bounds. For transhipment problems we could identify sets of nodes that do not share primal variables by performing a coloring of the network graph. Nodes that are not connected with an arc will get the same color, and can be processed in parallel. Furthermore, it is possible to design Jacobi variants of the row-action algorithms. These variants permit the concurrent iteration on multiple rows, even if they have primal variables in common. In general, we found that Jacobi variants require more iterations than Gauss-Seidel implementations based on the graph-coloring decomposition. However, Jacobi algorithms utilize a larger number of processors and solve the problem in less time.

3.1 Nonlinear Transportation problems

The development and implementation of row-action algorithms for nonlinear transportation problems is reported in Zenios and Censor [9]. The algorithm executes at 1.6 GFLOPS on a 64K CM–2 when implemented in C/Paris. A microcoded optimal implementation executes at 3.1 GFLOPS. The following table summarizes some results, and compares the massively parallel algorithm with an *equilibration* algorithm [4] implemented on a vector supercomputer:

Nodes × arcs	IBM 3090–600/VF	CM–2 (32K)
1K × 250K	1 min 9 sec	18 sec
2K × 1M	8 min 3 sec	22 sec
4K × 4M	1 hr 3 min 43 sec	47 sec

3.2 Nonlinear Multicommodity Flows

The development and implementation of row-action algorithms for nonlinear multicommodity problems is reported in Zenios [8]. The algorithm solves one commodity at a time, iterating concurrently on multiple rows. Once all commodity subproblems are solved to some desirable level of accuracy, the algorithm iterates on the coupling constraints. The coupling constraints do not have any primal variables in common, and can be iterated upon simultaneously. The iterations then proceed with the network subproblems and so on. The following results highlight the performance of the algorithm on a quadratic problem:

Multicommodity flow program:	8 commodities
	2048 nodes × 1,048,570 arcs

Equivalent non-linear program:	16,384 equality constraints
	104,521 active GUB constraints
	8,384,560 variables
Solution times:	30 min. on 4K processors
	5 min. on 32K processors

3.3 Nonlinear Stochastic Programs with Network Recourse

The development and implementation of row-action algorithms for nonlinear stochastic programming problems with network recourse is reported in Nielsen and Zenios [5]. The algorithm works on the split-variable formulation of Fig. 1. A Jacobi variant iterates on all the rows of all scenario subproblems simultaneously, before the algorithm iterates on the coupling constraints. A closed form solution can be computed for the coupling constraints. This avoids any coordination bottlenecks in the algorithm. We present sample results:

Scenarios	Equivalent NLP	32K CM–2
512	61,952 × 171,520	46.3 sec.
1024	123,904 × 343,040	86.3 sec.
2048	247,808 × 686,080	163.6 sec.
8192	495,616 ×1,272,160	11 min.

3.4 Solving Linear Network Problems

The massively parallel solution of linear network problems is substantially more challenging than the solution of nonlinear problems: it requires the embeding of a row-action algorithm within PMD. Hence, we can not expect to solve the linear problems with the same efficiency we have solved the nonlinear programs. Nevertheless, in Nielsen and Zenios [6] we have been able to solve to a high-level of accuracy, and quite efficiently, linear stochastic networks. We present sample results:

Scenarios	Equivalent NLP	16K CM-2	32K CM–2
512	54,287 × 154,657	16.5 min	9 min
1024	108,559 × 309,281	33 min	17.3 min
2048	217,103 × 618,529	NA	34.5 min

The most interesting observation from this table is the scalability of tha algorithm: The 2048 scenario problem is solved on the 32K CM–2 in

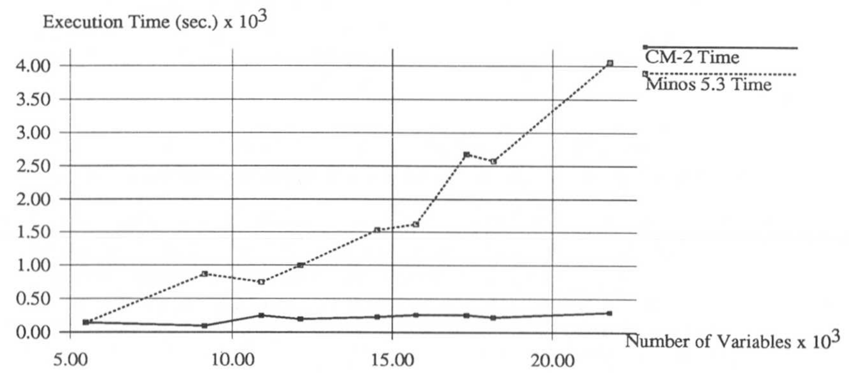

Figure 2: Relative performance of the massively parallel PMD algorithm compared to MINOS 5.3 for increasing problem sizes.

the same time it takes for the solution of the 1024 scenario problem on the 16K CM–2. This property is highlighted in Fig. 2 that compares the relative performance of PMD with MINOS 5.3. (MINOS is executing on a DECstation 5000/200. The results are only indicative of the scalability of the PMD and the CM–2, and should not be taken as a direct comparison of PMD with MINOS.)

4 Conclusions

With suitable reformulations of linear programs and the use of row-action methods we have designed massively parallel algorithms for a wide range of large-scale problems. These ideas have been proven very effective in the solution of multi-million variable problems. Scalability is an important feature of these algorithms. We expect that these methods will continue to do well on "universal" architectures, like the CM–5.

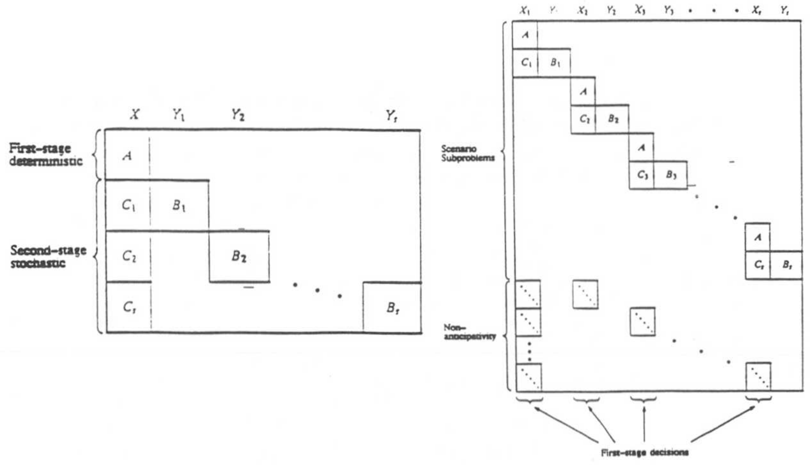

Figure 1: Structure of two-stage stochastic programming model — compact and split-variable formulations. Blocks correspond to network flow constraints.

adjust the dual price for the chosen row, and compute primal variables that satisfy complementarity, and achieve primal feasibility for that row. At the limit primal feasibility is satisfied for all rows. Since complementarity is preserved, the limit point is optimal. These row-action algorithms are applicable to linearly constrained problems, when the objective function is a *Bregman's* function. Hence, these algorithms can be used to solve the sequence of nonlinear problems within PMD (2).

3 Massively parallel decompositions

The judicious application of a row-action algorithm results in parallel decomposition of the optimization problem. Since the algorithm iterates on one row at a time, it is possible to iterate in parallel on multiple rows if those do not have any primal variables in common. Consider the case where the constraint matrix A corresponds to flow conservation constraints of a transportation problem. The row-action algorithm can iterate simultaneously on all origin nodes, then on all destination nodes, and finally on the variable nonlinear network optimization problems was indeed one of the first applications that ran on the Connection Machine CM–1 [7]. We summarize here some of our recent work in this direction. We review the key ideas that led to the design of scalable, massively parallel algorithms. Numerical results are included for three classes of network structured problems: (1) The nonlinear transportation problem, (2) The nonlinear multicommodity network flow problem, and (3) The stochastic programming problem with network recourse. A recent survey of parallel network optimization algorithms is given in Bertsekas et al. [1].

2 Overview of the algorithmic ideas

We consider the linear programming (LP) problem:

$$\text{Minimize}_{x \in X} c^T x \,, \tag{1}$$

where $X = \{x \mid Ax = b,\ 0 \leq x \leq u\}$. A is either a node-arc incidence matrix, or has the block-diagonal structure of the multicommodity network flow problem, or has the dual block-angular structure of stochastic programming problems with network blocks; Fig. 1. We summarize the two key ideas in designing massively parallel algorithms for this problem:

1. Proximal Minimization with D-functions (PMD), Censor and Zenios [3]: We introduce an auxiliary function $f : \Re^n \rightarrow \Re$, and define the D-function by $D_f(x, x^k) := f(x) - f(x^k) - < \nabla f(x^k), x - x^k >$. It can be shown that, if f has certain properties, the following iterative scheme converges to an optimal solution x^* of the linear program (1).

   <u>**The PMD Algorithm**</u>

   $$x^{k+1} \leftarrow \arg \min_{x \in X \bigcap S} c^T x + \frac{1}{\gamma^k} D_f(x, x^k). \tag{2}$$

   $\bar{S}$ is the *zone* of the function f. For convergence of PMD the auxiliary function must be a *Bregman's* function as characterized by Censor and Lent [2].

2. Row-action algorithms, Censor and Lent [2]: The algorithms, as their name implies, iterate on one row of the constraint set at a time. They

References

[1] D.P. Bertsekas, D. Castanon, J. Eckstein, and S.A. Zenios. Parallel computing in network optimization. In *Handbook of Operations Research*, North-Holland, Amsterdam, The Netherlands, 1992.

[2] Y. Censor and A. Lent. An iterative row-action method for interval convex programming. *Journal of Optimization Theory and Applications*, 34:321–353, 1981.

[3] Y. Censor and S.A. Zenios. The proximal minimization algorithm with d-functions. *Journal of Optimization Theory and its Applications*, 1992.

[4] A. Nagurney, D-S. Kim, and A.G. Robinson. Serial and parallel equilibration of large-scale constrained matrix problems with application to the social and economic sciences. *International Journal of Supercomputer Applications*, 4:49–71, 1990.

[5] S. Nielsen and S.A. Zenios. Massively parallel algorithms for nonlinear stochastic network problems. *Operations Research*, 1992. (to appear).

[6] S.S. Nielsen and S.A. Zenios. *Proximal minimizations with D-functions and the massively parallel solution of linear stochastic network programs.* Report 92–01–05, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1992.

[7] S. A. Zenios and R. A. Lasken. Nonlinear network optimization on a massively parallel Connection Machine. *Annals of Operations Research*, 14:147–165, 1988.

[8] S.A. Zenios. On the fine-grain decomposition of multicommodity transportation problems. *SIAM Journal on Optimization*, 1(4):643–669, 1991.

[9] S.A. Zenios and Y. Censor. Massively parallel row-action algorithms for some nonlinear transportation problems. *SIAM Journal on Optimization*, 1(3):373–400, 1991.

Serial and Massively Parallel SOR Algorithms for Large–Scale Engineering Problems*

R. DE LEONE† M. A. TORK ROTH†

Abstract

In this paper we discuss the solution of several engineering problems using serial and parallel successive overrelaxation (SOR) methods. The sparsity structure of the problems considered here allowed us to efficiently implement the SOR algorithm on a Connection Machine CM–2, a massively parallel Single–Instruction–Multiple–Data machine. Computational results are reported for three classes of problems: obstacle problems, elastic–plastic torsion problems, and journal bearing problems. Problems with up to 4 million variables have been solved in a few minutes on a CM–2 with 16,384 processors.

Keywords: Quadratic Programs, Successive Overrelaxation, Massively Parallel Algorithms.

1 Introduction

Our concern here is the quadratic programming problem with simple bounds:

$$\begin{aligned} &\text{minimize} \quad \tfrac{1}{2}x^T Mx + q^T x \\ &\text{subject to} \quad l \leq x \leq u \end{aligned} \tag{1}$$

where M is a symmetric, positive semidefinite nxn real matrix with positive diagonal entries and q, l, u, and x are n–dimensional vectors. Problems of this form arise in many physical and engineering applications such as contact and friction problems in rigid body mechanics, elastic–plastic torsion problems, and journal bearing lubrication [1, 2].

Various types of algorithms have been proposed and studied for the solution of these problems; for example, algorithms based on active set strategies and, more recently, interior point algorithms. An effective active set strategy for very large problems must add or drop many constraints at the same time. The recent algorithm proposed by Moré and Toraldo

*This material is based on research supported by the Air Force Office of Scientific Research Grant AFOSR-89-0410

†Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin Madison, 1210 West Dayton Street, Madison, WI 53706

Progress on the Massively Parallel Solution of Network "Mega"–Problems *

Stavros A. Zenios, Decision Sciences Department,
University of Pennsylvania, Philadelphia, PA 19104.

February 24, 1992

Abstract

The development of massively parallel architectures, like the Connection Machine CM–2, has motivated the design and implementation of algorithms for a variety of optimization problems with network structures. We report here one some recent work in this direction. The key algorithmic ideas that have produced scalable, massively parallel algorithms are reviewed. Illustrative computational results are presented for a variety of network-structured problems with millions of variables.

1 Introduction

It is an undisputed fact that TERAFLOP computing performance — a U.S. national goal in scientific computing — will be achieved by massively parallel computer architectures. Such machines are already available, in the form of the Connection Machine CM–2 and CM–5, Intel's Touchstone, the DECmpp 12000 and so on. Designing efficient algorithms for these architectures has been a challenge for all areas of scientific computing. These machines are, in the most part, SIMD (i.e., single instruction, multiple data). This computing paradigm was viewed as rather restrictive for sparse and ill-structured applications, like linear programming. While we have seen, with the introduction of the CM–5, a merging of the SIMD and MIMD (i.e., multiple instruction, multiple data) architectures, it appears that SIMD programming has the simplicity required for harnessing massive parallelism.

The field of network optimization has benefited substantially from the design and implementation of massively parallel algorithms. The solution of

*Research funded by NSF grant CCR–9104042 and AFOSR grant 91–0168.

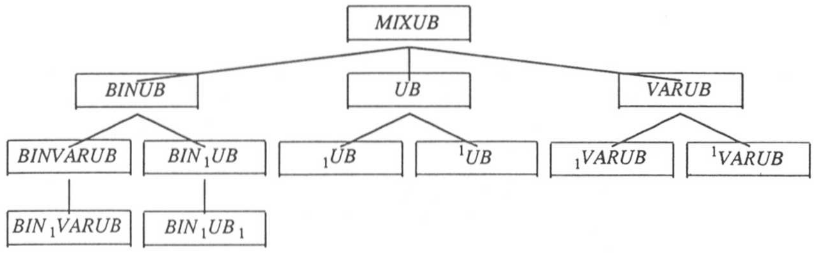

Figure 1. Hierarchical structure of less-than-or-equal constraints.

Classification	Inequality	Name
BIN_1EQ_1	$\sum_{j\in B} x_j=1$	generalized upper bound
BIN_1UB_1	$\sum_{j\in B} x_j\leq 1$	special ordered set
BIN_1UB	$\sum_{j\in B} x_j\leq k\ (k\neq 1)$	invariant knapsack
BIN_1VARUB	$\sum_{j\in B} x_j\leq a_k x_k$	plant-location
BIN_1VARLB	$\sum_{j\in B} x_j\geq a_k x_k$	reverse plant-location
$BINUB$	$\sum_{j\in B} a_j x_j\leq a_k$	knapsack
^1VARUB	$a_j y_j\leq a_k x_k$	variable upper bound
^1VARLB	$a_j y_j\geq a_k x_k$	variable lower bound
^1UB	$a_j y_j\leq a_k$	simple upper bound
^1LB	$a_j y_j\geq a_k$	simple lower bound

Table 1. Examples of classifications.

References

H. Crowder, E.L. Johnson, M.W. Padberg (1983). Solving large-scale zero-one linear programming problems. *Oper. Res. 31*, 803-834.

K.L. Hoffman, M. Padberg (1985). LP-based combinatorial problem solving. *Annals of Oper. Res. 4*, 145-194.

K.L. Hoffman, M. Padberg (1989). *ABC_OPT: A branch-and-cut optimizer for solving large 0-1 linear programming problems.* (In preparation)

K.L. Hoffman, M. Padberg (1991). Improving LP-representation of zero-one linear programs for branch-and-cut. *ORSA J. Comput. 3*, 121-134.

IBM Corporation (1990). *Optimization Subroutine Library, Guide and Reference.*

G.L. Nemhauser, L.A. Wolsey (1988). *Integer Programming and Combinatorial Optimization.* Wiley, Chichester.

M. Padberg, G. Rinaldi (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review 33*, 60-100.

T. van Roy, L.A. Wolsey (1986). Solving mixed 0-1 programs by automatic reformulation. *Oper. Res. 35*, 45-57.

M.W.P. Savelsbergh, G.C. Sigismondi, G.L. Nemhauser (1991). *MINTO, a Mixed INTeger Optimizer.* Memorandum COSOR 91-18, Eindhoven University of Technology; also appeared as Computation Optimization Report 91-03, Georgia Institute of Technology.

L.A. Wolsey (1990). Valid inequalities for 0-1 knapsacks and MIPS with generalised upper bound constraints. *Discrete Applied Mathematics 29*, 251-261.

[10] combines a conjugate gradient method to explore a face of the feasible polytope and a projected line search strategy to select a new face. The interior point algorithm proposed by Han, Pardalos and Ye [4] is (at least from a theoretical point of view) one of the most effective, requiring a total of $O(\sqrt{n}L)$ iterations, where L is the size of the input data of the problem. A review of the recent algorithms based on active set strategies can be found in [10, 9, 3]. We refer the reader to [4] for an outline of the results for interior point based algorithms.

In this paper we show that important large–scale engineering problems can be solved using both serial and parallel successive overrelaxation (SOR) methods [6, 7]. The effectiveness of the SOR algorithm in solving large sparse linear complementarity problems and linear programs derives in part from the fact that original problem data are never modified and the sparsity structure of the matrix M is preserved throughout the algorithm.

For all the problems considered here, the matrix M is pentadiagonal. This special sparsity structure of the matrix allowed us the effectively implement the SOR algorithm on the massively parallel Single–Instruction–Multiple–Data (SIMD) Connection Machine CM–2.

In a previous paper [3], we concentrated on the obstacle problem and results on the Connection Machine CM–2 and the MasPar MP–1 were reported. Two different implementations of the algorithm in Fortran 90 and C* were discussed. For larger problems the C* implementation was more efficient, reducing the computing time in some cases to 40% less than the Fortran 90 implementation. We attributed this to lower inter-processor communication costs in the C* implementation (see [3]).

In this paper, we report computational results for three classes of problems: obstacle problems, elastic–plastic torsion problems, and journal bearing problems. We compare the results of our algorithm with the results for the Han, Pardalos and Ye algorithm [4] and with the results reported by McKenna, Mesirov and Zenios in [11] for their implementation of the Moré and Toraldo [10] algorithm on the Connection Machine CM–2.

We briefly describe our notation now. Given the vectors l and u (with $l \leq u$) and a vector x all in $\mathbb{R}^n$, $x_\#$ will denote the vector with components $(x_\#)_i = \min\{u_i, \max\{l_i, x_i\}\}$. The scalar product of two vectors x and y in $\mathbb{R}^n$ will be denoted by x^Ty. The symbol $:=$ denotes definition of the term on the left side of the symbol.

2 The serial SOR algorithm

In this section we will discuss a serial implementation of the SOR algorithm for the quadratic program with simple bounds (1).

A point $x \in \mathbb{R}^n$ solves the quadratic program (1) if and only if ([6])

$$x = (x - \omega E(Mx + q))_\#$$

for some positive diagonal matrix E and some $\omega > 0$. The above formula is the basis of the SOR algorithm. The successive overrelaxation algorithm constructs a sequence of iterates $\{x^k\}$ as follows:

SOR Algorithm

For any initial feasible $x^0 \in [l, u]$, generate the sequence $\{x^k\}, k = 0, 1, 2, \ldots$, as follows:

$$x_i^{k+1} := \left(x_i^k - \omega E_{ii} \left(\sum_{j<i} M_{ij} x_j^{k+1} + \sum_{j \geq i} M_{ij} x_j^k + q_i \right) \right)_{\#} \qquad (2)$$

for $i = 1, 2, \ldots, n$.

In our implementation we used $E_{ii} = M_{ii}^{-1}$. Convergence of the algorithm can be established if the relaxation parameter ω is chosen in the interval $(0, 2)$ [5].

We refer the reader to [3] for more information on the serial implementation of the SOR algorithm including the data structure used for storing the matrix M and the choice of the relaxation parameter ω. For all the problems considered here, the value of ω was fixed to 1.95.

A simple but very effective "fixing–strategy" for the problem variables was implemented. During the updating process, the algorithm recorded if (and how many times) a particular variable remained fixed at its upper or lower bound. Based on this information, certain variables were held fixed for a number of iterations. A resetting step was performed every 30 iterations to verify the appropriateness of fixing these variables. We estimate that a 15% reduction in solution time was achieved by this strategy.

3 The massively parallel SOR algorithm

Although the SOR algorithm is inherently serial, two distinct components x_i and x_j (with $i < j$) can be concurrently updated using the above formula (2) provided that $M_{jr} = 0$ for all $r = i, i+1, \ldots, j-1$. This observation is the basis of our massively parallel implementation of the algorithm.

The quadratic programming problems considered in our parallel implementations arise as finite element approximations to elliptic variational inequalities. These approximations are obtained by triangulating the unit square, giving rise to a grid. In this context it is more convenient to describe our parallel implementation in terms of an $m \times m$ matrix X. Then $n = m^2$ and

$$X := \begin{bmatrix} X_{1m} & X_{2m} & \ldots & X_{mm} \\ & \ldots & \ldots & \\ X_{12} & X_{22} & \ldots & X_{m2} \\ X_{11} & X_{21} & \ldots & X_{m1} \end{bmatrix}, \quad x := [X_{11}, X_{21}, \ldots, X_{m1}, X_{12}, \ldots, X_{m2}, \ldots, X_{1m}, \ldots, X_{mm}].$$

At each grid point the value of a piecewise linear function at the vertices of the triangulation of the problem domain is stored (see also [10]).

To update a particular component X_{ij}, the SOR algorithm requires the value X_{ij}, along with its north, south, east and west neighbors in X. This special structure allowed us to take advantage of the NEWS (North–East–West–South) communication grid of the Connection Machine CM–2.

1	[a single variable]

2.3. *Coefficients of variables*

The third field specifies the coefficients of the variables that appear in the left-hand-side of the constraint, i.e., whether they all have coefficient c or whether they have arbitrary coefficients.

<coefficients of variables> ::= ∘ ∨ c

∘	[arbitrary coefficients]
c	[all coefficients equal to c]

2.4. *Type of bound*

The fourth field specifies the type of bound. The type of bound can be either simple or variable. In constraints with a variable bound all variables in the left-hand-side have positive coefficients and there is a single binary variable with a positive coefficient in the right-hand-side.

<type of bound> ::= ∘ ∨ VAR

∘	[simple bound]
VAR	[variable bound]

2.5. *Sense of bound*

The fifth field specifies the sense of bound. The sense of bound is determined by the sense of the constraint, which can be either $\leq$, $=$, or $\geq$.

<sense of bound> ::= UB ∨ EQ ∨ LB

UB	[upper bound]
EQ	[equal bound]
LB	[lower bound]

2.6. *Value of bound*

The sixth field specifies the value of the bound, i.e., whether it is equal to a specific value c or whether it can be an arbitrary value.

<value of bound> ::= ∘ ∨ c

∘	[arbitrary bound]
c	[bound equal to c]

The language presented above defines a hierarchical structure of constraint classes. Figure 1 explicitly shows this hierarchical structure for the constraints with sense $\leq$. We now define for each constraint a unique constraint class by assigning it to the smallest class in the hierarchical structure that contains it. Table 1 illustrates the classification scheme by presenting classes for several types of constraints encountered in the literature.

The language we propose to define constraint classes uses six fields. The first field describes the type of variables that occur in the constraint. The second field specifies the number of variables in the constraint. The third field characterizes the coefficients of the variables in the constraint. The fourth field describes the type of bound. The fifth field specifies the sense of the bound. The sixth field characterizes the value of the bound.

The classification language consists of a set of rules that define allowable structures. Each rule defines a nonterminal symbol (classification or field) in terms of other nonterminal symbols and terminal symbols (values of fields, or 'tokens'); the symbol ∨ is used to represent an exclusive or. Each nonterminal symbol is enclosed in angular brackets. Each token is followed by a comment on its interpretation between square brackets. The token ∘ indicates the empty symbol; it is used to indicate a default value, which is usually either the simplest or the most appropriate value.

Each constraint class under consideration is defined by a number of tokens, some of which may be equal to ∘. For notational convenience, the tokens are represented as follows

$$\langle \mathit{field}\ 1\rangle \substack{\langle \mathit{field}\ 2\rangle \\ \langle \mathit{field}\ 3\rangle} \langle \mathit{field}\ 4\rangle \langle \mathit{field}\ 5\rangle_{\langle \mathit{field}\ 6\rangle}$$

The partitioning of the set of constraints into a number of classes, as done by mixed integer programming systems, is in fact nothing more than the identification of interesting special cases.

<classification> ::=

<type of variables>
<number of variables>
<coefficients of variables>
<type of bound>
<sense of bound>
<value of bound>

2.1. *Type of variables*

The first field specifies the type of variables that appear in the left-hand-side of the constraint, i.e., whether there are both binary and non-binary variables, just binary variables, or just non-binary variables.

<type of variables> ::= ∘ ∨ BIN ∨ MIX

∘	[no binary variables]
BIN	[all binary variables]
MIX	[both binary and non-binary variables]

2.2. *Number of variables*

The second field specifies the number of variables that appear in the left-hand-side of the constraint, i.e., whether there is only a single variable or whether there is more than one variable. Note that in the case of a left-hand-side with both binary and non-binary variables there can never be a single variable.

<number of variables> ::= ∘ ∨ 1

∘	[an arbitrary number of variables]

In our C* implementation, the processors were organized in an $p \times p$ grid and assigned a $k \times k$ submatrix of X, where $n = p^2 \times k^2$. A standard red–black coloring [8] was imposed on X so that each processor would be active at every time step; all red components were updated simultaneously, followed by all black components.

4 Performance of the serial and parallel algorithms

The test problems considered are instances of three classes of problems: the obstacle problem, the elastic–plastic torsion problem, and the journal bearing problem. These three problems can be posed as the following constrained variational problem:

$$\min\{q(v) : v \in K\}.$$

For the obstacle problem and the elastic-plastic torsion problem, the objective function is given by

$$q(v) = 1/2 \int_{\mathcal{D}} \|\nabla v\|^2 \, d\mathcal{D} - f \int_{\mathcal{D}} v d\mathcal{D},$$
$$\mathcal{D} = (0,1) \times (0,1),$$

where ∇ is the Laplacian operator, and K is the subset of all functions v with compact support on $\mathcal{D}$ such that v and $\|\nabla v\|^2$ belong to the square integrable class $L^2(\mathcal{D})$.

For the obstacle problem, v varies between the bounds v_l and v_u given in Table 1 and the force $f = 1$.

Problem	v_l	v_u
1	$(\sin(9.2x_1)\sin(9.3x_2))^3$	$(\sin(9.2x_1)\sin(9.3x_2))^2 + 0.02$
2	$\sin(3.2x_1)\sin(3.3x_2)$	2000.0

TABLE 1: *Lower and upper bounds for the obstacle problems.*

For the elastic–plastic torsion problem, $f = c$ for some constant c, and the bounds on v are given by

$$\{|v(x)| \le dist(x, \partial\mathcal{D}), x \in \mathcal{D}\}.$$

where $dist(\cdot, \partial\mathcal{D})$ is the distance function to the boundary $\partial\mathcal{D}$.

We considered the three cases $c = 5$, $c = 10$, and $c = 20$.

The journal bearing problem has an objective function given by

$$q(v) = 1/2 \int_{\mathcal{D}} (1 + \epsilon \cos\theta)^3 \|\nabla v\|^2 \, d\mathcal{D} - \epsilon \int_{\mathcal{D}} \sin\theta v d\mathcal{D},$$

where

$$\mathcal{D} = \{(\theta, y) : 0 < \theta < 2\pi, 0 < y < 2b\},$$

$$\{v \geq 0\}.$$

In our tests, we set $b = 10$, and used $\epsilon = 0.1$ and $\epsilon = 0.5$.

Finite element approximations give rise to a quadratic minimization problem with a finite number of variables. For all three classes of problems, the matrix M is a pentadiagonal matrix. For the obstacle problem and elastic–plastic torsion problem, M has diagonal entries of 4 and off–diagonal entries of -1. While the matrix M for the journal bearing problem has the same pentadiagonal structure, diagonal and off–diagonal entries are more complicated to compute.

In tables 2 to 11 the computational results for the serial and parallel version of the SOR algorithm are reported. The serial results were obtained on an IBM RISC 6000 POWERstation 550 while the massively parallel results were obtained on a Connection Machine CM–2 using 8K and 16K processors. All numerical computation was carried out in double precision.

The results for the obstacle problem are reported in Tables 2 to 5. Table 2 shows the number of iterations, solution time (in seconds) and number of free variables at the optimum (i.e., the number of variables neither at the at the lower or upper bound) for the first of two obstacle problems. The number of variables ranges from 10,000 to 490,000 which corresponds to a grid with the number of points varying from 100x100 to 700x700. The fifth column contains solution accuracy defined as:

$$\text{Accuracy} := \left\| x^* - (x^* - (Mx^* + q))_{\#} \right\|_{\infty}.$$

The column labeled HPY reports solution times for the same problems with the Han–Pardalos–Ye interior point algorithm [4] on a IBM 3090–600S supercomputer with Vector Facilities. In their implementation they took full advantage of the special pentadiagonal structure of the matrix M to solve the system of linear equations arising at every iteration.

For the problems we tested, the solution time for our serial SOR algorithm was lower than the time required by the Han–Pardalos–Ye algorithm. The solution time ratio for the two algorithms is shown in the last column. The time per iteration for the SOR algorithm grows linearly with the number of variables while the number of iterations grows sublinearly. For Problem 1, about 1/5 of the variables were at their lower or upper bound at the optimum, and for Problem 2, almost 60% of the variables were at bound at the optimum.

In Table 4 we compare the massively parallel results obtained with our Fortran 90 implementation with the results obtained by McKenna, Mesirov and Zenios for their implementation of the Moré and Toraldo [10] algorithm. Their results are reported in the column labeled MMZ.

Table 5 reports the solution time for the faster C* implementation of SOR for both types of obstacle problems using 8K and 16K processors. The number of variables ranged from 16,384 to 1,048,576. The final solution accuracy for all was at least 10^{-7}. In all instances, the speedup efficiency was always very high: always above 84% and in many cases over 90%. We also note that, using 16K processors, we were able to reduce the solution time by a factor of 30 over the serial algorithm on the IBM RISC 6000 POWERstation 550.

Tables 6 to 9 report the serial and parallel results for the elastic–plastic torsion problems for different values of c. Two starting points are considered: $x^0 = 0$ and $x^0 = u$. Tables 6 and 7 show the serial results for the two different starting points. All problems were solved within an accuracy of at least 10^{-7}.

The first step in the analysis of the constraint matrix is the classification of each constraint. The set of constraints is partitioned into a number of general types. The partition should be based on the specific structures that a system uses for its preprocessing, constraint generation, and branching strategies. As a result, there is no consensus on terminology and classification scheme. Generalized upper bound constraints, for instance, are defined as $\sum_{j \in S} x_j = 1$ in Nemhauser and Wolsey [1988] and as $\sum_{j \in S} x_j \leq 1$ in Wolsey [1990]

In this note, we define a classification scheme that is used in our system MINTO [Savelsbergh, Sigismondi and Nemhauser 1991]. Its purpose is to identify important classes to be used in preprocessing, constraint generation, branching, etc. We propose it as a general scheme to be evaluated, modified and then, hopefully, adopted, by the mixed integer programming community.

2. Constraint classification

A general mixed integer programming problem is of the form

$$\max \sum_{j \in B} c_j x_j + \sum_{j \in I} c_j x_j + \sum_{j \in C} c_j x_j$$

subject to

$$\begin{array}{ll} \sum_{j \in B} a_{ij} x_j + \sum_{j \in I} a_{ij} x_j + \sum_{j \in C} a_{ij} x_j \sim b_i & i = 1, \ldots, m \\ 0 \leq x_j \leq 1 & j \in B \\ l_{xj} \leq x_j \leq u_{xj} & j \in I \cup C \\ x_j \in \mathbb{Z} & j \in B \cup I \\ x_j \in \mathbb{R} & j \in C \end{array}$$

where B is the set of binary variables, I is the set of integer variables, C is the set of continuous variables, the sense $\sim$ of a constraint can be $\leq, \geq$, or $=$, and the lower and upper bounds may be plus or minus infinity. See Nemhauser and Wolsey [1988] for a general treatment of the subject.

To classify constraints, we first distinguish binary variables from integer and continuous ones. Note that this is different from a variable classification that surely would separate integer and continuous variables if for no other reason than the need to do so in branching. However, we have not yet found a significant use of constraints that distinguish between integer and continuous variables, e.g., we do not use Gomory mixed integer cuts. We use the symbol y to indicate integer and continuous variables and x to indicate binary variables. Each constraint class will be an equivalence class with respect to complementing binary variables, i.e., if a constraint with term $a_j x_j$ is in a given class then the constraint with $a_j x_j$ replaced by $a_j(1-x_j)$ is also in the class. Consequently, the most general constraint that can appear in a mixed integer programming formulation can be represented as follows

$$\sum_{j \in B} a_j x_j + \sum_{j \in I \cup C} a_j y_j \sim b,$$

where a_j for $j \in B$ and b are positive, and a_j for $J \in I \cup C$ are nonzero.

Furthermore, we distinguish variable bounds from simple bounds. In a constraint with a variable bound, there is a distinct binary variable that bounds all others if it is set to either 0 or 1. The most general constraint with a variable bound can be represented as follows

$$\sum_{j \in B} a_j x_j + \sum_{j \in I \cup C} a_j y_j \sim a_k x_k,$$

where a_j for $j \in B$, a_j for $J \in I \cup C$, and a_k are all positive. Whenever we consider a constraint with a variable bound, we will assume that the distinct binary variable appears in the right-hand-side of the constraint and all other variables appear in the left-hand-side of the constraint.

Constraint Classification for Mixed Integer Programming Formulations

G.L. Nemhauser

Georgia Institute of Technology
School of Industrial and System Engineering
Atlanta, GA 30332-0205
USA

M.W.P. Savelsbergh

Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

G.C. Sigismondi

Georgia Institute of Technology
School of Industrial and System Engineering
Atlanta, GA 30332-0205
USA

1. Introduction

The success of branch-and-cut algorithms for combinatorial optimization problems [Hoffman and Padberg 1985, Padberg and Rinaldi 1989] and large scale 0-1 linear programming problems [Crowder, Johnson, and Padberg, 1983] has lead to a renewed interest in mixed integer programming. The key idea of the branch-and-cut approach is reformulation. Problems are reformulated so as to make the difference in the objective function values between the solutions to the linear programming relaxation and the integer program as small as possible.

There are various ways to tighten the linear programming relaxation of an integer program. Preprocessing techniques [Hoffman and Padberg, 1991] try, among others things, to reduce the size of coefficients in the constraint matrix and to reduce the size of bounds on the variables. Constraint generation techniques [Crowder, Johnson and Padberg, 1983, Van Roy and Wolsey, 1986] try to generate strong valid inequalities.

Reformulation techniques should make the best possible use of the problem structure. It is beneficial to distinguish two modes of operation. General reformulation techniques, which are embedded in mixed integer programming systems such as ABC_OPT [Hoffman and Padberg 1989], MINTO [Savelsbergh, Sigismondi and Nemhauser 1991], MPSARX [Van Roy and Wolsey 1986], and OSL [IBM Corporation, 1990] try to identify problem structure based on an analysis of the constraint matrix. Problem specific reformulation techniques are based on an a priori investigation of the polyhedron associated with the set of feasible solutions.

Once again in Table 6 we compare our results with the results reported for the Han–Pardalos–Ye interior point algorithm. In all instances, the solution time for our serial SOR algorithm is substantially lower than the time required by the Han–Pardalos–Ye method. The time ratio is bigger for the "easier" problems. For the case $c = 20$, the solution time was reduced 10–fold when $n = 490,000$, $x^0 = u$, while a maximum reduction of only 2.8 was achieved for the case $c = 5$.

The results for $x^0 = 0$ are reported in Table 7. The same starting point was also used in [4]. However, no results for problems in this class with more than 160,000 variables were reported and their algorithm failed to converge for the case $n = 160,000$ and $c = 20$.

The next two tables report the results for our massively parallel implementation. Due to memory limitation we were unable to run large problems with 8K processors. Using 16K processors a 30–fold time reduction was achieved over the serial counterpart. Problems with over 4,000,000 variables were solved with solution times varying from 7 minutes ($c = 20, x^0 = u$) to less than 50 minutes ($c = 5, x^0 = u$).

Finally Tables 10 and 11 report the solution times for the journal bearing problem for the serial and parallel implementation, respectively. Two problems were considered here with $\epsilon = 0.1$ and $\epsilon = 0.5$. Results for these problems were reported in [10] for n varying from 5625 to 15625. However only the number of iterations and number of function evaluations per iteration were reported. Once again on the IBM RISC 6000 POWERstation 550, we solved problems with up to 490,000 variables and on the 16K Connection Machine CM–2, problems with over a million variables. Due to memory limitation we were unable to run problems with more than 1,048,576 variables even with 16K processors.

5 Conclusions

We have implemented both serial and massively parallel SOR algorithms to solve several large scale engineering problems. On a 16,384-processor Connection Machine CM–2, we were able to solve problems with over 4,000,000 variables in less than 50 minutes. To the best of our knowledge, these are among the largest problems in this class ever attempted.

References

[1] G. Cimatti, *On a problem of the theory of lubrication governed by a variational inequality*, Applications of Mathematical Optimization, 3 (1977), pp. 227–242.

[2] G. Cimatti and O. Menchi, *On the numerical solution of a variational inequality connected with the hydrodynamic lubrication of a complete journal bearing*, Calcolo, 15 (1978), pp. 249–258.

[3] R. De Leone and M.A. Tork Roth, *Massively parallel solution of quadratic program via successive overrelaxation*, Tech. Report TR # 1041, University of Wisconsin, Computer Sciences Dept., August 1991.

[4] C. Han, P.M. Pardalos, and Y. Ye, *Solving some engineering problems using an interior-point algorithm*, Tech. Report CS-91-04, Department of Computer Science, The Pennsylvania State University, Pennsylvania, 1991.

[5] Z.-Q. Luo and P. Tseng, *On the convergence of a matrix splitting algorithm for the symmetric monotone linear complementarity problem*, Tech. Report LIDS-P-1884, Laboratory for Information and Decision System, Massachusetts Institute of Technology, Cambridge, 1990. to appear *SIAM Journal on Control and Optimization.*

[6] O.L. Mangasarian, *Solution of symmetric linear complementarity problems by iterative methods*, Journal of Optimization Theory and Applications, 22 (1977), pp. 465–485.

[7] ——, *Sparsity-preserving sor algorithms for separable quadratic and linear programming*, Computer and Operations Research, 11 (1984), pp. 105–112.

[8] J.J. Modi, *Parallel Algorithms and Matrix Computation*, Clarendon Press, Oxford, England, 1988.

[9] J.J. Morè, *On the performance of algorithms for large-scale bound constrained problems*, Tech. Report MCS-P140-0290, Argonne National Laboratory, Argonne, Illinois, 1990.

[10] J.J. Morè and G. Toraldo, *On the solution of large quadratic programming problems with bound constraints*, SIAM Journal on Optimization, 1 (1991), pp. 93–113.

[11] Mc Kenna M.P., Mesirov J.P., and S.A. Zenios, *Data parallel quadratic programming on box-constrained problems*, Tech. Report 91-04-01, Decison Sciences Department, The Warton School, Philadelphia, PA 19104, October 1991.

[14] Roos, C. and Vial, J.–Ph. (1989), Long Steps with the Logarithmic Penalty Barrier Function in Linear Programming, in *Economic Decision–Making: Games, Economics and Optimization*, dedicated to Jacques H. Drèze, edited by J. Gabszevwicz, J.–F. Richard and L. Wolsey, Elsevier Science Publisher B.V., 433–441.

References

[1] Anstreicher, K.M., den Hertog, D., Roos, C., Terlaky, T. (1990), A Long Step Barrier Method for Convex Quadratic Programming, Report No. 90–53, Faculty of Mathematics and Informatics/Computer Science, Delft University of Technology, Delft, Holland.

[2] Carroll, C.W. (1961), The Created Response Surface Technique for Optimizing Nonlinear Restrained Systems, *Operations Research* 9, 169–184.

[3] Den Hertog, D., Roos, C., Vial, J.–Ph. (1990), A $\sqrt{n}$ Complexity Reduction for Long Step Path–following Methods, Revised Version of Report No. 89–85, Faculty of Mathematics and Informatics/Computer Science, Delft University of Technology, Delft, Holland, To Appear in *SIAM Journal on Optimization.*

[4] Den Hertog, D., Roos, C., Terlaky, T. (1990), On the Classical Logarithmic Barrier Method for a Class of Smooth Convex Programming Problems, Report No. 90–, Faculty of Mathematics and Informatics/Computer Science, Delft University of Technology, Delft, Holland.

[5] Fiacco, A. V. and McCormick, G. P. (1968), Nonlinear Programming, Sequential Unconstrained Minimization Techniques, Wiley and Sons, New York.

[6] Frisch, R. (1955), The Logarithmic Potential Method for Solving Linear Programming Problems, Memorandum, University Institute of Economics, Oslo.

[7] Gonzaga, C.C. (1989), An Algorithm for Solving Linear Programming Problems in $O(n^3L)$ Operations, In *Progress in Mathematical Programming, Interior Point and Related Methods*, 1–28, N. Megiddo ed., Springer Verlag, New York.

[8] Gonzaga, C.C. (1989), Large–Steps Path–Following Methods for Linear Programming: Barrier Function Method, Report ES–210/89, Department of Systems Engineering and Computer Sciences, COPPE–Federal University of Rio de Janeiro, Rio de Janeiro, Brasil.

[9] Hamala, M. (1976), Quasibarrier Methods for Convex Programming, *Survey of Mathematical Programming* 1, 465–477, Proceedings of the IX-th International Symposium on Mathematical Programming (Budapest, 1976), North–Holland.

[10] Karmarkar, N. (1984), A New Polynomial–Time Algorithm for Linear Programming, *Combinatorica* 4, 373–395.

[11] Kowalik, J. (1966), Nonlinear Programming Procedures and Design Optimization, *Acta Polyntech. Scand.* 13, Trondheim.

[12] Megiddo, N. (1989), Pathways to the Optimal Set in Linear Programming, In *Progress in Mathematical Programming, Interior Point and Related Methods*, 131–158, N. Megiddo ed., Springer Verlag, New York.

[13] Roos, C. and Vial, J.–Ph. (1988), A Polynomial Method of Approximate Centers for Linear Programming, Report No. 88–68, Faculty of Mathematics and Informatics/Computer Science, Delft University of Technology, Delft, Holland, To Appear in *Mathematical Programming.*

n	# iter	# free	time	accuracy	HPY	time ratio
10,000	306	7,588	6.30	$0.1848 * 10^{-7}$	16.3	2.59
40,000	323	31,239	26.86	$0.1643 * 10^{-7}$	131.1	4.88
90,000	361	70,896	68.10	$0.2027 * 10^{-7}$	437.6	6.43
115,600	393	91,363	95.89	$0.1900 * 10^{-6}$	700.3	7.30
160,000	449	126,904	148.54	$0.6603 * 10^{-8}$	1035.8	6.97
250,000	608	198,759	314.58	$0.5813 * 10^{-7}$	2110.5	6.71
360,000	872	286,712	750.58	$0.1171 * 10^{-7}$	4090.3	5.45
490,000	851	390,736	848.79	$0.9143 * 10^{-7}$	8977.8	10.58

TABLE 2: *Comparison of serial SOR algorithm on the IBM RISC 6000 POWERstation 550 and the HPY algorithm on the IBM 3090-600S. Obstacle Problem 1.*

n	# iter	# free	time	accuracy	HPY	time ratio
10,000	191	3,843	3.53	$0.2056 * 10^{-6}$	25.4	7.20
40,000	417	15,880	32.03	$0.1244 * 10^{-7}$	203.9	6.37
90,000	459	36,160	80.44	$0.1288 * 10^{-6}$	699.9	8.70
115,600	535	46,528	124.15	$0.7085 * 10^{-7}$	1018.7	8.21
160,000	780	64,639	241.42	$0.6745 * 10^{-8}$	1534.7	6.36
250,000	1114	101,242	560.28	$0.5935 * 10^{-7}$	3141.9	5.61
360,000	1594	146,167	1357.48	$0.5659 * 10^{-7}$	5312.4	3.91

TABLE 3: *Comparison of serial SOR algorithm on the IBM RISC 6000 POWERstation 550 and the HPY algorithm on the IBM 3090-600S. Obstacle Problem 2.*

n	# iter	time	MMZ
90,000	480	8.3	19.27
160,000	620	18.6	40.31
250,000	980	57.4	73.05
360,000	1460	87.0	110.55
490,000	2000	139.7	-
640,000	2620	264.2	-
810,000	3330	546.9	-

TABLE 4: *Comparison of parallel Fortran–90 SOR algorithm and MMZ algorithm on the Connection Machine CM-2 with 8K processors. Obstacle Problem 2.*

n	Problem 1			Problem 2		
	# iter	8K	16K	# iter	8K	16K
16,384	320	3.39	2.00	360	3.66	2.16
65,536	320	3.40	2.00	440	4.51	2.68
262,144	540	17.58	10.14	1040	32.04	18.57
1,048,576	2280	253.77	143.36	4240	472.47	249.86

TABLE 5: *Solution time in seconds for the parallel C* SOR algorithm on the Connection Machine CM-2 with 8K and 16K processors. Obstacle Problems 1 and 2.*

Proof:
Using convexity of the f_i's and (1) we may write

$$f_0(\overline{y}) - f_0(y) + \sum_{i=1}^{n} x_i(f_i(\overline{y}) - f_i(y)) \geq \nabla f_0(y)^T(\overline{y} - y) + \sum_{i=1}^{n} x_i \nabla f_i(y)^T(\overline{y} - y) = 0.$$

Using this, we get for $\lambda \neq 0$,

$$\phi_\lambda^p(\overline{y}, \mu) - \phi_\lambda^d(x, y, \mu) \geq \sum_{i=1}^{n} x_i f_i(\overline{y}) + \mu \sum_{i=1}^{n} \frac{1}{\lambda}(-f_i(\overline{y}))^{-\lambda} - \mu^{\frac{1}{\lambda+1}} \sum_{i=1}^{n} \frac{\lambda+1}{\lambda} x_i^{\frac{\lambda}{\lambda+1}}. \tag{5}$$

Now setting the derivatives with respect to x_i of the right hand side equal to zero, we get

$$x_i = \frac{\mu}{(-f_i(\overline{y}))^{\lambda+1}}.$$

This choice of x_i minimizes the right hand side of (5), since it is convex in x_i. Substituting this into (5), we get

$$\phi_\lambda^p(\overline{y}, \mu) - \phi_\lambda^d(x, y, \mu) \geq 0.$$

It is easy to verify that equality holds for $\overline{y} = y = y(\mu)$ and $x = x(\mu)$. The logarithmic case $(\lambda = 0)$ can be proved in the same way. □

As a consequence of Theorem 1 we have that $(x(\mu), y(\mu))$ maximizes $\phi_\lambda^d(x, y, \mu)$. Now we are ready to give the main result.

Theorem 2 *The objective function of the dual problem (D)*

$$f_0(y(\mu)) + \sum_{i=1}^{n} x_i(\mu) f_i(y(\mu))$$

is monotonically increasing if μ decreases, where $y(\mu)$ is on the λ–path, $\lambda > -1$.

Proof:
The proof follows easily by applying Fiacco and McCormick's proof to $\phi_\lambda^d(x, y, \mu)$. We will only give the proof for $\lambda \neq 0$; the logarithmic barrier case can be treated similarly. Since $(x(\mu), y(\mu))$ maximizes $\phi_\lambda(x, y, \mu)$ and $(x(\overline{\mu}), y(\overline{\mu}))$ maximizes $\phi_\lambda(x, y, \overline{\mu})$, we have

$$\phi(x(\mu), y(\mu), \mu) \geq \phi(x(\overline{\mu}), y(\overline{\mu}), \mu), \tag{6}$$

and

$$\phi(x(\overline{\mu}), y(\overline{\mu}), \overline{\mu}) \geq \phi(x(\mu), y(\mu), \overline{\mu}). \tag{7}$$

Multiplying (6) by $\overline{\mu}^{\frac{1}{\lambda+1}}$ and (7) by $\mu^{\frac{1}{\lambda+1}}$ and adding up, results into

$$(\overline{\mu}^{\frac{1}{\lambda+1}} - \mu^{\frac{1}{\lambda+1}}) \left(f_0(y(\mu)) + \sum_{i=1}^{n} x_i(\mu) f_i(y(\mu)) - (f_0(y(\overline{\mu})) + \sum_{i=1}^{n} x_i(\overline{\mu}) f_i(y(\overline{\mu})) \right) \geq 0.$$

Hence if $\overline{\mu} < \mu$ then $f_0(y(\mu)) + \sum_{i=1}^{n} x_i(\mu) f_i(y(\mu)) \leq f_0(y(\overline{\mu})) + \sum_{i=1}^{n} x_i(\overline{\mu}) f_i(y(\overline{\mu}))$, which proves the theorem. □

For convenience we put a factor $\frac{1}{\lambda}$ before the barrier if $\lambda \neq 0$. So for $\lambda > -1,\ \lambda \neq 0$ we have the (quasi) barrier function

$$\phi^p_\lambda(y,\mu) = f_0(y) + \mu \sum_{i=1}^{n} \frac{1}{\lambda}(-f_i(y))^{-\lambda},$$

and $\lambda = 0$ corresponds to the logarithmic barrier function

$$\phi^p_0(y,\mu) = f_0(y) - \mu \sum_{i=1}^{n} \ln(-f_i(y)).$$

It can be proved that $\phi^p_\lambda(y,\mu)$, for $\lambda \geq -1$, has a unique minimum, which is denoted by $y(\mu)$ (see [5] and [9]). The necessary and sufficient Karush–Kuhn–Tucker conditions for these minima are:

$$f_i(y) \leq 0,\ \ 1 \leq i \leq n, \tag{2}$$
$$\textstyle\sum_{i=1}^{n} x_i \nabla f_i(y) + \nabla f_0(y) = 0,\ \ x \geq 0, \tag{3}$$
$$(-f_i(y))^{\lambda+1} x_i = \mu,\ \ 1 \leq i \leq n. \tag{4}$$

We will call the minimizing point $y(\mu)$ the center of (CP) with respect to μ.

The λ–path of the problem is defined as the set of centers $y(\mu)$, where μ runs from ∞ to 0. Note that not only $y(\mu)$ is primal feasible, but also $(x(\mu), y(\mu))$ is dual feasible, due to equation (3). ¿From the literature it is well-known that if $\mu \downarrow 0$ then $y(\mu) \to y^*$, where y^* is an optimal solution for (CP). Moreover, $f_0(y(\mu))$ is monotonically decreasing if μ decreases.

For the logarithmic barrier function ($\lambda = 0$) it is easy to show that the duality gap along the path is decreasing. See for example Fiacco and McCormick [5]. For linear programming and quadratic programming it is well-known that the dual objective along the 0–path (the logarithmic case) is monotonically increasing if μ decreases. The argument then is that in this special case the dual pair $(x(\mu), y(\mu))$ is the unique solution for the dual logarithmic barrier problem. As we now will show a similar argument can be used for the general case.

We introduce the following functions which are in fact the duals of ϕ^p_λ, $\lambda \neq 0$, and ϕ^p_0, respectively,

$$\phi^d_\lambda(x,y,\mu) = f_0(y) + \sum_{i=1}^{n} x_i f_i(y) + \mu^{\frac{1}{\lambda+1}} \sum_{i=1}^{n} \frac{\lambda+1}{\lambda} x_i^{\frac{\lambda}{\lambda+1}},$$

and

$$\phi^d_0(x,y,\mu) = f_0(y) + \sum_{i=1}^{n} x_i f_i(y) + \mu \sum_{i=1}^{n} \ln x_i + n\mu(1 + \ln \mu).$$

So, loosely speaking, the dual of an inverse barrier function is a quasi–barrier function, and reversely, and the dual of the logarithmic barrier function is again the logarithmic barrier function.

Theorem 1 *We have that $\phi^p_\lambda(\overline{y},\mu) \geq \phi^d_\lambda(x,y,\mu)$ for all primal feasible $\overline{y}$ and dual feasible (x,y). Moreover, $\phi^p_\lambda(x(\mu),\mu) = \phi^d_\lambda(x(\mu),y(\mu),\mu)$*

n	c = 5			c = 10			c = 20		
	# iter	time	HPY	# iter	time	HPY	# iter	time	HPY
10,000	475	9.09	12.4	469	6.54	8.8	467	5.14	5.4
40,000	672	50.68	104.9	548	29.71	71.0	539	22.74	43.0
90,000	1763	291.68	354.5	654	77.98	236.7	656	58.99	146.1
160,000	1526	501.61	911.5	724	149.87	597.5	703	109.72	381.1
250,000	2288	1034.80	1810.5	563	190.78	1128.6	300	99.46	722.5
360,000	3157	2404.78	3338.4	815	439.45	2141.1	373	190.04	1280.7
490,000	2302	1964.54	5528.3	819	508.49	3405.8	292	190.56	2052.4

TABLE 6: *Comparison of serial SOR algorithm on the IBM RISC 6000 POWERstation 550 and the HPY algorithm, $x^0 = u$. Elastic-Plastic Torsion Problem.*

n	c = 5			c = 10			c = 20		
	# free	# iter	time	# free	# iter	time	# free	# iter	time
10,000	7016	464	8.96	3632	469	6.54	1768	467	5.15
40,000	28112	832	63.82	18416	570	31.24	7432	561	23.53
90,000	63336	2109	362.77	35328	698	88.39	16976	651	60.71
160,000	112630	2125	668.14	59736	1040	232.04	30384	766	125.48
250,000	176040	3224	1582.09	93540	1164	436.74	47696	550	173.41
360,000	253540	4512	3189.77	134804	1676	1055.16	68888	649	357.98
490,000	345114	5981	5787.66	183640	2263	1640.34	93952	827	531.67

TABLE 7: *Solution time in seconds for the serial SOR algorithm on the IBM RISC 6000 POWERstation 550, $x^0 = 0$. Elastic-Plastic Torsion Problem.*

n	c = 5			c = 10			c = 20		
	# iter	8K	16K	# iter	8K	16K	# iter	8K	16K
16,384	300	2.98	1.61	300	2.86	1.58	280	2.52	1.67
65,536	400	3.78	2.13	320	2.70	1.71	280	2.52	1.48
262,144	1860	49.94	28.73	400	10.69	6.14	260	6.95	3.99
1,048,576	6320	NEM	324.75	1700	NEM	87.28	360	NEM	18.43
4,194,304	9980	NEM	2836.75	5700	NEM	1678.15	1540	NEM	432.14

TABLE 8: *Solution time in seconds with 8K and 16K processors on the Connection Machine CM-2, accuracy* $= 10^{-7}$, $x^0 = u$. *Elastic-Plastic Torsion Problem.*

n	c = 5			c = 10			c = 20		
	# iter	8K	16K	# iter	8K	16K	# iter	8K	16K
16,384	320	3.30	1.69	320	3.05	2.07	320	2.88	1.68
65,536	320	6.62	4.21	360	3.39	2.01	340	3.06	1.82
262,144	2700	73.57	41.43	920	24.75	14.20	480	12.83	7.37
1,048,576	9680	NEM	520.44	3740	NEM	193.51	1440	NEM	75.13

TABLE 9: *Solution time in seconds with 8K and 16K processors on the Connection Machine CM-2, accuracy* $= 10^{-7}$, $x^0 = 0$. *Elastic-Plastic Torsion Problem.*

Moreover, along the path the primal objective monotonically decreases and on the path there is also available a feasible point for the dual problem associated to (CP).

Later, barrier methods became out–of–date more or less, due to the fact that they suffer from numerical difficulties in the limit. However, recent developments in polynomial time algorithms for linear programming, initiated by Karmarkar [10], renewed the interest in barrier methods. Due to this, Fiacco and McCormick's book was published again, recently. Strangely enough only the logarithmic barrier function was restudied by many researchers. This resulted in polynomial methods for linear programming based on the logarithmic barrier function. See [7], [8], [13], [14], [3] for linear and [1] for convex quadratic programming.

At least for the analysis of large–step logarithmic barrier methods it is necessary to prove that the dual objective is increasing along the path. For linear and quadratic programming this is easy: the dual feasible point $x(\mu)$ associated to $y(\mu)$ is the unique minimum for the dual barrier function, which is also convex. So, again using Fiacco and McCormick's argument for the dual barrier function gives the result. We note that Fiacco and McCormick's book does not deal with the monotonicity of the dual objective along barrier paths.[3]

The present authors [4] recently proved that the dual objective of a convex program is increasing along the path of the logarithmic barrier function. This was done by derivating the Karush–Kuhn–Tucker conditions and playing around with these equations. In that proof we needed the objective and constraint functions to be twicely continuously differentiable.

In this note we will prove a more general result. Assuming only convexity and differentiability of the functions involved, we prove that the dual objective along the paths defined by the logarithmic barrier, the inverse barrier and quasi–barrier functions, is monotonically increasing.

2 Monotonicity Along the Dual Path

In the sequel of this section we will assume that the functions $f_i(y)$ are convex and continuously differentiable, and the feasible region of (CP) is bounded and has nonempty interior. The Wolfe dual problem associated to (CP) is

$$(D) \qquad \begin{aligned} \max \quad & f_0(y) + \textstyle\sum_{i=1}^n x_i f_i(y), \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^n x_i \nabla f_i(y) + \nabla f_0(y) = 0, \\ & x_i \geq 0. \end{aligned} \tag{1}$$

Note that there is no symmetry between the primal and dual problem, as in linear programming; the dual problem contains both the variables y and x. Needless to say that the dual problem is not necessarily convex.

We consider some special cases of (quasi) barrier functions, namely such (quasi) barrier functions Ψ for which $\Psi'(\eta) = \eta^{-\lambda-1}$, for $\lambda > -1$, where λ is called the rank of the barrier. Note that $\lambda = 0$ corresponds to the logarithmic barrier, $\lambda = 1$ to the inverse barrier, and $\lambda = 2$ to the quadratic inverse barrier. Moreover, for $-1 < \lambda < 0$, this corresponds to a quasi barrier function, as developed by Hamala [9].

[3]McCormick recently told us that at that time they 'thought' it must be true, but they were unable to prove it. On page 101–102, they give a specific example for which the dual objective is really increasing.

ON THE MONOTONICITY OF THE DUAL OBJECTIVE ALONG BARRIER PATHS [1]

D. den Hertog, C. Roos and T. Terlaky [2]

October, 1991

Abstract

Under some mild conditions, barrier functions for convex programs have a unique minimum, for each value of the barrier parameter. These minima define a path, which leads to the optimum. It is well-known that the primal objective value is monotonically decreasing along this path.

In this note we show that the dual objective is increasing along the dual paths defined by the logarithmic barrier, the inverse barrier and quasi–barrier functions.

Key Words: interior point method, convex programming, barrier function, monotonicity.

1 Introduction

The classical barrier methods for solving the constrained optimization problem

$$(CP) \quad \min \{f_0(y) : \; f_i(y) \leq 0, \; i = 1, \cdots, n\},$$

are based on barrier functions of the form

$$\phi(y,\mu) = f_0(y) - \mu \sum_{i=1}^{n} \Psi(-f_i(y)).$$

Here, μ denotes a positive controlling parameter and Ψ is a function of one variable η, such that $\Psi(0_+) = -\infty$. Because of the singularity of Ψ in $\eta = 0$, these barrier functions prevent the iterates from going outside the feasible region. Many choices for Ψ have been proposed in the literature, e.g. the inverse function $-\eta^{-1}$ proposed by Carroll [2], the inverse quadratic function $-\eta^{-2}$ described by Kowalik [11], but $\Psi(\eta) = \ln \eta$, originally due to Frisch [6], has got most of the attention in the literature.

Quasi–barrier functions were developed by Hamala [9], but has not got much attention. These functions have the properties that $\Psi(0_+)$ is finite and $\Psi'(0_+) = \infty$. (E.g. $\Psi(\eta) = \sqrt{\eta}$.) It is easy to see that the minima of these functions are again in the interior of the feasible region.

In the well-known book of Fiacco and McCormick [5] barrier functions were extensively studied. They showed that, under certain mild assumptions, for each value of μ the barrier function has a unique minimum (denoted as $y(\mu)$), and that $y(\mu)$ goes to an optimum if μ goes to zero.

[1]This work is completed under the support of a research grant of SHELL.
[2]On leave from the Eötvös University, Budapest, and partially supported by OTKA No. 2116.

n	$\epsilon = 0.1$			$\epsilon = 0.5$		
	# free	# iter	time	# free	# iter	time
5625	3808	255	2.87	3359	289	3.24
10000	6768	264	5.33	6040	290	5.81
15625	10564	255	8.01	9426	288	8.94
40000	27082	378	30.58	24174	290	23.27
90000	60940	908	254.55	54402	582	107.16
160000	108438	1573	436.24	96742	1066	300.13
250000	169472	2371	987.66	151156	1648	672.42
360000	244032	3292	1928.65	217684	2328	1310.18
490000	332137	4328	3445.18	296314	3102	2319.78

TABLE 10: *Solution time in seconds for the serial SOR algorithm on the IBM RISC 6000 POWERstation 550, accuracy* = 10^{-7}. *Journal Bearing Problem.*

n	$\epsilon = 0.1$				$\epsilon = 0.5$			
	# free	# iter	8K	16K	# free	# iter	8K	16K
16,384	11072	300	3.53	2.87	9892	320	3.95	2.18
65,536	44410	480	5.67	3.49	39598	340	4.01	2.32
262,144	177682	2000	72.53	40.91	158438	1380	49.62	28.33
1,048,576		6920	NEM	485.09		5080	NEM	354.06

TABLE 11: *Solution time in seconds with 8K and 16K processors on the Connection Machine CM-2, accuracy* = 10^{-7}. *Journal Bearing Problem.*

Editorial

At the 14th Symposium in Amsterdam, the Mathematical Programming Society Council approved the re-naming of the "COAL Newsletter" to more accurately reflect the future editorial policy of the publication as outlined in the last issue (No. 19, August 1991): "...a forum for the rapid dissemination of ideas and philosophies on the present and future role of computational science in the next decade...(we) encourage the submission of thought provoking articles that report on ideas which have not been fully developed for formal communication, but which the authors are willing to share in order to stimulate research and discussion. We are also interested in receiving articles on the teaching of optimization algorithms and on the impact of new computer architectures and languages on mathematical programming."

We were pleased, but not overwhelmed, with the response to our call for submissions to future issues. This first issue of the "COAL Bulletin" contains four articles: two contributed and two invited by Bob Meyer; the latter on the theme "Mega- Problem Applications." The next issue (scheduled for publication in late 1991) is on the theme "Computational Aspects of Combinatorial Optimization" and will be guest edited by Panos Pardalos and Mauricio Resende. We welcome suggestions for future theme issues and a lively dialogue between the readers.

As a service to our readers, we shall accept for publication full page announcements and advertising that are germane to computational optimization for a fee of $300.00 per page. Software vendors and publishers are particularly encouraged to contact one of the editors for further details.

Faiz A. Al-Khayyal
Chairman & US Editor

Jens Clausen
European Editor

COAL OBJECTIVES

The Committee on Algorithms is involved in computational developments in mathematical programming. There are three major goals: (1) ensuring a suitable basis for comparing algorithms, (2) action as a focal point for computer programs that are available for general calculations and for test problems, and (3) encouraging those who distribute programs to meet certain standards of portability, testing, ease of use and documentation.

BULLETIN OBJECTIVES

The Bulletin's primary objective is to provide a vehicle for the rapid dissemination of new results in computational mathematical programming. To date, our profession has not developed a clear understanding of the issues of how computational tests should be carried out, how the results of these tests should be presented in the literature, or how mathematical programming algorithms should be properly evaluated and compared. These issues will be addressed in the Bulletin.

Professor Trond Steihaug
Institutt for informatikk
Universitetet i Bergen

Mathematical Programming Society

Committee on Algorithms

Bulletin

NO. 20	JENS CLAUSEN	EDITORS
May 1992	FAIZ A. AL-KHAYYAL	

Rønsholt Trykkeri A/S