiplier estimates improve immediately. By contrast, the approach based on partial separability has the potential to recover immediately. In general the resulting approximation H_k will not be positive semidefinite and this has consequences as to the definition and method of solution of the QP subproblem. Many of the issues that arise due to the lack of convexity are similar to those that arise when using exact second derivatives and are discussed in Section 6.

Projected quasi-Newton approximations

Given a nonsingular matrix Q_k and an approximation $Q_k^T H_k Q_k$ to the projected Hessian $Q_k^T \nabla^2 L(x_k) Q_k$ then it is not necessary to know H_k explicitly to implement an SQP method. At first sight it may not seem we have made a significant step by requiring an approximation to $Q_k^T H_k Q_k$ rather than H_k. It will be seen in Section 5 that provided Q_k is chosen appropriately this strategy fits well with the null-space method for solving the QP subproblem in the dense case. The same is true in the large-scale case provided care is exercised in how Q is defined. The basis of this approach (and the null-space method of solving QPs) is the following observation. Suppose we have the QP problem

$$\begin{array}{ll} \underset{p \in \mathbb{R}^n}{\text{minimize}} & g^T p + \frac{1}{2} p^T H p \\ \text{s.t.} & Jp = 0 \end{array}$$

then the solution is given by

$$p^* = Z p_Z, \qquad \text{where} \quad p_Z = -(Z^T H Z)^{-1} Z^T g \tag{4.5}$$

and Z is a basis for the null-space of the rows of J. The key point of this result is that *only* the matrix $Z^T H Z$ is required to define the solution and not H. If the dimension of Z is small then $Z^T H Z$ may be stored as a dense matrix. If we define $Q \equiv (Z\ Y)$, where Y is chosen to make Q nonsingular then we could define $Q^T H Q$ as

$$Q^T B Q = \begin{pmatrix} Z^T B Z & 0 \\ 0 & D \end{pmatrix},$$

where D is a diagonal matrix. Of course if Q is a dense matrix or cannot be stored in compact form we have gained nothing. Fortunately it is possible to define Q in such a manner as to make operations with Q and Q^{-1} efficient. Moreover, when a null-space method is used to solve the QP subproblem no additional storage is required.

In practice when the constraints are nonlinear we do not have zero on the right-hand-side of the QP constraints nor in general do we have just equality constraints and these differences raise complications. The adaptations require are wedded to how the solution of the QP subproblem is found and are described in the next section.

5. Solving large QP problems

We shall assume that the QP is convex. Note that unlike the case of dense problems it may be that H_k is structurally singular (4.4). This limits (but does not eliminate) the use of the dual when solving the QP subproblem.

There are a variety of methods to solve large QP problems and which is best will depend on the characteristics of the QP and its solution. We shall consider four basic approaches (there are others):

- A Schur-complement method.
- A null-space method.
- A range-space method.
- A barrier (interior-point) method.

The first three are active-set methods and differ only in how the relevant linear equations are solved. Which of the three active-set methods to use depends largely on the number of active constraints. Null-space methods are efficient when the number of active constraints is almost the same as the number of variables. Range-space methods are efficient when there are only a few active constraints. If neither of these conditions hold or is *known* to hold then the Schur-complement approach is recommended. The use of a barrier algorithm to solve the subproblem as opposed to applying a barrier approach to the nonlinear problem is likely to be preferred when the user's functions are expensive to compute. Usually applying a barrier approach to the original nonlinear problem results in a substantial increase in the number of nonlinear iterations. In theory the number of nonlinear iterations is independent of *how* the QP subproblem is solved. In practice there are likely to be small but essentially negligible differences on most problems. Whether a barrier algorithm is preferable to an active-set method to solve the QP is more difficult to decide. Barrier methods may prove

very efficient when the number of constraints is small. They are likely to be inefficient when there are many variables on their bounds.

It is sometimes beneficial if the QP is in standard form although this is not always strictly necessary. This helps in the barrier approach since the ill-conditioning in the resulting subproblems is then benign (see [37]) and in the null-space method it simplifies the updating procedures.

In all four approaches we are required at each iteration to solve the KKT equations, which are of the form

$$K\begin{pmatrix} p \\ -\pi \end{pmatrix} = -\begin{pmatrix} g \\ 0 \end{pmatrix}, \qquad \text{where} \quad K \equiv \begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix}, \tag{5.1}$$

where H is the Hessian approximation and A is some set of rows from the constraint Jacobian. The three active-set methods differ in how they partition these equations in order to solve them.

A Schur-complement method

If K_0 denotes the initial KKT matrix the Schur-complement algorithm (see [25]) first forms a sparse factorization of this matrix to determine the first iteration. Subsequent iterations are performed using this factorization and a dense factorization of the Schur-complement of the initial matrix. In order to do that it is necessary to construct in the tth QP iteration a KKT matrix, K_t, that is an augmented form of the initial matrix. The Schur complement grows monotonically and since this is in general a dense matrix it is necessary to replace periodically the initial factors of K_0 with a factorization of the current KKT matrix. The method fits well with the incomplete solution approach since then the need to perform a refactorization is reduced.

The application of the standard Schur-complement algorithm to the special case of solving the sequence of QP problems that arises in SQP methods is straightforward (see [23] and [25]). It is not strictly necessary to be able factor K_0. What is required is to be able to solve linear systems of the form $K_0 = v_j$ relatively quickly. In general two such systems require solving at each QP iteration. Such a requirement usually eliminates the use of iterates procedures, but not more elaborate factorization approaches such as also using a Schur-complement approach when solving these systems.

A range-space method

This is similar in some respects to the Schur-complement algorithm except instead of taking the Schur complement of the initial KKT matrix we use in the tth QP iteration the Schur complement of H_t, where H_t is the Hessian portion of K_t. Note that this approach requires H_t to be nonsingular. Since the Schur complement is in general a dense matrix the approach is only feasible when the Schur-complement is limited in size as it would be if we have few constraints. The fact the Schur-complement is limited in size implies $\nabla^2 L(x)$ is nearly full rank hence assuming that H_t is full rank is not unreasonable. It is necessary with

this approach to solve many systems with H_t and while iterative methods are possible the approach is best suited to the case where solves with H_t are cheap and this usually implies H_t is sparse and possibly well structured. The adaptation to the case of solving a sequence of QPs is again straightforward. Unlike the Schur-complement approach where the Schur complement is built from scratch as iterations proceed we need to form an initial Schur complement and factor it. In general there will never be a need to discard these factors due to them becoming too large since a given KKT matrix does not have to be an augmentation of the preceeding KKT matrix. Consequently, the Schur complement may grow or shrink in size as the iterations proceed and there is an apriori limit on its growth. Obviously if a problem has only a few constraints it is known apriori that the Schur complement will not be large. Only general constraints are of significance since bounds on the variable may be used to solve a KKT system of reduced size.

The range-space approach is occasionally suitable for problems for which the Schur complement is large provided it is sparse. Such problems do arise when $\nabla^2 L(x)$ has such a simple structure that $(\nabla^2 L(x))^{-1}$ is sparse. Usually in such cases $\nabla^2 L(x)$ may be computed or approximated directly by a small number of finite differences. The simplest case is when $\nabla^2 L(x)$ is a diagonal matrix. Sometimes a problem may be reformulated by adding some additional variables and constraints to obtain a such a simple structure. This is the case when $\nabla^2 L(x)$ is a low rank change to a diagonal matrix.

A null-space method

The null-space method is based on the Eqs. (4.5). It may be used either in conjunction with a projected Hessian approach or with a direct approximation to the Hessian. Approximating the reduced Hessian has two benefits, it requires storing only a small matrix and it facilitates the solution of the QP subproblem, which is solved very efficiently if the predicted active set is correct. In the approach taken by Gill et al. [20] and Eldersveld [15] the matrix Q is never required explicitly and the null-space algorithm may be efficiently implemented if solves with Q and $Q^T H Q$ are efficient. As before we define $Q = (Z\ Y)$, but now the *first* columns of Z span the null space of the Jacobian of the current active set. By allowing the columns of Z to span a bigger space than that defined by the null-space of the active set we are able to cope with some changes to the active set. If Z is not too large then $Q^T H Q$ is still cheap to operate with and requires little storage. Moreover, and this is a key point, the solution of the QP is relatively trivial to compute since it is not necessary to form $Z^T H Z$ (in practice the Cholesky factor of this matrix is recurred).

We still require a sparse representation for Q. The usual sparse representation for Z is to make use of the matrix

$$Z = \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix},$$

where B is a basis from the Jacobian of the active set and S is the remaining columns of the Jacobian plus some columns of the full Jacobian corresponding to variables currently on their bounds. The matrix Z is not stored explicitly instead the sparse LU factors of B are stored. Since the algorithm requires only operations by Z and Z^T this suffices.

It is convenient when using a null-space method to assume the problem is in standard form. If the active set changes the columns of S and B may also change, but not necessarily. It is shown in [21] that it is important to include any columns corresponding to slack variables in B. This is because it is known that the Lagrangian function is linear with respect to the slack variables, which raises the spectra that the reduced Hessian could be singular unless the slack columns are in B.

The choice of Y is less clear. In [20] and [15] the matrix Y is represented by

$$Y = \begin{pmatrix} B^{-1} \\ 0 \end{pmatrix}.$$

It can then be shown that Q^{-1} is given by

$$Q^{-1} = \begin{pmatrix} 0 & I \\ B & S \end{pmatrix}.$$

A barrier method

In theory barrier methods for convex QP are a simple extension to barrier methods for linear programming (LP). In practice the computational implications of having a quadratic terms significantly alter the algorithm. It is helpful to assume the original nonlinear problem is in standard form, that is the constraints are of the form

$$c(x) = 0 \qquad x \geq 0.$$

It follows that the QP subproblem is in the following standard form.

$$\text{QPSUB} \quad \underset{p \in \mathbb{R}^n}{\text{minimize}} \quad g^T p + \tfrac{1}{2} p^T H p \qquad \text{subject to} \quad Ap = -c, \quad p \geq -x,$$

As we discuss below, this choice of format is crucial.

The logarithmic barrier subproblem for this QP is

$$\underset{p \in \mathbb{R}^n}{\text{minimize}} \quad g^T p + \tfrac{1}{2} p^T H p - \mu \sum_{j=1}^{n} \ln(x_j + p_j) \qquad \text{subject to} \quad Ap = -c,$$

where μ is the barrier parameter.

Newton's method applied to the subproblem results in the need to solve a sequence of KKT systems of the form (5.1). The leading principal submatrix is the Hessian of the barrier function, $H_B = H + \mu D^{-2}$, where $D = \text{diag}(x_j + p_j)$. In the LP case, H_B is the positive-definite diagonal matrix μD^{-2} and the KKT system can be reduced to two systems that define the normal equations and residual vector for the weighted least-squares problem $\min \|Dg - DA^T\pi\|_2$. The least-squares approach is not generally applicable to quadratic problems, since H_B can be indefinite or singular, but in any event we believe that better numerical properties are obtained by treating the KKT system directly. The primal-dual method results in a KKT-type system in which the leading principal submatrix is $H + ZD^{-1}$, where Z is an estimate of dual slack variables.

Benefits of the standard form

It was shown by Murray [31] that if the number of variables on their bounds at the solution is between zero and $n - m$, then $\lim_{\mu \to 0} \kappa(\mu) = \infty$, where $\kappa(\mu)$ is the condition of the KKT matrix at the minimizer of the subproblem. We can now show that this ill-conditioning, though still present, is benign under certain circumstances [37].

It is easy to see that the ill-conditioning is harmful if the problem is not in the proposed format. Suppose that the problem includes some general *inequality* constraints $\hat{A}x \geq \hat{b}$ as well as bounds $x \geq 0$. The Hessian of the barrier function is then

$$H_B = H + \mu \hat{A}^T \hat{D}^{-2} \hat{A} + \mu D^{-2}, \tag{5.2}$$

where $\hat{D} = \text{diag}(\hat{a}_i^T x - \hat{b}_i)$ and $D = \text{diag}(x_j)$. If an inequality constraint $\hat{a}_i^T x \geq \hat{b}_i$ is active at the solution, it can be shown that $\lim_{\mu \to 0} \mu / \hat{D}_{ii}^2 = \infty$, and precision in H must be lost during the *formation* of H_B. The same situation must arise if the problem is in standard form and the optimal value of x_j is zero; however, only the precision of the jth diagonal of H is affected. Analysis shows that the condition number governing the sensitivity of the KKT matrix is κ_{BS}, the condition of the KKT matrix of the original QP in just the basic and superbasic variables. It is possible to solve systems of the form $H_B x = y$ accurately but to do so requires either transforming the system into solving KKT-type systems or decomposing the problem into solving two systems of equations. In either case it is necessary to divide the rows of A into two types. The benefit of the standard form is that such special techniques are not required.

Solving the KKT system

If the solution of a system is insensitive to small perturbations in the matrix, it does not follow that all solution methods will be satisfactory. However, a direct method using an LU or LBL^T factorization preserves the benign nature of the ill-conditioning.

Warm starts

We are not solving a single QP but a sequence of related QPs. For active-set methods "warm starts" mainly means using the "old" active set as an initial trial active set. For barrier methods warm starts are somewhat more complex. It may be that once the active set settles down it is worth switching to an active-set method.

6. Methods using second derivatives

It has been our observation over many years that for large problems for which the first derivatives may be computed it is often possible to compute second derivatives (see [7] and [39]). In many practical problems (see Section 7) the Hessian matrix, like the Jacobian matrix, is sparse. Quasi-Newton approximations to the Hessian of the Lagrangian function

are usually positive definite with a bounded condition number. In this way, a strictly convex subproblem is obtained, and if a feasible point exists, a solution exists and is unique. In contrast, the use of exact Hessians or quasi-Newton approximation that are not positive semi-definite presents a number of technical difficulties, most of which stem from the loss of control over the properties of H_k. Defining the Hessian of the QP subproblem as the Hessian of the Lagrangian function leads in general to nonconvex subproblems. On the other hand, there are numerous theoretical and practical benefits to be derived from the explicit use of second derivatives. For example, it is possible to define an algorithm that generates a sequence that converges to a *second-order* KKT point. Also, in practice it has been observed when solving other classes of optimization problems that second-derivative methods usually converge in much fewer iterations than alternative methods. In order to reap all the benefits from the availability of second derivatives, it is necessary to define the search direction other than as the minimizer of QP. The modifications necessary are similar to those required to Newton's method when solving unconstrained optimization.

In the approach adopted in [32] for quasi-Newton methods a search direction was defined based not on a solution of the QP subproblem but on information at a constrained stationary point. Murray and Prieto show in [33] how this approach may be adapted to ensure convergence of the iterates to a second-order KKT point when exact derivatives are used. Clearly information at a stationary point alone will no longer suffice since an active-set method when applied to a nonconvex QP cannot be assured of finding a stationary point. Moreover, the procedure to construct the search direction from information at the stationary point (should one exist) is no longer assured of being a descent direction for the merit function.

In order to prove convergence to a *second-order* KKT point it is also necessary to be able to generate directions of negative curvature for the Hessian of the Lagrangian function. In this case, a conventional linesearch may no longer be adequate, as the termination criteria for this type of search depend on the value of the initial projected gradient, $\nabla F(x_k)^T p_k$, which may be zero or arbitrarily small. In such circumstances the algorithm in [33] makes use of a curvilinear search, based on the model introduced in McCormick [29] and developed by Moré and Sorensen [30].

It is shown in [33] that the method of constructing a search direction given in [32] is only unsatisfactory if a direction of negative curvature is encountered in the QP active-set method or at the stationary point the reduced Hessian is not positive definite. In either case a direction of negative curvature may be computed. It is shown that it is sufficient to compute the direction of negative curvature only at the initial point for the QP. From the direction of negative curvature and the step to the stationary point (if it exists) it can be shown how a suitable search direction may be constructed.

7. Industrial implementations

Several special SQP codes have been written and applied to specific large-scale applications. Starting in the early 80's several codes were developed at GE and used to solve optimal power flow problems. Such problems may have as many as 60,000 variables and 45,000 nonlinear constraints. The original approach taken was to use MINOS ([34] and [35]) as

the QP solver. Later a special QP routine was written based on the Schur-complement approach [25]. A special class of very large structured problems were solved by applying Benders decomposition to the QP subproblem. The master problem was solved by the Schur-complement algorithm. The slave problems were LP's. Explicit second derivatives were available. The user's function and derivatives were relatively cheap to compute. The Harwell code MA27 was used to factor the initial KKT matrix, but a special technique was necessary to enable the ANALYSE phase to obtain a good ordering. The success of SQP methods on OPF problems is both encouraging and surprising. In this class of methods the problem functions and their derivatives are cheap to compute compared to solving the linear systems in the QP subproblems. The reason for the success of the SQP approach was the very small number of nonlinear iterations required.

A sparse SQP method has been developed at Boeing (see [1]) and used on trajectory problems. These problems are not super large, but are sometimes hard to solve. The algorithm is very similar to that developed at GE except the Hessian here is approximated by finite differences. In both cases the Hessian is sparse. Boeing has developed their own sparse linear equation solver. The user's functions are very expensive to evaluate for trajectory problems, which is just the type of problem one expects SQP methods to perform well on relative to alternative approaches.

Several codes have been developed to solve problems in process control in chemical engineering. These problems may be very large (100,000 variables), but are almost always highly constrained (the reduced Hessian is rarely larger than 50). There are almost as many equality constraints as there are variables. The QP subproblems are therefore highly suited to being solved by a null-space algorithm. This has been done in a conventional manner by DMC Corp. Since the number of equality constraints is nearly equal to the number of variables the solution of QP subproblem may be determined by solving a dense QP in a small number of variables (the dimension of the null space of the equality constraints) and a large number of inequality constraints. An algorithm along these lines has been developed by Shell. Such QP subproblems are usually most efficiently solved by solving the dual.

An SQP method based on a barrier function approach has been developed by Power Associates to solve OPF and related problems. Recall that in such problems the user's function and derivative are cheap to evaluate. Consequently, it is better to apply the barrier to the original nonlinear problem, which leads to equality QP subproblems. The work to perform the additional nonlinear iterations is more than off set by the savings in solving the simpler QP subproblems.

Acknowledgments

It is a pleasure to acknowledge the contribution to this paper from the many years of cooperative research with Philip Gill, Michael Saunders and Francisco Prieto. Any new ideas presented here are taken from our current joint research program.

References

1. J.T. Betts and W.P. Huffman, "Path constrained trajectory optimization using sparse sequential quadratic programming," J. of Guidance, Control, and Dynamics, vol. 16, no. 1, pp. 59–68, 1993.

2. J.T. Betts and P.D. Frank, "A sparse nonlinear optimization algorithm," J. Optim. Theory and Applics., vol. 82, pp. 519–541, 1994.
3. L.T. Biegler, J. Nocedal, and C. Schmid, "A reduced Hessian method for large-scale constrained optimization," SIAM J. on Optimization, vol. 5, pp. 314–347, 1995.
4. M.C. Biggs, "On the convergence of some constrained minimization algorithms based on recursive quadratic programming," JIMA, vol. 21, pp. 67–81, 1978.
5. A. Buckley and A. LeNir, "QN-like variable storage conjugate gradients," Mathematical Programming, vol. 27, pp. 155–175, 1983.
6. A. Buckley and A. LeNir, "BBVSCG—A variable storage algorithm for function minimization," ACM Transactions on Mathematical Software, vol. 11, pp. 103–119, 1985.
7. R.C. Burchett, H.H. Happ, and D.R. Vierath, "Quadratically convergent optimal power flow," IEEE Transactions on Power Apparatus and Systems, vol. PAS-103, pp. 3267–3275, 1984.
8. R.H. Byrd, P. Lu, and J. Nocedal, "A limited memory algorithm for bound constrained optimization," Report NAM 07, EECS Department, Northwestern University, 1993.
9. R.H. Byrd, J. Nocedal, and R.B. Schnabel, "Representations of quasi-Newton matrices and their use in limited memory methods," Mathematical Programming, vol. 63, pp. 129–156, 1994.
10. A.R. Conn, N.I.M. Gould, and Ph.L. Toint, "An introduction to the structure of large-scale nonlinear optimization problems and the LANCELOT project," in Computing Methods in Applied Sciences and Engineering, R. Glowinski and A. Lichnewsky (Eds.), SIAM, Philadelphia, pp. 42–54, 1990.
11. A.R. Conn, N.I.M. Gould, and Ph.L. Toint, "An introduction to the standard data input format (SDIF) for nonlinear mathematical programming problems," Département de Mathématique, Facultés Universitaires de Namur, Technical Report 91/8, 1991.
12. A.R. Conn, N.I.M. Gould, and Ph.L. Toint, "LANCELOT: A fortran package for large-scale nonlinear optimization (Release A)," Lecture Notes in Computation Mathematics 17, Springer Verlag: Berlin, Heidelberg, New York, London, Paris and Tokyo, 1992.
13. A.R. Conn, N.I.M. Gould, and Ph.L. Toint, "Large-scale nonlinear constrained optimization: A current survey," Technical Report 94/1, Département de Mathématique, Facultés Universitaires de Namur, 1994.
14. G. Di Pillo, F. Facchinei, and L. Grippo, "An (RQP) algorithm using a differentiable exact penalty function for inequality constrained problems," Mathematical Programming, pp. 49–68, 1992.
15. S.K. Eldersveld, "Large-scale sequential quadratic programming algorithms," Stanford, Report SOL 92-4, Department of Operations Research, Stanford University, 1992.
16. M.C. Fenelon, "Preconditioned conjugate-gradient-type methods for large-scale unconstrained optimization," Ph.D. Thesis, Department of Operations Research, Stanford University, 1981.
17. R. Fletcher, "An optimal positive definite update for sparse Hessian matrices," SIAM J. on Optimization, vol. 5, pp. 192–218, 1995.
18. J.Ch. Gilbert and C. Lemaréchal, "Some numerical experiments with variable-storage quasi-Newton algorithms," Mathematical Programming, pp. 407–435, 1989.
19. P.E. Gill and W. Murray, "Conjugate-gradient methods for large-scale nonlinear optimization," CA, Report SOL 79-15, Department of Operations Research, Stanford University, Palo Alto, 1979.
20. P.E. Gill, W. Murray, and M.A. Saunders, "Large-scale SQP methods and their application in trajectory optimization," Control Applications of Optimization, International Series of Numerical Mathematics, R. Bulirsch and D. Kraft (Eds.), Birkhäuser Basel, vol. 115, pp. 29–42, 1994.
21. P.E. Gill, W. Murray, and M.A. Saunders, "An SQP algorithm of large-scale optimization," Report SOL 95-x, Department of Operations Research, Stanford University (to appear).
22. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "User's guide for NPSOL (version 4.0): A fortran package for nonlinear programming," Report SOL 86-2, Department of Operations Research, Stanford University, 1986.
23. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "Inertia-controlling methods for quadratic programming," SIAM Review, vol. 33, pp. 1–33, 1988.
24. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "Constrained nonlinear programming," in Optimization, Handbooks in Operations Research and Management Science, G.L. Nemhauser and A.H.G. Rinnooy Kan (Eds.), Elsevier, vol. 1, Ch. III, pp. 171–210, 1989.

25. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "A Schur-complement method for sparse quadratic programming," in Reliable Numerical Computation, M.G. Cox and S. Hammarling (Eds.), Oxford University Press, pp. 113–138, 1990.
26. P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, "Some theoretical properties of an augmented Lagrangian merit function," in Advances in Optimization and Parallel Computing, P.M. Pardalos (Ed.), North-Holland, pp. 101–128, 1992.
27. S.P. Han, "Superlinearly convergent variable metrix algorithms for general nonlinear programming problems," Math. Prog., vol. 11, pp. 263–282, 1976.
28. M.W. Leonard, "Improved quasi-Newton methods for optimization," Ph.D. Thesis, Department of Mathematics, University of California, San Diego, 1995.
29. G. McCormick, "A modification of Armijo's step-size rule for negative curvature," Mathematical Programming, vol. 13, pp. 111–115, 1977.
30. J.J. Moré and D.C. Sorensen, "Newton's method," in Studies in Numerical Analysis, G.H. Golub (Ed.) (Mathematical Association of America), pp. 29–82, 1984.
31. W. Murray, "An algorithm for constrained minimization," in Optimization, R. Fletcher (Ed.), Academic Press: London and New York, pp. 247–258, 1969.
32. W. Murray and F.J. Prieto, "A sequential quadratic programming algorithm using an incomplete solution of the subproblem," in SIAM J. on Optimization, vol. 5, pp. 589–639, 1995.
33. W. Murray and F.J. Prieto, "A second-derivative method for nonlinearly constrained optimization," Technical Report Report SOL 95-3, Department of Operations Research, Stanford University, Stanford, 1995.
34. B.A. Murtagh and M.A. Saunders, "A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints," Mathematical Programming Study, vol. 16, pp. 84–117, 1982.
35. B.A. Murtagh and M.A. Saunders, MINOS 5.4 user's guide, Report SOL 83-20R, Department of Operations Research, Stanford University, 1993.
36. J. Nocedal, "Updating quasi-Newton matrices with limited storage," Mathematics of Computation, vol. 35, pp. 773–782, 1980.
37. D.B. Ponceleón, "Barrier methods for large-scale quadratic programming," Report SOL 91-2, Stanford University, Stanford, 1991.
38. M.J.D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in Numerical Analysis, Dundee 1977, Lecture Notes in Mathematics 630, G.A. Watson (Ed.), Springer-Verlag, pp. 144–157, 1978.
39. U.T. Ringertz, "A mathematical programming approach to structural optimization," Report No. 88-24, Dept. of Aeronautical Structures and Materials, The Royal Institute of Technology, Stockholm, 1988.

Computational Optimization and Applications 7, 143–158 (1997)

A Scalable Parallel Interior Point Algorithm for Stochastic Linear Programming and Robust Optimization*

DAFENG YANG
Operations and Information Management, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104

STAVROS A. ZENIOS
Department of Public and Business Administration, University of Cyprus, Kallipoleos 75, Nicosia, Cyprus

Abstract. We present a computationally efficient implementation of an interior point algorithm for solving large-scale problems arising in stochastic linear programming and robust optimization. A matrix factorization procedure is employed that exploits the structure of the constraint matrix, and it is implemented on parallel computers. The implementation is perfectly scalable. Extensive computational results are reported for a library of standard test problems from stochastic linear programming, and also for robust optimization formulations. The results show that the codes are efficient and stable for problems with thousands of scenarios. Test problems with 130 thousand scenarios, and a deterministic equivalent linear programming formulation with 2.6 million constraints and 18.2 million variables, are solved successfully.

Keywords: planning under uncertainty, parallel computing, optimization, software

1. Introduction

Stochastic linear programming (SLP), see e.g., Wets [20], and robust optimization (RO), see Mulvey et al. [12], model problems with uncertain input data. References to the wide range of applications are made by Wets and Mulvey et al., and in the textbook of Kall and Wallace [8]. The mathematical programming formulations arising in SLP and RO are usually of extremely large size, as they model constraints for a large number of realizations (called *scenarios*) of the uncertain data. However, these programs are extremely sparse and structured. Considerable research efforts have gone into the development of efficient algorithms for solving these problems. With the advances in parallel computer architectures research has focused on the design of *decomposition* algorithms, and their implementation on parallel machines, Dantzig [5]. A non-exhaustive list of recent works in this direction includes (1) the parallelization of Benders decomposition due to Dantzig et al. [6] and Nielsen and Zenios [16], (2) the development of parallel decomposition algorithms based on diagonal quadratic approximations of augmented Lagrangian due to Mulvey and Ruszczyński [11] and Berger et al. [1], and (3) the

*Research funded in part by contract HPC-Finance of DGIII of the European Union.

development of parallel decomposition algorithms for stochastic programs with network structures [13, 15].

The above references report encouraging computational results with large-scale problems. However, the parallel implementations do not scale well since decomposition algorithms contain a coordination phase which becomes critical for very large problems. It is also, usually, the case that some amount of fine-tuning is required in order to ensure convergence of the algorithm to a neighborhood of a solution, and this prohibits the use of the codes in industrial settings by non-experts.

An alternative school of thought for solving these problems, that emerged more recently, seeks solution procedures using interior point algorithms. In particular, it has been observed that the number of iterations required by an interior point algorithm to solve SLP and RO is a low order polynomial in the number of scenarios. Hence, if we could develop efficient and stable, parallel, procedures for computing the steps of an interior point algorithm we will have an effective methodology for solving, routinely, these problems. The foundations for this strand of research were laid in the paper of Birge and Qi [2] who developed a matrix factorization procedure to compute the projections of an interior point algorithm for SLP. Birge and Holmes [3] tested alternative implementations of an interior point algorithm for SLP, and concluded that an implementation using the Birge-Qi procedure was stable and efficient. Jessup et al. [7] showed that the Birge-Qi matrix factorization procedure could be implemented on parallel machines in a scalable fashion. Their results, obtained on an Intel iPSC/860 (hypercube communication architecture) and a Connection Machine CM-5 (fat tree communication architecture), exhibit perfect scalability.

Some gaps remain in the above literature, and our goal in this paper is to fill these gaps. In doing so we develop an efficient and scalable parallel code for SLP and RO. We address three questions. First, does the perfect scalability of the Birge-Qi matrix factorization procedure translate to scalability of an interior point algorithm, based on this procedure, for solving SLP? Second, is the parallel implementation of the Birge-Qi procedure numerically stable when applied to systems arising in an interior point algorithm? Third, do the encouraging results with SLP carry over to the quadratic programming formulations arising in RO? As the rest of the paper demonstrates the answers are, in all cases, affirmative. Section 2 formulates the problem. Section 3 describes the algorithm and the parallel procedures of the code we developed, called $ROBO_pT$ for ROBust OpTimizer. Section 4 reports the results of the computational experiments using both high-performance workstations and parallel machines.

2. Problem formulation

Two-stage stochastic linear programs determine an optimal first-stage decision vector $x_0 \in \mathbb{R}^{n_0}$, with cost vector $c_0 \in \mathbb{R}^{n_0}$, before some random coefficients are observed, and then it takes an optimal *recourse* action after the random coefficients become known. We assume $l = 1, 2, \ldots, N$ scenarios of the random coefficients: $c_l \in \mathbb{R}^{n_l}$, $T_l \in \mathbb{R}^{m_l \times n_0}$, $W_l \in \mathbb{R}^{m_l \times n_l}$, $b_l \in \mathbb{R}^{m_l}$. (For simplicity we assume that the probability of scenario l is incorporated in the cost coefficients c_l). The recourse (second-stage) decision under scenario l is denoted by $y_l \in \mathbb{R}^{n_l}$. With this notation the problem is written as:

$$\begin{aligned}
&\text{minimize } c_0^\top x_0 + \textstyle\sum_{l=1}^{N} c_l^\top y_l \\
&\text{subject to:} \\
&\qquad A_0 x_0 \qquad\qquad = b_0, \\
&\qquad T_l x_0 + W_l y_l \quad = b_l, \quad \text{for } l = 1, 2, \ldots, N, \\
&\qquad x_0 \geq 0, \; y_l \geq 0, \quad \text{for } l = 1, 2, \ldots, N.
\end{aligned}$$

A_0 is an $m_0 \times n_0$ constraints matrix for the first-stage variables, and $b_0 \in \mathbb{R}^{m_0}$ is the vector of right-hand side coefficients for these constraints. This problem has $n = n_0 + \sum_{l=1}^{N} n_l$ variables and $m = m_0 + \sum_{l=1}^{N} m_l$ equality constraints. A_0 and W_l are assumed to have full row rank, with $m_l \leq n_l$ for all $l = 0, 1, 2, \ldots, N$, and $n_0 \leq \sum_{l=1}^{N} n_l$. (Full row rank of A_0 is a reasonable assumption, but for many real-world problems the assumption of full row-rank of W_l may not hold true, at least for some l. It is reasonable to assume that the concatenated matrices $(T_l \mid W_l)$ have full row rank, but once T_l is removed the remaining matrix (W_l) may not be of full row rank.)

The objective function of SLP minimizes the cost of the first-stage decision, $c_0^\top x_0$, plus the expected cost of the second-stage decisions, $\sum_{l=1}^{N} c_l^\top y_l$. RO minimizes higher-order moments of the objective function value, and this is one of its key distinguishing features over SLP. For example, a variance term may be minimized. Alternatively, for maximization problem, RO may use an expected utility maximization formulation [12].

Denoting by $\mathbf{c}$ the concatenated vector $\mathbf{c}^\top = (c_0^\top \mid c_1^\top \mid \cdots \mid c_N^\top)$, and, similarly, $\mathbf{x}^\top = (x_0^\top \mid y_1^\top \mid \cdots \mid y_N^\top)$, and letting Q be a positive-definite matrix (e.g., a variance matrix) we can formulate the RO problem as:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \mathbf{c}^\top \mathbf{x} + \mathbf{x}^\top Q \mathbf{x} \tag{1}$$

$$\text{s.t.} \quad A\mathbf{x} = \mathbf{b}, \tag{2}$$

$$\mathbf{x} \geq 0. \tag{3}$$

where $\mathbf{b}^\top = (b_0^\top \mid b_1^\top \cdots \mid b_N^\top)$, and A is the constraint matrix:

$$A = \begin{pmatrix} A_0 & & & \\ T_1 & W_1 & & \\ \vdots & & \ddots & \\ T_N & & & W_N \end{pmatrix}.$$

Stochastic linear programming formulations are cast in the formulation of (1)–(3) simply by ignoring the quadratic term. (More general formulations of RO are given in [12].)

3. The robust optimization code *ROBO$_p$T*

We develop a code for solving problem (1)–(3) based on the primal-dual path-following algorithm of Monteiro and Adler [10] and Vanderbei and Carpenter [19]. It can be described as follows:

Algorithm 3.1 (The primal-dual path following algorithm for quadratic programs).
Initialization: *Start with a triplet* $(\mathbf{x}^0, y^0, z^0)$ *satisfying* $\mathbf{x}^0 > 0$, $z^0 > 0$, *and any* $\mu^0 > 0$. $\mathbf{x}$ *are the primal variables in* (1)–(3), $y \in \mathbb{R}^m$ *are the dual variables for constraints* (2), *and* $z \in \mathbb{R}^n$ *are the reduced cost variables for the bound constraints* (3). *Initialize the iteration index* $\nu \leftarrow 0$.
Iterative Step: *Calculate the dual step* Δy *by solving*:

$$(A\Theta A^\top)\Delta y = \psi, \tag{4}$$

where $\Theta = (Q + ZX^{-1})^{-1}$, $\psi = \rho + A\Theta(\sigma - X^{-1}\phi)$, $Z = \operatorname{diag}\{z^\nu\}$ *and* $X = \operatorname{diag}\{\mathbf{x}^\nu\}$. *The constants* ρ, σ, *and* ϕ *are defined by*:

$$\rho \doteq b - Ax,$$
$$\sigma \doteq c + Qx - A^\top y - z,$$
$$\phi \doteq \mu\mathbf{1} - XZ\mathbf{1},$$

where $\mathbf{1} \in \mathbb{R}^n$ *is the vector of all one's. Compute the primal step* Δx *from*

$$\Delta x = -\Theta(\sigma - X^{-1}\phi - A^\top \Delta y), \tag{5}$$

and the slack variable step Δz *from*

$$\Delta z = X^{-1}(\phi - Z\Delta x). \tag{6}$$

Update:

$$x^{\nu+1} = x^\nu + \alpha\Delta x, \tag{7}$$
$$y^{\nu+1} = y^\nu + \alpha\Delta y, \tag{8}$$
$$z^{\nu+1} = z^\nu + \alpha\Delta z, \tag{9}$$

where $0 < \alpha \leq 1$ *is the step length, chosen so that the primal and reduced cost variables remain positive. It is computed as* $\alpha \doteq \delta \min\{\hat{\alpha}_P, \hat{\alpha}_D\}$, *where* $0 < \delta < 1$, *and*

$$\hat{\alpha}_P \doteq \max_{\alpha_j > 0}\{\alpha_j \mid x_j + \alpha_j \Delta x_j \geq 0\},$$
$$\hat{\alpha}_D \doteq \max_{\alpha_j > 0}\{\alpha_j \mid z_j + \alpha_j \Delta z_j \geq 0\}.$$

A typical value for δ *is* 0.9995.
Reduce μ^ν *to* $\mu^{\nu+1}$, *update* $\nu \leftarrow \nu + 1$, *and repeat the iterative step.*
The algorithm terminates when the gap between the primal and the dual objective function values is smaller than some acceptable tolerance. (In our numerical experiments the algorithm terminates when the primal and dual objective functions agree to, at least, 8 *decimal points.)*

The computationally expensive part of this algorithm is the solution of the system of eqn. (4) to calculate the dual step Δy. In the next section we describe the numerical procedures used for solving this system.

3.1. *Solving for the dual step* Δy

The procedure for solving eqn. (4) for the dual step Δy of the quadratic programming RO model is a straightforward extension of the procedure developed by Birge and Qi for the solution of SLP. (We assume that Q is a diagonal matrix. This assumption simplifies the presentation of the procedure and allows us to exploit the problem's structure. RO formulations with non-diagonal matrices can be reformulated by computing the factorization of $Q = L^\top L$, defining $\bar{\mathbf{x}} = L\mathbf{x}$ and rewriting $\mathbf{x}^\top Q\mathbf{x} = \mathbf{x}^\top L^\top L\mathbf{x} = \bar{\mathbf{x}}^\top \bar{\mathbf{x}}$.) It is based on the following result:

Theorem 3.1. *Let* $M \doteq A\Theta A^\top$, *where* Θ *is diagonal, and* $S \doteq \operatorname{diag}\{S_0, S_1, \ldots, S_N\}$ *where* $S_l = W_l \Theta_l W_l^\top \in \mathbb{R}^{m_l \times m_l}$, $l = 1, \ldots, N$, $S_0 = I$ *is an* $m_0 \times m_0$ *identity matrix, and* $\Theta_l \in \mathbb{R}^{n_l \times n_l}$ *is the (diagonal) submatrix of* Θ *corresponding to the* l*th block. Also, let*

$$G_1 \doteq \Theta_0^{-1} + A_0^\top A_0 + \sum_{l=1}^{N} T_l^\top S_l^{-1} T_l, \tag{10}$$

$$G \doteq \begin{bmatrix} G_1 & A_0^\top \\ -A_0 & 0 \end{bmatrix}, \quad U \doteq \begin{pmatrix} A_0 & I \\ T_1 & 0 \\ \vdots & \vdots \\ T_N & 0 \end{pmatrix}, \quad V \doteq \begin{pmatrix} A_0 & -I \\ T_1 & 0 \\ \vdots & \vdots \\ T_N & 0 \end{pmatrix}.$$

If A_0 *and* W_l, $l = 1, \ldots N$, *have full row rank then* M *and* $G_2 \doteq -A_0 G_1^{-1} A_0^\top$ *are invertible, and*

$$M^{-1} = S^{-1} - S^{-1} U G^{-1} V^\top S^{-1}. \tag{11}$$

Proof: Follows along the same lines as the proof of Birge and Qi for linear programs, under the assumption that Q is invertible. □

It is easy to verify, using eqn. (11), that the solution of the linear system $(A\Theta A^\top)\Delta y = \psi$ is given by $\Delta y = p - r$, where p solves $Sp = \psi$, and r is obtained from the system

$$Gq = V^\top p, \quad \text{and} \quad Sr = Uq. \tag{12}$$

The vector p can be computed component-wise by solving $S_l p_l = \psi_l$, for $l = 1, \ldots, N$. In order to solve for q we exploit the block structure of G and write:

$$Gq = \begin{bmatrix} G_1 & A_0^\top \\ -A_0 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \end{bmatrix}, \quad \text{where} \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \end{bmatrix} \equiv V^\top p. \tag{13}$$

Hence, we get

$$q_2 = -G_2^{-1}\left(\hat{p}_2 + A_0 G_1^{-1}\hat{p}_1\right), \tag{14}$$

$$q_1 = G_1^{-1}\left(\hat{p}_1 - A_0^{\top} q_2\right). \tag{15}$$

Once q is known, r can be computed component-wise by solving $S_l r_l = T_l q_1$. The procedure for calculating Δy is summarized below. (We use $(A)_{i\cdot}$ and $(A)_{\cdot i}$ to denote the ith row and column of a matrix A respectively.)

Procedure 3.1 (Matrix factorization for dual step calculation).

Step 1: *Solve* $Sp = \psi$.

Step 2: (*Solve* $Gq = V^{\top} p$).

a. *Solve* $S_l(u_l)_i = (T_l)_{\cdot i}$, *for* $(u_l)_i$, $i = 1, \ldots, n_0$, *thus computing the columns of the matrix* $S_l^{-1} T_l$, *for all* $l = 1, \ldots, N$.

b. *Multiply* $T_l^{\top}(u_l)_i$, *for* $i = 1, \ldots, n_0$, *to form* $T_l^{\top} S_l^{-1} T_l$, *for all* $l = 1, \ldots, N$. *Form* G_1 (*cf. eqn.* (10)). *Compute* $\hat{p}_1$, $\hat{p}_2$.

c. *Solve* $G_1 u = \hat{p}_1$ *for* u, *and set* $v = \hat{p}_2 + A_0 u$, (*cf. eqn.* (14)).

d. *Form* G_2 *by solving* $(G_1)w_i = (A_0^{\top})_{\cdot i}$ *for* w_i, *for* $i = 1, \ldots, m_0$, *and setting* $G_2 = -A_0[w_1 w_2, \ldots, w_{m_0}]$.

e. *Solve* $G_2 q_2 = -v$ *for* q_2, *and solve* $G_1 q_1 = \hat{p}_1 - A_0^{\top} q_2$ *for* q_1 (*cf. eqn.* (15)).

Step 3: (*Solve* $Sr = Uq$). *Set* $r_0 = A_0 q_1 + q_2$, *and, for* $l = 1, \ldots, N$, *solve* $S_l r_l = T_l q_1$, *for* r_l.

Step 4: (*Form* Δy). *Set* $\Delta y = p - r$.

3.2. *Parallel implementation of ROBO$_p$T*

The implementation of the interior code in *ROBO$_p$T* uses the parallel implementation of Procedure 3.1, developed by Jessup et al. [7], to compute the dual step. Computations of the various constants in Algorithm 3.1, and the calculation of Δx, are also done in parallel as explained below.

The parallel solution begins with the following data distribution. Processor l holds the data corresponding to the lth scenario for second stage decisions T_l, W_l, Θ_l, ρ_l and σ_l. Each processor also holds a copy of the data for the first stage parameters A_0, Θ_0, ρ_0, σ_0. With this data distribution calculations that involve the scenario matrices and variables are computed by multiple processors in parallel, with the lth processor performing the calculations for the lth scenario. (We assume for simplicity that there are as many processors as there are scenarios.) Calculations that involve the first stage variables and matrices are computed, redundantly, on multiple processors. By doing so each processor has available, locally, the information on the first-stage decisions and the need to communicate this information from some "master" processor is avoided. We describe now the parallel implementation of all steps of the algorithm. First, the right-hand side of eqn. (4) is computed by the following procedure, called Formrhs.

Procedure 3.2 (Parallel formrhs).

We start with the vector $\hat{\sigma} = \sigma - X^{-1}\phi$. *This vector is easily computed, componentwise, for every* $\hat{\sigma}_l$, $l = 0, 1, \ldots, N$, *using parallel vector multiplications to postmultiply the diagonal matrix* X^{-1} *by* ϕ.

1. (*Form* $A\Theta\hat{\sigma}$). *On all processors form* $A_0\Theta_0\hat{\sigma}_0$. *On processor* $l = 1, \ldots, N$, *form* $T_l\Theta_0\hat{\sigma}_0 + W_l\Theta_l\hat{\sigma}_l$.
2. (*Form* $\rho + A\Theta\hat{\sigma}$). *On all processors form* $\rho_0 + A_0\Theta_0\hat{\sigma}_0$. *On processor* $l = 1, \ldots, N$, *form* $\rho_l + T_l\Theta_0\hat{\sigma}_0 + W_l\Theta_l\hat{\sigma}_l$.

Once the right-hand-side of (4) has been computed we can solve the system using the parallel matrix factorization procedure developed by Jessup et al. [7]. This procedure, called Finddy, uses a *global reduction* function that takes as input vectors (or matrices) distributed in every processor, sums them, and leaves a single vector (or matrix) sum at every node. (For optimal implementations of this function on hypercube networks and fat trees, and additional references, see [7]). It can be described as follows:

Procedure 3.3 (Parallel finddy).

1. (*In parallel, solve* $Sp = b$). *On all processors solve* $S_0p_0 = b_0$. *On processor* $l = 1, \ldots, N$, *solve* $S_lp_l = b_l$.
2. (*Solve* $Gq = V^\top p$).
   (a) *On processor* $l = 1, \ldots, N$, *solve* $S_l(u_l)_i = (T_l)_{\cdot i}$, $i = 1, \ldots, n_0$.
   (b) *On processor* $l = 1, \ldots, N$, *multiply* $T_l^\top(u_l)_i$. *Call global reduction to form* $(G_1)_i = (\Theta_0)_{ii} + \sum_{l=1}^N T_l^\top(u_l)_i + (A_0^\top A_0)_i$. *Use global reduction to form* $\hat{p}_1$ *and* $\hat{p}_2$.
   (c) *On all processors solve* $G_1u = \hat{p}_1$ *for* u *and set* $v = \hat{p}_2 + A_0u$.
   (d) *On all processors form* G_2 *by solving* $(G_1)w_i = (A_0^\top)_{\cdot i}$ *for* w_i, *for* $i = 1, \ldots, m_0$ *and setting* $G_2 = -A_0[w_1w_2, \ldots, w_{m_0}]$.
   (e) *On all processors solve* $G_2q_2 = -v$ *for* q_2, *and solve* $G_1q_1 = \hat{p}_1 - A_0^\top q_2$ *for* q_1.
3. (*In parallel solve* $Sr = Uq$). *On all processors set* $r_0 = A_0q_1 + q_2$. *On processor* $l = 1, \ldots, N$ *solve* $S_lr_l = T_lq_l$ *for* r_l.
4. (*In parallel form* Δy). *On all processors set* $\Delta y_0 = p_0 - r_0$. *On processor* $l = 1, \ldots, N$ *set* $\Delta y_l = p_l - r_l$ *for* $l = 1, \ldots, N$.

Finally the parallel procedure Finddx below constructs the primal step direction vector Δx defined by Eq. (5).

Procedure 3.4 (Parallel finddx).

1. *On all processors form* $A_0^\top\Delta y_0$. *On processor* $l = 1, \ldots, N$, *form* $T_l^\top\Delta y_l$. *Call global reduction to form* $\Delta\hat{x}_0 = A_0^\top\Delta y_0 + \sum_{l=1}^N T_l^\top\Delta y_l$.
2. *On processor* $l = 1, \ldots, N$, *form* $\Delta\hat{x}_l = W_l^\top\Delta y_l$.
3. *On all processors form* $\Delta x_0 = -\Theta_0(\hat{\sigma}_0 - \Delta\hat{x}_0)$. *On processor* $l = 1, \ldots, N$, *form* $\Delta x_l = -\Theta_l(\hat{\sigma}_l - \Delta\hat{x}_l)$.

The step on the reduced cost variables, Δz, is finally trivially computed using parallel vector multiplications, since all matrices involved are diagonal, and the block corresponding

to the lth subvector of z resides at the lth processor. The lth processors can now take a step in the x_l, y_l, z_l variables, as well as the x_0, y_0, z_0 variables, since all required quantities are available locally.

4. Computational results

We now report results from the computational experiments carried out with $ROBO_pT$ with a suite of large-scale test problems. The objective of our experimental design is to address the questions raised at the introduction of this paper. Namely, to establish that the developed matrix factorization procedures are stable when used to solve large scale SLP and RO, that the scalability of these procedures, when implemented on parallel machines, translates to scalability of an interior point algorithm, and that very large scale problems can be solved efficiently with $ROBO_pT$. Comparisons with a state-of-the-art code, LOQO of Vanderbei [18], illustrate that $ROBO_pT$ is competitive even for small to medium size problems, on serial computers.

The results with the parallel code were obtained on a Connection Machine CM-5e [9]. Serial computing experiments were carried out on an IBM RS6000/550 workstation. Both codes, LOQO and $ROBO_pT$ are written in C. $ROBO_pT$ uses the sparse, supernodal Cholesky factorization and solver routines SUPFCT and SUPSLV of Ng and Peyton [14] to solve the scenario systems, and LAPACK for the dense matrix systems of the first-stage problem. The communications in the parallel code are implemented via the usage of standard routines from the CMSSL library of the Connection Machine. The code is compiled with the `gcc` compiler with `-O3` flag for maximum optimization.

4.1. Test problems

We solved the five sets of problems—sc205, scagr7, scfxm1, scrs8 and scsd8—from the library of SLP test problems described by Birge and Holmes [3]. We also experimented with the SLP formulation of a telecommunications problems, sen, described in Sen et al. [17]. For each set we generated several problems, with increasing number of scenarios. Their characteristics are described in Table 1. For two of the test sets, scsd8 and sen, we generated problems with thousands of scenarios, as described in Table 2. To the best of our knowledge these are the largest SLP problems solved to date.

RO models were generated by adding a randomly generated diagonal quadratic matrix to the objective function of the SLP test problems. The condition number of this matrix is user specified, and is reported with the computational results below. The algorithm would terminate when the primal and dual objective values would agree in, at least, 8 decimal points.

4.2. Serial computations: comparing $ROBO_pT$ with LOQO

The Birge-Qi matrix factorization procedure is not the most efficient way for computing the interior point steps for small-scale problems. In order to establish the penalty paid by

Table 1. Characteristics of the stochastic linear programming test problems.

Problem	Scenarios	Constraints	Variables
sc205.4	4	101	102
sc205.8	8	189	190
sc205.16	16	365	366
sc205.32	32	717	718
sc205.64	64	1,421	1,422
scagr7.4	4	167	180
scagr7.8	8	319	340
scagr7.16	16	623	660
scagr7.32	32	1,231	1,300
scagr7.64	64	2,447	2,580
scfxm1.4	4	684	1,014
scfxm1.8	8	1,276	1,914
scfxm1.16	16	2,460	3,714
scfxm1.32	32	4,828	7,314
scfxm1.64	64	9,564	14,514
scrs8.4	4	140	189
scrs8.8	8	252	341
scrs8.16	16	476	645
scrs8.32	32	924	1,253
scrs8.64	64	1,820	2,460
scsd8.4	4	90	630
scsd8.8	8	170	1,190
scsd8.16	16	330	2,310
scsd8.32	32	650	4,550
scsd8.64	64	1,290	9,030
sen.4	4	701	2,913
sen.8	8	1,401	5,737
sen.16	16	2,801	11,385
sen.32	32	5,601	22,681
sen.64	64	11,201	45,273

$ROBO_pT$ *viz a viz* a state-of-the-art serial code, LOQO, we solved several instances of scsd8 and sen with increasing number of scenarios. Results are summarized in figures 1. For problems such as sen, which has a single first-stage constraint, $ROBO_pT$ is faster than LOQO even for problems with very few scenarios. For problems where the first-stage constraint matrix is large, compared to the second-stage matrices, $ROBO_pT$ does not gain an advantage unless we solve problems with large number of scenarios.

Table 2. Characteristics of very large-scale test problems.

Problem	Scenarios	Constraints	Variables
scsd8.128	128	2,570	17,990
scsd8.256	256	5,130	35,910
scsd8.512	512	10,250	71,750
scsd8.1024	1,024	20,490	143,360
scsd8.2048	2,048	40,970	286,790
scsd8.130172	130,172	2,603,440	18,224,080
sen.128	128	22,401	90,457
sen.256	256	44,801	180,825
sen.512	512	89,601	361,561
sen.16384	16,384	2,867,201	13,025,369

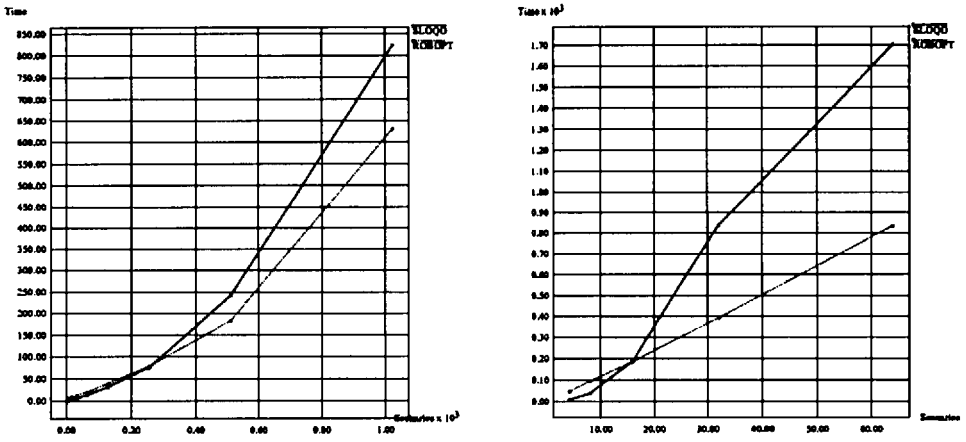

Figure 1. Comparing $ROBO_pT$ with LOQO for the scsd8 (figure on left) and sen (figure on right) test problems with increasing number of scenarios.

4.3. *Parallel computations: relative speedup and scalability*

To establish the suitability of the code for parallel computations we solve the scsd8 and sen test problems on a Connection Machine CM-5, using up to 64 processors. Figure 2 shows the relative speedup (i.e., ratio of solution time with the serial implementation of $ROBO_pT$ to the solution time with its parallel implementation) as the number of scenarios increases and for different number of processors.

Superlinear speedup is achieved with the scsd8 problem. That is, the parallel code on p processors solves the problem more than p times faster than the same code implemented serially. This is due to the effect of cache memory, which can hold more than one block (i.e., W_l, T_l matrices) of the scsd8 problem. When only a single processor is available not all blocks will fit in cache memory, and the solution time is affected by the transfer of

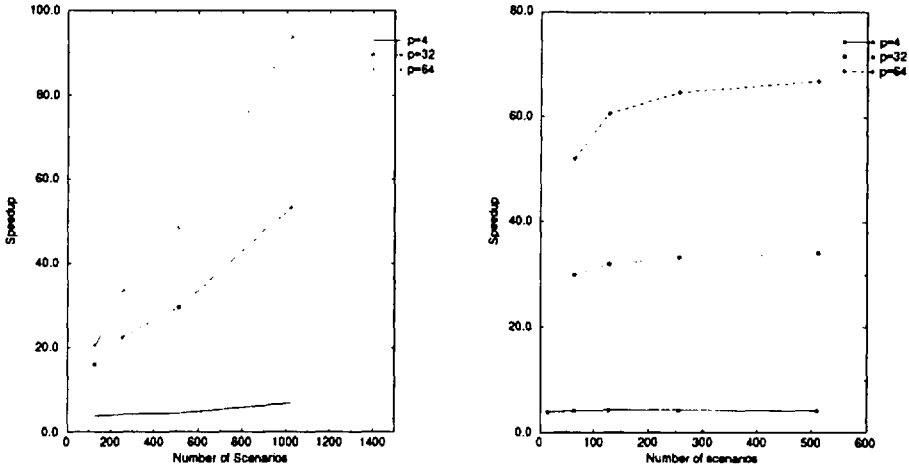

Figure 2. Relative speedup of parallel $ROBO_pT$, implemented on the Connection Machine CM-5 with $p = 4$, 32 and 64 processors, for the solution of the scsd8 (figure on left) and sen (figure on right) test problems.

data in and out of the cache. In the parallel implementation each processor holds in cache memory all the blocks operated upon by that processor, and the overhead of caching is avoided. No caching effect is observed for the sen test problem, since each block of this problem is large and even a single block cannot fit in cache memory. Hence, data need to be transferred into and out of cache memory in both the serial and the parallel implementation. The speedup achieved for the sen problem is solely due to the efficient exploitation of the multiple processors by the parallel procedures Formrhs, Finddy and Finddx. The efficiency of parallel $ROBO_pT$ for the solution of sen is 98–99%.

Similar efficiency has been observed for all test problems. The results obtained with the parallel solution of RO problems are identical to those reported here for the solution of SLPs.

We also conducted experiments to establish the *scalability* of the parallel code. Scalability is the ability of a parallel code to maintain a constant level of efficiency as the number of processors increases, by solving problems whose sizes increase in proportion to the number of processors. See, e.g., Censor and Zenios [4, Chapter 1]. This measure is important in establishing whether a massively parallel machine can be used to solve extremely large problems, or if the benefits from parallelism are restricted to machines with few processors. We solved problems scsd8 and sen, with 4, 32 and 64 scenarios, using an equal number of processors. Results are summarized in Table 3. We observe that the solution time, per interior point iteration, remains virtually constant as we increase the number of processors to match the number of scenarios. Parallel $ROBO_pT$ is perfectly scalable.

4.4. *Benchmark results with parallel* $ROBO_pT$

To benchmark $ROBO_pT$, and establish the joint effects of the matrix factorization procedures and their parallel implementation, we solved the suite of SLP test problems on the Connection Machine CM-5. For the smaller test problems we use as many processors as

Table 3. Testing the scalability of the parallel implementation of $ROBO_pT$. The solution time, per iteration, remains virtually constant when the number of processors increases in proportion to the number of scenarios. All times in CM seconds.

Problem	Processors	Iterations	Solution time	Time per itn.
scsd8.4	4	9	1.19	.132
scsd8.32	32	9	1.22	.135
scsd8.64	64	9	1.22	.135
sen.4	4	18	13.6	.756
sen.32	32	19	14.5	.763
sen.64	64	21	16.1	.767

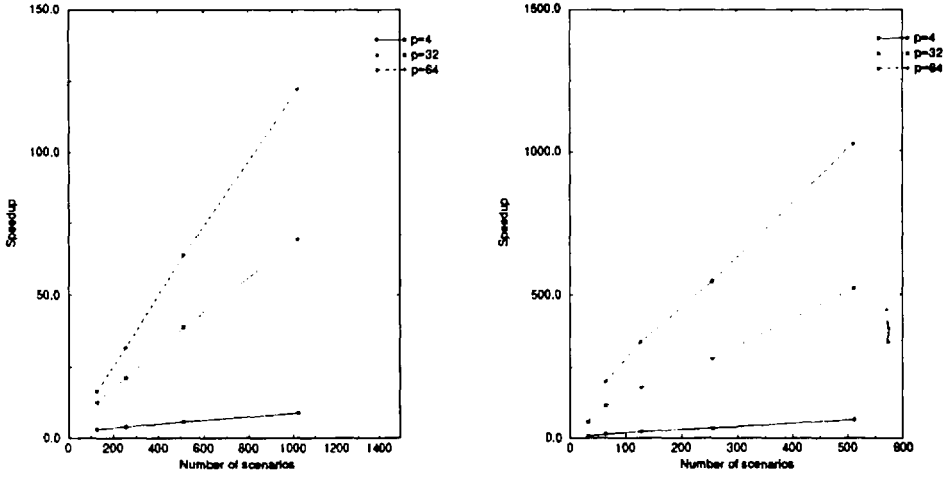

Figure 3. Speedup of parallel $ROBO_pT$ compared to LOQO for the solution of the scsd8 and sen test problems.

the number of scenarios. Results are summarized in Table 4, where the parallel implementation of $ROBO_pT$ is compared with LOQO executing on a single processor of the CM-5. $ROBO_pT$ outperforms LOQO, as the number of scenarios becomes large. The exact number of scenarios for which it becomes preferable to use $ROBO_pT$ over LOQO depends on the structure of the blocks of the test problem. Problems with small values of n_0, m_0 compared to n_l, m_l favor $ROBO_pT$. This is the case with the sen test problem. Problems with large values of n_0, m_0 compared to n_l, m_l favor LOQO unless the number of blocks is large.

The relative performance of parallel $ROBO_pT$ over LOQO improves for larger problems. Figure 3 shows the speedup of $ROBO_pT$ over LOQO on machines with up to 64 processors, and for an increasing number of scenarios. The fact that the speedups exceed the number of the available processors provides additional support to the claim of Section 4.2 that, even when implemented serially, $ROBO_pT$ is more efficient than LOQO for large scale problems.

Table 4. Benchmark results with the parallel implementation of $ROBO_pT$ on the Connection Machine CM-5, and comparisons with LOQO. LOQO executes on a single processor of the CM-5; $ROBO_pT$ uses as many processors as there are scenarios. Solution times in seconds. NA: not available at the required level of accuracy due to numerical errors.

	LOQO		PARALLEL $ROBO_pT$	
Problems	Iterations	Time	Iterations	Time
sc205.4	12	0.398	10	1.18
sc205.8	14	0.605	12	1.45
sc205.16	16	1.389	15	2.16
sc205.32	17	1.788	19	2.22
sc205.64	18	3.54	21	2.39
scagr7.4	14	0.59	16	2.33
scagr7.8	15	1.03	15	2.44
scagr7.16	16	1.67	16	3.07
scagr7.32	18	3.61	18	2.66
scagr7.64	22	8.81	19	2.86
scfxm1.4	21	3.79	20	80.21
scfxm1.8	25	8.92	22	106.1
scfxm1.16	30	23.12	28	180.6
scfxm1.32	NA	NA	32	113.0
scfxm1.64	NA	NA	41	165.3
scrs8.4	13	0.57	17	13.10
scrs8.8	14	0.84	17	13.15
scrs8.16	15	1.38	17	13.56
scrs8.32	17	2.70	19	14.50
scrs8.64	18	5.69	19	14.50
scsd8.4	9	0.73	9	5.31
scsd8.8	9	1.45	9	5.54
scsd8.16	9	3.37	9	6.03
scsd8.32	9	5.92	9	5.45
scsd8.64	9	12.8	9	5.42
sen.4	12	7.30	18	13.0
sen.8	14	37.31	19	25.1
sen.16	16	188.1	19	48.5
sen.32	17	837.2	20	14.5
sen.64	19	1702.1	21	16.1

4.5. *Parallel computations: Stability of $ROBO_pT$ for robust optimization*

In the testing of matrix factorization procedures conducted by Birge and Holmes [3] it was demonstrated that the Birge-Qi procedure is more stable and accurate than alternative methods based on problem reformulation or Schur complements. The experimental results summarized in Table 5 illustrate that the procedure remains stable and accurate when

Table 5. Stability and accuracy of parallel $ROBO_pT$ when solving ill-conditioned robust optimization problems. For each test problem we report the number of interior point iterations, and (in parenthesis) the number of decimal points of accuracy of the primal and dual objective values.

Problem	No. of procs.	Condition number of Q matrix			
		1	10	100	1000
scsd8.4	4	18 (8)	20 (10)	18 (10)	19 (11)
scsd8.32	32	21 (9)	25 (10)	23 (11)	24 (11)
scsd8.64	64	21 (8)	25 (10)	25 (11)	26 (13)
scsd8.128	64	22 (9)	27 (10)	27 (11)	27 (13)
scsd8.256	64	24 (11)	28 (12)	29 (13)	30 (13)
scsd8.512	64	24 (8)	25 (9)	31 (12)	32 (13)
scsd8.1024	64	29 (12)	29 (11)	31 (11)	35 (15)
scsd8.2048	64	29 (12)	32 (13)	36 (15)	37 (15)
sen.4	4	25 (11)	29 (13)	28 (13)	23 (13)
sen.32	32	27 (14)	23 (14)	24 (14)	29 (15)
sen.64	64	28 (14)	32 (14)	33 (15)	33 (15)
sen.128	64	30 (14)	33 (15)	34 (15)	35 (15)
sen.256	64	27 (15)	32 (15)	32 (15)	33 (15)
sen.512	64	29 (15)	34 (15)	34 (15)	34 (15)
sen.16384	64	37 (15)	49 (14)	49 (15)	49 (15)

implemented in parallel and for RO problems as well. Stability and accuracy is maintained even for problems with thousands of scenarios, and for ill-conditioned Q matrices.

4.6. Solving very large scale test problems

As a last experiment we use the parallel implementation of $ROBO_pT$ to solve the very large scale problems described in Table 2. Results are summarized in Table 6. We observe that the interior point algorithm is capable of solving problems with millions of variables and constraints, to an accuracy of more than 8 decimal points. Solution times are, for most problems, less than one hour of computer time. The folklore that interior point algorithms take a small number of iterations holds true even with the multi-million variable problems solved here.

5. Conclusions

We have discussed the efficient and stable parallel implementation of a primal-dual path-following algorithm for structured linear and nonlinear programs arising when planning under uncertainty. The developed code, $ROBO_pT$, is competitive with state-of-the-art optimization software, when applied to small scale problems. It has superior performance

Table 6. Solving very large-scale stochastic linear programs using $ROBO_pT$. (Solution times in seconds on a Connection Machine CM-5e with 64 processors.)

Problem	Itns.	Solution time
scsd8.128	10	1.88
scsd8.256	10	2.39
scsd8.512	11	3.79
scsd8.1024	12	6.73
scsd8.2048	14	13.82
scsd8.131072	19	1066.1
sen.64	21	16.1
sen.128	23	30.8
sen.256	31	78.3
sen.512	31	153.5
sen.16384	49	7638.3

for large-scale problems, and it also parallelizes extremely well and is perfectly scalable. Work is under way for the extension of the ideas of this paper to the solution of multi-stage planning problems. Preliminary results are equally encouraging and will be reported elsewhere.

References

1. A.J. Berger, J.M. Mulvey, and A. Ruszczyński, "An extension of the DQA algorithm to convex stochastic programs," SIAM Journal on Optimization, vol. 4, no. 4, pp. 735–753, 1994.
2. J.R. Birge and L. Qi, "Computing block-angular Karmarkar projections with applications to stochastic programming," Management Science, vol. 34, pp. 1472–1479, Dec. 1988.
3. J.R. Birge and D.F. Holmes, "Efficient solution of two-stage stochastic linear programs using interior point methods," Computational Optimization and Applications, vol. 1, pp. 245–276, 1992.
4. Y. Censor and S.A. Zenios, Parallel Optimization: Theory, Algorithms and Applications, Oxford University Press: Oxford, England, 1997 (in print).
5. G.B. Dantzig, "Planning under uncertainty using parallel computing," Annals of Operations Research, vol. 14, pp. 1–16, 1988.
6. G.B. Dantzig, J.K. Ho, and G. Infanger, "Solving stochastic linear programs on a hypercube multicomputer," Technical report sol 91-10, Operations Research Department, Stanford University, Stanford, CA, 1991.
7. E.R. Jessup, D. Yang, and S.A. Zenios, "Parallel factorization of structured matrices arising in stochastic programming, SIAM Journal on Optimization, vol. 4, no. 4, pp. 833–846, 1994.
8. P. Kall and S.W. Wallace, Stochastic Programming, John Wiley & Sons: New York, 1994.
9. C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas, C.R. Feynman, M.N. Ganmukhi, J.V. Hill, W.D. Hillis, B.C. Kuszmaul, M.A. St. Pierre, D.S. Wells, M.C. Wong, S.-W. Yang, and R. Zak, "The network architecture of the Connection Machine CM-5," Manuscript, Thinking Machines Corporation, Cambridge, Massachusetts 02142, 1992.
10. R.D.C. Monteiro and I. Adler, "Interior path-following primal-dual algorithms. Part II: Convex quadratic programming," Mathematical Programming, vol. 44, pp. 43–66, 1989.
11. J.M. Mulvey and A. Ruszczyński, "A new scenario decomposition method for large-scale stochastic optimization," Operations Research, vol. 43, pp. 477–490, 1994.

12. J.M. Mulvey, R.J. Vanderbei, and S.A. Zenios, "Robust optimization of large scale systems," Operations Research, vol. 43, pp. 264–281, 1995.
13. J.M. Mulvey and H. Vladimirou, "Evaluation of a parallel hedging algorithm for stochastic network programming," in Impacts of Recent Computer Advances on Operations Research, R. Sharda, B.L. Golden, E. Wasil, O. Balci, and W. Stewart (Eds.), North-Holland, New York, USA, 1989.
14. E. Ng and B. Peyton, "A supernodal Cholesky factorization algorithm for shared-memory multiprocessors," SIAM Journal on Scientific and Statistical Computing, vol. 14, pp. 761–769, 1993.
15. S.S. Nielsen and S.A. Zenios, "A massively parallel algorithm for nonlinear stochastic network problems," Operations Research, vol. 41, no. 2, pp. 319–337, 1993.
16. S.S. Nielsen and S.A. Zenios, "Scalable parallel Benders decomposition for stochastic linear programming," Technical report, Management Science and Information Systems Department, University of Texas at Austin, Austin, TX, 1994.
17. S. Sen, R.D. Doverspike, and S. Cosares, "Network planning with random demand," Working paper, Systems and Industrial Engineering Department, University of Arizona, Tucson, AZ, 1992.
18. R.J. Vanderbei, "LOQO user's manual," Technical report SOR 92-5, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, 1992.
19. R.J. Vanderbei and T.J. Carpenter, "Symmetric indefinite systems for interior point methods," Mathematical Programming, vol. 58, pp. 1–32, 1993.
20. R.J.-B. Wets, "Stochastic programming," in Handbooks in Operations Research and Management Science, G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd (Eds.), vol. 1, pp. 573–629, North-Holland, Amsterdam, 1989.