PARALLEL AND SERIAL SOLUTION OF LARGE-SCALE LINEAR COMPLEMENTARITY PROBLEMS

by

Karen M. Thompson

Computer Sciences Technical Report #714

August 1987

PARALLEL AND SERIAL SOLUTION OF LARGE–SCALE LINEAR COMPLEMENTARITY PROBLEMS

by

Karen M. Thompson

A thesis submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
(Industrial Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

1987

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis adviser Professor Olvi L. Mangasarian for his encouragement and guidance. He has always been generous with sharing his knowledge and time. He has done much to make my experience at the University of Wisconsin a pleasant one. I could not have asked for a better adviser.

I would also like to thank Professors Robert Meyer and James Morris for reading drafts of this thesis and Professors Jerry Sanders and John Strikwerda for serving on the examination committee.

Deepankar has been an enormous source of support these past three years. He has helped me keep perspective during my low points and shared in my joy during the high points. I will always be grateful to him.

Finally, this would not be complete without thanking my parents. Without their continuing confidence in me as well as their encouragement and support this thesis would not have been possible. To them I dedicate this thesis.

PARALLEL AND SERIAL SOLUTION OF LARGE–SCALE LINEAR COMPLEMENTARITY PROBLEMS

Karen M. Thompson

Under the supervision of Professor Olvi L. Mangasarian

ABSTRACT

This thesis is concerned with developing algorithms for solving large–scale versions of the linear complementarity problem (LCP), which consists of finding an n–dimensional vector z such that $z \geq 0$, $Mz+q \geq 0$, and $z(Mz+q) = 0$ where M is an n–by–n real matrix and q is a n–vector. The first algorithm we propose is an asynchronous parallel successive overrelaxation (SOR) algorithm which is suitable for large sparse symmetric problems. The second algorithm is a distributed version of Lemke's classical algorithm for solving the LCP. We design the algorithm for a loosely–coupled network of computers. The third is a serial two–stage successive overrelaxation algorithm for the symmetric positive semidefinite LCP. This algorithm concentrates on updating a certain prescribed subset of variables which is determined by exploiting the complementarity property. We demonstrate that this algorithm successfully solves problems with as many as 10,000 variables

which cannot be tackled by other current algorithms. A fourth hybrid algorithm finds an exact solution for the positive definite symmetric linear complementarity problem in a finite number of steps. In this algorithm we use a successive over-relaxation preprocessing step. This provides us with a partition of the variables into two sets. We then combine a projected Newton step along one set of variables with a projected gradient step along the complement of that set, thereby determining a feasible descent direction. We demonstrate that the preprocessing step provides an effective partition so that in many cases the algorithm will terminate after one additional step. We also demonstrate that the algorithm efficiently solves medium–sized problems as well as large problems with the special property of many zeros in the solution. The final algorithm we propose is a modification of the previous hybrid algorithm so that it can handle positive semidefinite symmetric matrices. Finally, we test the various algorithms on randomly generated test problems to demonstrate their effectiveness.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

ABSTRACT iii

Chapter 1 INTRODUCTION 1

1.1 Large–Scale Programs 1

1.2 Parallel Computing 2

1.3 Linear Complementarity Problem 3

1.4 Proposed Methods 5

1.5 Notation 7

Chapter 2 PARALLEL ASYNCHRONOUS SUCCESSIVE OVERRELAXATION ALGORITHM 9

2.1 Introduction 9

2.2 Optimality Conditions for the LCP 12

2.3 Gradient Projection SOR Algorithm 14

2.4 The Parallel GPSOR 18

2.5 Optimality Conditions 21

2.6 Asynchronous SOR Algorithm 23

2.7 Computational Experience 27

Chapter 3 A PARALLEL PIVOTAL ALGORITHM FOR SOLVING THE LINEAR COMPLEMENTARITY PROBLEM 35

3.1 Introduction 35

3.2 Lemke's Algorithm for solving the LCP 36

3.3 Implementation of Parallel Lemke 40

3.4 Computational Experience 43

Chapter 4 TWO–STAGE SUCCESSIVE OVERRELAXATION ALGORITHM 54

4.1 Introduction 54

4.2 Two–Stage SOR Algorithm 56

4.3 Convergence of TSOR 60

4.4 Implementation and Computational Results 66

Chapter 5 A HYBRID ALGORITHM FOR SOLVING THE LINEAR COMPLEMENTARITY PROBLEM 76

5.1 Introduction 76

5.2 The Hybrid Algorithm 78

5.3 Convergence of the Hybrid Algorithm 81

5.4 Computational Results 88

Chapter 6 A HYBRID METHOD FOR SOLVING
POSITIVE SEMIDEFINITE LCPS 93

6.1 Introduction 93

6.2 Proximal Point Algorithm 95

6.3 The Modified Hybrid Algorithm 96

6.4 Convergence of the Modified Hybrid Algorithm 99

6.5 Computational Results 104

Chapter 7 COMPARATIVE COMPUTATIONAL RESULTS 107

7.1 Introduction 107

7.2 Testing Specifications 108

7.3 Performance Results 109

7.4 Further Research 113

BIBLIOGRAPHY 116

Chapter 1

INTRODUCTION

This thesis is concerned with developing efficient algorithms for solving large–scale linear complementarity problems. We first discuss relevant aspects of this problem.

1.1 LARGE–SCALE PROGRAMS

Currently, industry often uses heuristic methods to estimate solutions of large–scale programs rather than attempting to solve these problems exactly. This is largely due to two inherent difficulties in solving large–scale programs. The first difficulty is that computing time may be prohibitive. Industry must balance the cost of solving the program against the savings that may be obtained by arriving at an optimal solution rather than a sub–optimal solution. Secondly, large–scale programs may have tremendous data storage needs. Therefore effective algorithms for large–scale programs must address these issues.

Much work in large–scale programming has focused on using existing methods for smaller programs and employing sparsity–preserving features. See [Gill & Murray74] for details on adapting Newton–type and quasi–Newton type algorithms for large scale–programming. Often large–scale programs have a preponderance of zeros in the data matrices. Therefore, one way to address both difficulties is to take

advantage of this property. To avoid excessive computation time we eliminate any unnecessary floating point operations such as a multiplication by a zero element. To address storage considerations, we need only store non–zero elements. An appropriate data structure is used to indicate where the zeros lie. Therefore the general procedure is to make sparse factorizations of the data matrices. From here we compute the next iterate. Then we update the factorization and continue. We note that in the absence of special structure the factorizations may fill in. Therefore costly sparsity–preserving techniques rely heavily on special matrix structures.

Some mathematical programming problems can be solved by algorithms which keep initial data intact thereby completely preserving sparsity. These algorithms are ideal for large–scale programs because there is no danger of fill–in since only the original elements of the data matrices are used at a given iteration. Mangasarian [Mangasarian84] has proposed completely sparsity–preserving methods for linear programming and separable quadratic programming problems.

1.2 PARALLEL COMPUTING

Parallel algorithms can provide greater storage capacities as well as speed up computing time and therefore are an important tool for large–scale programming. There are two approaches to developing parallel algorithms.

The first approach is to distribute certain operations in an existing serial optimization algorithm among several processors. For example, several processors

may compute a given function evaluation thereby speeding up the entire computation time. If an algorithm can be largely distributed we may be able to produce an effective distributed version of an existing serial algorithm. The considerations for designing such an algorithm will include finding an efficient way to store data and divide computations evenly in order to avoid communication overhead and idle processor time.

The second approach to developing parallel algorithms must be employed when the original algorithm is serial in nature. In this case, the serial components dominate the algorithm. Therefore a distributed program will provide little speedup and may in fact take longer if the communication overhead exceeds the computational speedup. In this case a new algorithm must be developed. Often this involves breaking up the problem into smaller subproblems solving or partially solving these problems in parallel then invoking a stepsize to find a new iterate for the original problem. Some examples of these types of algorithms can be found in [Chang86],[Chen87],[Feijoo85],[Mangasarian & DeLeone86b],[Medhi87], and [Pang & Yang87]. For a general discussion and a survey of parallel optimization see [Schnabel84].

1.3 LINEAR COMPLEMENTARITY PROBLEM

In this thesis we focus on solving the large–scale linear complementarity problem (LCP). The problem consists of finding an n–dimensional vector z such that

$$z \geq 0$$

$$Mz + q \geq 0$$

$$z(Mz + q) = 0$$

where M is an $n \times n$ real matrix and q is a n–dimensional real vector. We note that when M is positive semidefinite the LCP constitutes a necessary and sufficient condition for solving the problem

$$\min_{z \geq 0} f(z) := \min_{z \geq 0} \frac{1}{2} zMz + qz$$

Linear complementarity problems arise in finite difference schemes for free boundary problems [Cryer71] and economic equilibrium problems [Hogan75]. In addition we can formulate bimatrix games and linear and quadratic programs as LCPs.

The best known method for solving the LCP is due to [Lemke65]. This is a direct method which moves from vertex to adjacent vertex of a polyhedral set until the LCP is solved or until it is determined that the LCP is unsolvable. This method is effective on relatively small–sized LCPs which have special properties such as positive semidefinitenes. Shiau, [Shiau83] proposes an iterative linear programming algorithm which solves LCPs when M is positive semidefinite in a finite number of steps. Iterative methods such as successive overrelaxation have been proposed for large sparse symmetric LCPs. Examples of these algorithms

can be found in [Cryer71] and [Mangasarian77]. These methods take advantage of sparsity and therefore are very effective for large–scale LCPs.

1.4 PROPOSED METHODS

In Chapters 2 and 3 we propose parallel algorithms for solving the linear complementarity problem. In Chapter 2 we present an asynchronous successive overrelaxation algorithm. This is an extension of a gradient projection successive overrelaxation algorithm (GPSOR) proposed by [Mangasarian & De Leone86b]. Here we design an algorithm so as to minimize communication overhead by making the algorithm asynchronous. In Chapter 3 we present a distributed version of Lemke's algorithm for solving the LCP. This algorithm is designed for a network of computers which is a loosely coupled system with relatively high communication costs, such as the CRYSTAL multi–computer, [DeWitt et al84]. Data is stored and activities are divided among processors to minimize communication costs. We demonstrate that for $p << n$ where p is the number of processors and n is the problem dimension, the speedup is nearly p when n is large.

In Chapter 4 we exploit the complementarity property to develop a two–stage SOR algorithm (TSOR) which solves symmetric, positive semidefinite LCPs faster than the ordinary SOR algorithm. The SOR algorithm does not directly exploit the complementary property. Because of this property, if we know a priori which variables remain positive at a solution then solving the LCP reduces to solving a system of equations where the number of equations corresponds to the

number of positive variables at a solution. The idea of TSOR is to take a finite number of SOR iterations as a preprocessing step. From there on we guess which variables will remain positive at a solution. We then obtain a good feasible descent direction by concentrating computational effort on these variables. We find this direction by iteratively solving the corresponding system of equations using the SOR iteration for solving equations and combining this with a gradient direction along the remaining variables. We show that we can solve the equations to any accuracy and still arrive at a feasible descent direction. By providing for the possibility of revising the set of positive variables in the algorithm we establish convergence of this method.

In Chapter 5 we propose a hybrid algorithm for positive definite LCPs based on the ideas in Chapter 4 and work by [Bertsekas82]. As in Chapter 4 we use an SOR preprocessing step and then partition the variables. Here, instead of using the SOR iteration in the second stage we exactly solve the corresponding system of equations. We show that we can identify the solution in a finite number of steps.

In Chapter 6 we modify the algorithm in Chapter 5 to handle positive semidefinite LCPs. We note that in this case the system of equations that must be solved in the hybrid algorithm may be inconsistent. Therefore instead we solve a perturbed system of equations which is guaranteed to have a solution. This provides us with a feasible descent direction once it is combined with a gradient direction along the remaining variables.

Finally, in Chapter 7 we present comparative computational results for the various proposed algorithms based on tests on randomly generated positive semidefinite LCPs.

1.5 NOTATION

All matrices, vectors and scalars are real. The following notations are used throughout this thesis.

1) The n–dimensional real space is denoted by $\mathbb{R}^n$.

2) Given a vector z, we shall denote its j^{th} component by z_j. We say $z \geq 0$ if one has $z_j \geq 0$ for all j. For a given scalar λ, we define $(\lambda)_+ = \max\{0, \lambda\}$. If $z \in \mathbb{R}^n$, we write z_+ for the vector whose j^{th} component is $(z_j)_+$.

3) Superscripts are used to distinguish between vectors, e.g., z^1, z^2.

4) The scalar product of two vectors x, y will be written xy.

5) All matrices are indicated by uppercase letters. The j^{th} row of M is indicated by M_j. The j^{th} column of M is indicated by $M_{.j}$. The transpose of M is indicated by M^T. The symbol I indicates the identity matrix of appropriate dimension while e shall indicate a vector of all ones of appropriate dimension.

6) Real valued functions defined on subsets of $\mathbb{R}^n$ are denoted by f. We write ∇f to indicate the gradient vector, where $\nabla f(z) = (\frac{\partial f(z)}{\partial z_1}, \ldots, \frac{\partial f(z)}{\partial zn})$.

7) For $z \in \mathbb{R}^n$, $\| z \| = (z^T z)^{1/2}$ is the standard Euclidean norm. We also have $\| z \|_M = (zMz)^{1/2}$.

8) We indicate bibliographic references by author's name and year of publication, e.g., [Mangasarian77]. All references are arranged alphabetically and chronologically for each author.

9) The end of the proof of an assertion is indicated by ∎.

Chapter 2

PARALLEL ASYNCHRONOUS SUCCESSIVE OVERRELAXATION ALGORITHM

2.1 INTRODUCTION

In this chapter we propose a parallel asynchronous successive overrelaxation algorithm to solve the linear complementarity problem which we define as follows

$$Mz + q \geq 0, z \geq 0, z(Mz + q) = 0$$

where M is a $n \times n$ symmetric matrix and q is a vector in $\mathbb{R}^n$. The serial SOR iterate is as follows:

$$z^{i+1} = (z^i - \omega E(Mz^i + q + K(z^{i+1} - z^i)))_+$$

where E is a positive diagonal matrix in $\mathbb{R}^{n \times n}$,ω is some positive number, and K is some substitution operator such as the lower or upper triangular part of M.

Related work has been done on parallel iterative methods for solving the system of equations

$$Ax = b$$

Missirlis, [Missirlis85] proposed a Jacobi–type parallel iterative scheme based on finding an approximation of the Neumann series to A^{-1}. Conrad and Wallach, [Conrad & Wallach77], propose a parallel Gauss–Seidel method based on breaking

the matrix A into blocks. They show convergence for diagonally dominant systems. An asynchronous algorithm was proposed by [Barlow & Evans82] for matrices A which have the special structure

$$A = \begin{bmatrix} I & B \\ B & I \end{bmatrix}$$

Previously proposed parallel SOR algorithms for solving the LCP have been described in [Mangasarian & De Leone86a] and [Mangasarian & De Leone86b]. Both methods required synchronization after each update of z. By contrast algorithms proposed here require communications only after asynchronous multiple updates of z. Pang and Yang, [Pang & Yang87], propose a parallel asynchronous algorithm which breaks the matrix M into blocks and makes multiple iterations in parallel. The algorithm proposed here is an asynchronous version of [Mangasarian & De Leone86a].

The significance of developing an asynchronous algorithm lies in the need to develop efficient parallel algorithms. Ideally, the idle time of all processors as well as redundant computing time should be zero. If an algorithm consisted of p independent tasks which required the same amount of computing time we could assign the work to p different processors and complete the work p-times as fast. However, optimization algorithms have serial components, thereby requiring communication between the processors. If communication must occur often then communication costs become a large part of the computation time. Therefore it is

desirable to communicate as little as possible without degrading the convergence properties of the algorithm.

In this chapter we propose a more general version of GPSOR than [Mangasarian & De Leone86b], which can be implemented in an asynchronous way. Like Pang and Yang's work we divide the matrix M into blocks and perform multiple iterations in parallel. However, Pang and Yang's method requires a two stage regular splitting of M which can only be guaranteed for positive definite M. In contrast our method requires positive semidefinite M.

Sections 2.2, 2.3 and 2.4 discuss the previous work of Mangasarian and De Leone found in [Mangasarian & De Leone86b]. Section 2.2 will develop optimality conditions for the Gradient Projection Successive Overrelaxation Algorithm (GPSOR). In Section 2.3 we specify the GPSOR algorithm and present convergence proofs. Section 2.4 develops Mangasarian and De Leone's parallel GPSOR.

Our principal contribution is contained in Sections 2.5, 2.6 and 2.7 where we discuss the proposed Asynchronous SOR (ASOR) algorithm. Section 2.5 develops optimality conditions for the proposed ASOR. In Section 2.6 we specify the algorithm and present convergence results. Finally in Section 2.7 computational results are presented.

2.2 OPTIMALITY CONDITIONS FOR THE LCP

We start by recalling that the LCP is equivalent to finding $z \in \mathbb{R}^n$ such that

$$z = (z - \omega(Mz + q))_+ \text{ for some } \omega > 0 \tag{2.1}$$

The proof of this can be found in [Mangasarian77]. The above relationship suggests an obvious algorithm. Given $z^i \in \mathbb{R}^n$,

$$z^{i+1} = (z^i - \omega(Mz^i + q))_+$$

If M is symmetric this algorithm is a gradient projection algorithm. We can modify this algorithm by updating z component by component and using the most current information to update subsequent components. Therefore the modified algorithm would be

$$p(z) = (z - \omega E(Mz + q + K(p(z) - z))_+ \tag{2.2}$$

where ω is some positive number, E is a positive diagonal matrix in $\mathbb{R}^{n\times n}$ and K is some substitution operator such as the strictly lower or upper triangular part of M. Clearly, a fixed point of this algorithm also solves the LCP.

We will now develop optimality conditions which are thoroughly developed in [Mangasarian & De Leone86b]

Theorem 2.1. *(GPSOR optimality condition) Let $M,K,E \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$, $\omega > 0$ such that E is a positive diagonal matrix.*

(a) If z solves the LCP and $(\omega E)^{-1} + K$ is positive definite then $p(z) = z$ where $p(z)$ is a solution of (2.2).

(b) If $p(z)$ solves (2.2) and $p(z) = z$ then z solves the LCP.

Proof :

See [Mangasarian & De Leone86b] ∎

The following result shows that given proper choices of K and E we can develop an optimality function $\theta(z)$.

Theorem 2.2. *(Continuous optimality function for an LCP) Let* $\omega > 0$ *and* $E \in \mathbb{R}^{n \times n}$ *be a positive diagonal matrix and let* K *be a strictly lower or upper triangular matrix, or let* $(\omega E)^{-1} + K$ *be a* P*-matrix. Then*

$$\theta(z) := \| p(z) - z \| \tag{2.3}$$

is a nonnegative continuous function on $\mathbb{R}^n$ *which vanishes only on solution points of the LCP.*

Proof :

Nonnegativity follows from the nonnegativity of a norm. The function $\theta(z)$ vanishes only on solution points of the LCP as a consequence of Theorem 2.1. Finally continuity follows for K strictly lower (upper) triangular, because $p(z)$ can be computed componentwise $p_j(z), j = 1, \ldots, n (j = n, \ldots, 1)$ as a unique piecewise linear, and hence continuous function of z. For the case when $(\omega E)^{-1} + K$ is a P-matrix, $p(z)$ exists and is unique [Murty83], and by Theorem 3.3 of [Mangasarian & Shiau86] is Lipschitz continuous and hence continuous in z. ∎

2.3 GRADIENT PROJECTION SOR ALGORITHM

We now make the blanket assumption that M is symmetric. Therefore the LCP can be considered as necessary optimality conditions for

$$\min_{z \geq 0} f(z) := \frac{1}{2} zMz + qz \tag{2.4}$$

The following is the specification of the serial GPSOR algorithm due to Mangasarian and De Leone.

Algorithm 2.1.

Direction Generation: Define the direction

$$d^i := p(z^i) - z^i \tag{2.5}$$

such that

$$p(z^i) = (z^i - \omega E(Mz^i + q + K(p(z^i) - z^i)))+ \tag{2.6}$$

where E is a positive diagonal matrix, $\omega > 0$ and for some $\gamma > 0$

$$z((\omega E)^{-1} + K)z \geq \gamma \parallel z \parallel^2 \quad \forall z \in \mathbb{R}^n \tag{2.7}$$

Stop if $d^i = 0$, else continue.

Stepsize Generation

$$z^{i+1} = z^i + \lambda d^i$$

$$f(z^i + \lambda^i d^i) = \min_{\lambda}\{f(z^i + \lambda d^i) \mid z^i + \lambda d^i \geq 0\} \qquad (2.8)$$

Note that because f is quadratic (2.8) is easy to solve. Now we can use the optimality function to prove convergence results for the algorithm. The following two theorems are convergence results for Algorithm 2.1 found in [Mangasarian & De Leone86b]. We repeat them here for completeness.

Theorem 2.3. *(Serial GPSOR convergence) Let M be symmetric. Either the sequence $\{z^i\}$ generated by algorithm 2.1 terminates at a solution of the LCP or each accumulation point of $\{z^i\}$ solves the LCP.*

Proof :

The sequence $\{z^i\}$ terminates only if for some i, $p(z^i) = z^i$, in which case by Theorem 2.1, z^i solves the LCP. Suppose now $\{z^i\}$ does not terminate and that $\bar{z}$ is an accumulation point of $\{z^i\}$. Let $p^i := p(z^i)$. We then have for $i = 0, 1, \ldots$ that

$$\begin{aligned} -\nabla f(z^i)d^i &= -\nabla f(z^i)(p^i - z^i) \\ &= [z^i - \omega E(Mz^i + q + K(p^i - z^i)) - p^i](\omega E)^{-1}[p^i - z^i] \\ &\quad + \| p^i - z^i \|^2_{(\omega E)^{-1}+K} \end{aligned}$$

$$\geq \| p^i - z^i \|^2_{(\omega E)^{-1}+K} \quad \text{(By Lemma 2.2 [Mangasarian77])}$$

$$\geq \gamma \| p^i - z^i \|^2 \quad \text{(By (2.7))}$$

$$= \gamma\theta(z^i)^2$$

Hence

$$-\nabla f(z^i)d^i \geq \gamma\theta(z^i)^2 \geq 0, i = 0, 1, \ldots \tag{2.9}$$

Now let $\{z^{i_j}\}$ be a subsequence converging to the accumulation point $\bar{z}$. We claim that $p^{i_j} := p(z^i_j)$ is bounded. For if it were not bounded it would follow from (2.6) that

$$\lim_{j\to\infty} \frac{p(z^{i_j})}{\| p(z^{i_j}) \|} = \lim_{j\to\infty} \Big(-\omega EK \frac{p(z^{i_j})}{\| p(z^{i_j}) \|}\Big)_+$$

and hence for some accumulation point $\bar{p}$ we would have

$$\bar{p} = (-\omega EK\bar{p})_+, \ \bar{p} \neq 0. \tag{2.10}$$

This however is equivalent to $0 \neq \bar{p} \geq 0, \bar{p}((\omega\bar{E})^{-1} + K)\bar{p} = 0, ((\omega E)^{-1} + K)\bar{p} \geq 0$ since $a = b_+ \iff a - b \geq 0, a \geq 0, a(a - b) = 0$. But this contradicts (2.7). Consequently, without loss of generality, let $\{z^{i_j}, p^{i_j}\} \to (\bar{z}, \bar{p})$ and hence$\{d^{i_j}\} \to \bar{d} = \bar{p} - \bar{z}$. Now, since $p^{i_j} \geq 0$ it follows that

$$z^{i_j} + \lambda d^{i_j} = (1 - \lambda)z^{i_j} + \lambda p^i_j \geq 0 \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Consequently

$$f(\bar{z} + \lambda\bar{d}) \geq f(z^{i_j+1}) \geq f(z^{i_j+1}) \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Letting $j \to \infty$ we have that

$$f(\bar{z} + \lambda\bar{d}) \geq f(\bar{z}) \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Hence $\nabla f(\bar{z})\bar{d} \geq 0$. Combining this with (2.9) gives

$$0 \geq -\nabla f(\bar{z})\bar{d} = \lim_{j\to\infty} -\nabla f(z^{i}j)d^{i}j \geq \lim_{j\to\infty} \gamma\theta(z^{i}j)^2 \geq 0$$

Hence

$$\begin{aligned} 0 &= \lim_{j\to\infty} \theta(z^{i}j) = \theta(\bar{z}) && \text{(By Theorem 2.2)} \\ &= \| \ (\bar{z} - \omega E(M\bar{z} + q + K(\bar{p} - \bar{z})))_+ - \bar{z} \ \| && \text{(By (2.3))} \end{aligned}$$

By (2.7), $((\omega E)^{-1} + K)$ is positive definite, and hence by Theorem 2.1, $\bar{z}$ solves the LCP. ∎

When K and E are chosen appropriately we have special cases described in the next corollary.

Corollary 2.4. *(GPSOR special cases) Condition (2.7) of Algorithm 2.1 holds under either of the following two assumptions:*

(i) $K = L$ or U, and $0 < \omega < \min_{j=1,\ldots,n} 2/E_{jj} \sum_{\substack{l=1 \\ l\neq j}}^{n} | M_{jl} |$

(ii) $K = L$ or U, $E = D^{-1} > 0$, M is positive semidefinite and $0 < \omega < 2$

where

$$L + D + U := M, \tag{2.10}$$

and L is the strictly lower triangular part of M, U is the strictly upper triangular part of M and D is the diagonal of M.

Proof :

See [Mangasarian & De Leone86b] ∎

2.4 THE PARALLEL GPSOR

If we choose $K = L$ as in Corollary 2.4 we see that z_j^{i+1} replaces z_j^i during the computation of z_l^{i+1} for all $l > j$. This selection of K produces an algorithm which is sequential in nature. The idea Mangasarian and De Leone use to develop a parallel GPSOR is to find an appropriate K which will allow the algorithm to be split into parallel parts. Therefore K is chosen to be a block diagonal matrix consisting of the strictly lower triangular part of the principal submatrix of each horizontal partition of the matrix M. Then the new algorithm can be easily distributed on to a number of processors equal to the number of submatrices in the horizontal partition of M. This choice of K was first proposed in [Mangasarian & De Leone86a] for a different SOR procedure which didn't involve a stepsize. Therefore, to get convergence an ω restricted to an interval depending on E and K must be chosen. In practice this resulted in small ω which resulted in a slow algorithm. GPSOR allows a choice of ω in the range (0,2) when M is positive semidefinite. To specify parallel GPSOR, partition the matrix M into p contiguous horizontal blocks as follows:

$$M := \begin{bmatrix} M_{\mathcal{I}_1} \\ M_{\mathcal{I}_2} \\ \vdots \\ M_{\mathcal{I}_p} \end{bmatrix}$$

where the blocks $M_{\mathcal{I}_j}$ correspond to the variables $z_{\mathcal{I}_j}$ and $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_p\}$ is a consecutive partition of $\{1, 2, \ldots, n\}$. Now partition $M_{\mathcal{I}_j}$ as follows:

$$M_{\mathcal{I}_j} := [M_{\mathcal{I}_j \mathcal{I}_j} \quad M_{\mathcal{I}_j \mathcal{I}_j^c}]$$

where $\mathcal{I}_j^c$ is the complement of $\mathcal{I}_j$ in $\{1, 2, \ldots, p\}$. Therefore, $M_{\mathcal{I}_j \mathcal{I}_j}$ is a principal square submatrix of M. Now

$$M_{\mathcal{I}_j \mathcal{I}_j} := L_{\mathcal{I}_j \mathcal{I}_j} + D_{\mathcal{I}_j \mathcal{I}_j} + U_{\mathcal{I}_j \mathcal{I}_j}$$

where $L_{\mathcal{I}_j \mathcal{I}_j}$ and $U_{\mathcal{I}_j \mathcal{I}_j}$ are the strictly lower and upper triangular part of $M_{\mathcal{I}_j \mathcal{I}_j}$, and $D_{\mathcal{I}_j \mathcal{I}_j}$ is the diagonal part of $M_{\mathcal{I}_j \mathcal{I}_j}$.

Now define K as follows:

$$K := \begin{bmatrix} L_{\mathcal{I}_1 \mathcal{I}_1} & & & \\ & L_{\mathcal{I}_2 \mathcal{I}_2} & & \\ & & \ddots & \\ & & & L_{\mathcal{I}_p \mathcal{I}_p} \end{bmatrix} \tag{2.11}$$

Algorithm 2.1 can now be performed for each row block $\mathcal{I}_j$ for $j = 1, \ldots, p$ simultaneously. We are now ready to specify the algorithm.

Algorithm 2.2 Parallel GPSOR Algorithm for the LCP

Let $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_p\}$ be a consecutive partition of $\{1, 2, \ldots, n\}$,let E be a positive diagonal matrix in $\mathbb{R}^{n \times n}$ and let $z^0 \geq 0$. For $i = 0, 1, 2, \ldots$ do the following

Direction Generation Define the direction

$$d^i := p(z^i) - z^i := \begin{pmatrix} p_{\mathcal{I}_1}(z^i) - z^i_{\mathcal{I}_1} \\ \vdots \\ p_{\mathcal{I}_p}(z^i) - z^i_{\mathcal{I}_p} \end{pmatrix} \tag{2.12}$$

such that $p(z^i)$ satisfies

$$\begin{aligned} p_{\mathcal{I}_j}(z^i) = (z_{\mathcal{I}_j} - \omega E_{\mathcal{I}_j\mathcal{I}_j}(M_{\mathcal{I}_j} z^i \\ + q_{\mathcal{I}_j} + L_{\mathcal{I}_j\mathcal{I}_j}(z^{i+1}_{\mathcal{I}_j} - z^i_{\mathcal{I}_j})))_+ \end{aligned} \tag{2.13}$$

for $j = 1, \ldots p$ where $\omega > 0$ is chosen so that for some $\gamma > 0$

$$z_{\mathcal{I}_j}((\omega E_{\mathcal{I}_j})^{-1} + L_{\mathcal{I}_j\mathcal{I}_j}) z_{\mathcal{I}_j} \geq \gamma \parallel z_{\mathcal{I}_j} \parallel^2 \quad \forall z_{\mathcal{I}_j} \tag{2.14}$$

for $j = 1, \ldots, p$

Stop if $d^i = 0$, else continue.

Stepsize Generation

$$z^{i+1} = z^i + \lambda^i d^i$$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda} \{ f(z^i + \lambda d^i) | z^i + \lambda d^i \geq 0 \} \tag{2.15}$$

The direction generation can be performed in parallel on p processors. The stepsize generation is performed and the new value of z^{i+1} is shared between p processors.

The following results are from [Mangasarian & De Leone86b].

Theorem 2.5. *(Convergence of the Parallel GPSOR Algorithm) Either the sequence* $\{z^i\}$ *generated by the Parallel GPSOR Algorithm 2.2 terminates at a solution of the LCP or each of its accumulation points solves the LCP.*

Corollary 2.6. *Condition (2.14) of algorithm 2.2 hold with either of the following two assumptions:*

(i) $0 < \omega < \min\limits_{j=1,\ldots,p} \min\limits_{i \in \mathcal{I}_j} 2/E_{ii} \sum\limits_{\substack{l \in \mathcal{I}_j \\ l \neq i}} | M_{il} |$

(ii) $0 < \omega < 2, E = D^{-1}$ and M is positive semidefinite.

2.5 OPTIMALITY CONDITIONS

Parallel GPSOR must be synchronized after every iteration. On a loosely coupled network this would involve communicating information after every iteration. Because the SOR iteration is relatively cheap communication costs may be a large proportion of the total time, thus causing the algorithm to be inefficient. Therefore, it would be advantageous to construct an algorithm that is asynchronous.

The proposed asynchronous algorithm is a multi-sweep GPSOR. The idea of the algorithm is that each processor makes multiple updates of $z_{\mathcal{I}_j}$ before sharing information. Assuming K is defined as in (2.11) we have

$$p(z) := \begin{pmatrix} p_{\mathcal{I}_1}^{k_1}(z) \\ \vdots \\ p_{\mathcal{I}_p}^{k_p}(z) \end{pmatrix}$$

where

$$p^{k_j}_{\mathcal{I}_j}(z) = \Big(p^{k_j-1}_{\mathcal{I}_j}(z) - \omega E_{\mathcal{I}_j\mathcal{I}_j}\big(M_{\mathcal{I}_j}p^{k_j-1}_{\mathcal{I}_j}(z) + q_{\mathcal{I}_j} + L_{\mathcal{I}_j\mathcal{I}_j}(p^{k_j}_{\mathcal{I}_j}(z) - p^{k_j-1}_{\mathcal{I}_j}(z))\big)\Big)_+ \tag{2.16}$$

We now develop optimality conditions for multi-sweep GPSOR.

Theorem 2.7. *(Optimality Conditions for Asynchronous SOR)*

Let $M, K, E \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$, $\omega > 0$ such that E is a positive diagonal matrix.

(a) (Necessity)If z solves the LCP and $(\omega E)^{-1} + K$ is positive definite, then $p(z)=z$.

(b) (Sufficiency) If $p(z) = z$, $(\omega E_{\mathcal{I}_j})^{-1} + L_{\mathcal{I}_j\mathcal{I}_j}$ is positive definite and $(\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} + L_{\mathcal{I}_j\mathcal{I}_j} - \frac{M_{\mathcal{I}_j\mathcal{I}_j}}{2}$ is positive definite $\forall j$, then z solves the LCP.

Proof :

Part (a) follows from the proof of part (a) in Theorem 2.1. We now prove part (b).

Suppose $p^{k_j}_{\mathcal{I}_j}(z) = z_{\mathcal{I}_j}\ \forall j$. If $k_j = 1\ \forall j$, then the result follows from part b of Theorem 2.1 because this is equivalent to communicating after each update. Therefore we will show that if $p^{k_j}_{\mathcal{I}_j}(z) = z_{\mathcal{I}_j}$ then $p^1_{\mathcal{I}_j} = z_{\mathcal{I}_j}$ in which case the result follows from Theorem 2.1.

Suppose that $z_{\mathcal{I}_j} \neq p^1_{\mathcal{I}_j}(z)$ but $z_{\mathcal{I}_j} = p^{k_j}_{\mathcal{I}_j}(z)$. Then $z_{\mathcal{I}_j}$ is an accumulation point for the following iterate:

$$z^{i+1}_{\mathcal{I}_j} = (z^i_{\mathcal{I}_j} - \omega E_{\mathcal{I}_j\mathcal{I}_j}(M_{\mathcal{I}_j\mathcal{I}_j}z^i_{\mathcal{I}_j} + q_{\mathcal{I}_j} + M_{\mathcal{I}_j\mathcal{I}_j^c}z_{\mathcal{I}_j^c} + L_{\mathcal{I}_j\mathcal{I}_j}(z^{i+1}_{\mathcal{I}_j} - z^i_{\mathcal{I}_j})))+$$

where $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_p\}$ is a consecutive partition of $\{1, 2, \ldots, n\}$ and $\mathcal{I}_j^c$ is the complement of $\mathcal{I}_j$ in $\{1, 2, \ldots, p\}$. Then by Theorem 2.1 of [Mangasarian77] we have that $z_{\mathcal{I}_j}$ solves the following LCP:

$$w_{\mathcal{I}_j} = M_{\mathcal{I}_j\mathcal{I}_j} z_{\mathcal{I}_j} + q_{\mathcal{I}_j} + M_{\mathcal{I}_j\mathcal{I}_j^c} z_{\mathcal{I}_j^c} \geq 0$$

$$w_{\mathcal{I}_j} \times z_{\mathcal{I}_j} = 0$$

$$z_{\mathcal{I}_j} \geq 0$$

since $(\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} + L_{\mathcal{I}_j\mathcal{I}_j} - \frac{M_{\mathcal{I}_j\mathcal{I}_j}}{2}$ is positive definite. But by Theorem 2.1 part a and $(\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} + K_{\mathcal{I}_j\mathcal{I}_j}$ positive definite this contradicts the fact that $z_{\mathcal{I}_j} \neq p^1_{\mathcal{I}_j}(z)$. Therefore from Theorem 2.1(b) z solves the LCP. ∎

2.6 ASYNCHRONOUS SOR ALGORITHM

We are now prepared to specify the algorithm.

Algorithm 2.3.

Direction Generation Define the direction

$$d^i := p(z^i) - z^i$$

Let $z^i := [z^i_{\mathcal{I}_j} \; z^i_{\mathcal{I}_j^c}]$. Define $p^0_{\mathcal{I}_j}(z^i) := z^i_{\mathcal{I}_j}$ Then $p^{k_j}_{\mathcal{I}_j}$ is defined as in (2.16). Choose $E_{\mathcal{I}_j\mathcal{I}_j}$ and $\omega > 0$ such that for some $\gamma_1, \gamma_2 > 0$

$$y(L_{\mathcal{I}_j\mathcal{I}_j} + (\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} - \frac{M_{\mathcal{I}_j\mathcal{I}_j}}{2})y > \gamma_1 \parallel y \parallel^2 \quad \forall y \qquad (2.17)$$

$$y(L_{\mathcal{I}_j\mathcal{I}_j} + (\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1})y > \gamma_2 \parallel y \parallel^2 \quad \forall y \tag{2.18}$$

stop if $d^i = 0$ else continue.

Step Generation

$$z^{i+1} = z^i + \lambda^i d^i$$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda}\{f(z^i + \lambda d^i) | z^i + \lambda d^i \geq 0\}$$

We can now use the optimality results to establish convergence for Algorithm 2.3.

Theorem 2.8. *(ASOR Convergence) Let M be symmetric and positive semidefinite. Either the sequence $\{z^i\}$ generated by Algorithm 2.3 terminates at a solution of the LCP or each accumulation point of $\{z^i\}$ solves the LCP.*

Proof :

The sequence terminates only if for some $i, p(z^i) = z^i$ in which case z^i solves the LCP. Suppose $\{z^i\}$ does not terminate and $\bar{z}$ is an accumulation point of $\{z^i\}$.

$$\begin{aligned} -\nabla f(z^i)d^i &= -\nabla f(z^i)(p^i(z^i) - z^i) \\ &= -\nabla_{\mathcal{I}_1} f(z^i)(p^{k_1}_{\mathcal{I}_1}(z^i) - z^i_{\mathcal{I}_1}) - \ldots - \nabla_{\mathcal{I}_p} f(z^i)(p^{k_p}_{\mathcal{I}_p}(z^i) - z_{\mathcal{I}_p}) \end{aligned}$$

Therefore define

$$f^i_{\mathcal{I}_j}(z_{\mathcal{I}_j}) = f(z^i_{\mathcal{I}_1}, z^i_{\mathcal{I}_2}, \ldots, z_{\mathcal{I}_j}, \ldots, z^i_{\mathcal{I}_j})$$

For ease of notation we will drop the superscript i on $f^i_{\mathcal{I}_j}(z_{\mathcal{I}_j})$. Then

$$
\begin{aligned}
-\nabla_{\mathcal{I}_j} f(z^i)(p^{k_j}_{\mathcal{I}_j} - z^i_{\mathcal{I}_j}) &= f_{\mathcal{I}_j}(z^i_{\mathcal{I}_j}) - f_{\mathcal{I}_j}(p^{k_j}_{\mathcal{I}_j}(z_{\mathcal{I}_j})) \\
&\quad + \frac{1}{2} \parallel p^{k_j}_{\mathcal{I}_j}(z_{\mathcal{I}_j}) - z^i_{\mathcal{I}_j} \parallel^2_{M_{\mathcal{I}_j\mathcal{I}_j}} \\
&\geq f_{\mathcal{I}_j}(z^i_{\mathcal{I}_j}) - f_{\mathcal{I}_j}(p^1_{\mathcal{I}_j}(z_{\mathcal{I}_j})) \\
&\quad + \frac{1}{2} \parallel p^{k_j}_{\mathcal{I}_j}(z_{\mathcal{I}_j}) - z^i_{\mathcal{I}_j} \parallel^2_{M_{\mathcal{I}_j\mathcal{I}_j}} \\
&\qquad \text{(By (2.17) and Theorem 2.1 [Mangasarian77])} \\
&\geq \parallel p^1_{\mathcal{I}_j}(z^i_{\mathcal{I}_j}) - z^i_{\mathcal{I}_j} \parallel^2_{L_{\mathcal{I}_j\mathcal{I}_j} + (\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} - M_{\mathcal{I}_j\mathcal{I}_j}/2} \\
&\quad + \frac{1}{2} \parallel p^{k_j}_{\mathcal{I}_j}(z_{\mathcal{I}_j}) - z^i_{\mathcal{I}_j} \parallel^2_{M_{\mathcal{I}_j\mathcal{I}_j}} \\
&\qquad \text{(By Theorem 2.1 [Mangasarian77])} \\
&\geq \gamma_1 \parallel p^1_{\mathcal{I}_j}(z^i_{\mathcal{I}_j}) - z^i_{\mathcal{I}_j} \parallel^2 + \frac{1}{2} \parallel p^{k_j}_{\mathcal{I}_j}(z_{\mathcal{I}_j}) - z^i_{\mathcal{I}_j} \parallel^2_{M_{\mathcal{I}_j\mathcal{I}_J}} \\
&\qquad \text{(By (2.17))} \\
&\geq \gamma_1 \theta_{\mathcal{I}_j}(z_{\mathcal{I}_j})^2 \quad \text{(By positive semidefiniteness of } M\text{)}
\end{aligned}
$$

Hence,

$$-\nabla f(z_i) d^i \geq \gamma(\theta_{\mathcal{I}_1}(z^i_{\mathcal{I}_1})^2 + \ldots \theta_{\mathcal{I}_p}(z^i_{\mathcal{I}_p})^2) = \gamma\theta(z^i)^2 \geq 0$$

Now let $\{z^{ij}\}$ be a subsequence converging to $\bar{z}$. We claim that $p(z^{ij})$ is bounded. Suppose $p(z^{ij})$ is not bounded and suppose

$$p(z^{ij}) = \begin{pmatrix} p^1_{\mathcal{I}_1}(z^{ij}) \\ p^1_{\mathcal{I}_2}(z^{ij}) \\ \vdots \\ p^1_{\mathcal{I}_p}(z^{ij}) \end{pmatrix}$$

Then

$$\lim_{j\to\infty} \frac{p^1_{\mathcal{I}_k}(z^{ij})}{\| p^1_{\mathcal{I}_k}(z^{ij}) \|} = \lim_{j\to\infty} \Big(-\omega E_{\mathcal{I}_k\mathcal{I}_k} L_{\mathcal{I}_k\mathcal{I}_k} \frac{p^1_{\mathcal{I}_k}(z^{ij})}{\| p^1_{\mathcal{I}_k}(z^{ij}) \|}\Big)_+$$

and hence for an accumulation point $\bar{p}$ we have

$$\bar{p} = (-\omega E_{\mathcal{I}_k\mathcal{I}_k} L_{\mathcal{I}_k\mathcal{I}_k} \bar{p})_+ \quad \bar{p} \neq 0$$

This however is equivalent to

$$0 \neq \bar{p} \geq 0, \bar{p}((\omega E_{\mathcal{I}_k\mathcal{I}_k})^{-1} + L_{\mathcal{I}_k\mathcal{I}_k})\bar{p} = 0$$

$$((\omega E_{\mathcal{I}_k\mathcal{I}_k})^{-1} + K_{\mathcal{I}_k\mathcal{I}_k})\bar{p} \geq 0$$

But this contradicts (2.18). By induction on l for $p^l_{\mathcal{I}_k}(z^{ij})$ we see that $p(z^{ij})$ is bounded. Without loss of generality let $\{z^{ij}, p(z^{ij})\} \to \{\bar{z}, \bar{p}\}$ and hence $\bar{d} = \bar{p} - \bar{z}$. Since $p(z^{ij}) \geq 0$

$$z^{ij} + \lambda d^{ij} = (1-\lambda)z^{ij} + \lambda p(z^{ij}) \geq 0 \quad \text{for} \quad 0 \leq \lambda \leq 1$$

Consequently

$$f(z^{ij} + \lambda d^{ij}) \geq f(z^{ij+1}) \geq f(z^{ij+1}) \quad \text{for} \quad 0 \leq \lambda \leq 1$$

and letting $j \to \infty$ we get

$$f(\bar{z} + \lambda\bar{d}) \geq f(\bar{z}) \quad 0 \leq \lambda \leq 1$$

Hence $\nabla f(\bar{z})\bar{d} \geq 0$ and

$$0 \geq -\nabla f(\bar{z})\bar{d} = \lim_{j\to\infty} -\nabla f(z^{ij}) d^{ij} \geq \lim_{j\to\infty} \gamma\theta(z^{ij})^2 \geq 0$$

Therefore $\theta(\bar{z}) = 0$. Then by Theorem 2.7 and conditions (2.17) and (2.18) $\bar{z}$ solves the LCP. ∎

When E is chosen appropriately we have the following special case.

Corollary 2.9. *(Special Case) Conditions (2.17) and (2.18) of Algorithm 2.3 hold when M is positive semidefinite, $E_{\mathcal{I}_j\mathcal{I}_j} = D^{-1}_{\mathcal{I}_j\mathcal{I}_j} > 0$ and $0 < \omega < 2$.*

Proof :

(i)

$$z((\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} + L_{\mathcal{I}_j\mathcal{I}_j} - \frac{M_{\mathcal{I}_j\mathcal{I}_j}}{2})z =$$

$$= z(\omega^{-1}D_{\mathcal{I}_j\mathcal{I}_j} + L_{\mathcal{I}_j\mathcal{I}_j} - \frac{L_{\mathcal{I}_j\mathcal{I}_j} + U_{\mathcal{I}_j\mathcal{I}_j} + D_{\mathcal{I}_j\mathcal{I}_j}}{2})z$$

$$= \frac{1}{2}z((2\omega^{-1} - 1)D_{\mathcal{I}_j\mathcal{I}_j})z \geq \gamma \parallel z \parallel^2$$

where the last inequality follows form the positivity of $D_{\mathcal{I}_j\mathcal{I}_j}$ and $0 < \omega < 2$.

(ii)

$$z((\omega E_{\mathcal{I}_j\mathcal{I}_j})^{-1} + L_{\mathcal{I}_j\mathcal{I}_j})z = z(\omega^{-1}D_{\mathcal{I}_j\mathcal{I}_j} + \frac{L_{\mathcal{I}_j\mathcal{I}_j} + U_{\mathcal{I}_j\mathcal{I}_j}}{2})z$$

$$= \frac{1}{2}z((2\omega^{-1} - 1)D_{\mathcal{I}_j\mathcal{I}_j} + M_{\mathcal{I}_j\mathcal{I}_j}) \geq \gamma \parallel z \parallel^2$$

where the last inequality follows from the positive semidefiniteness of M, the positivity of D and $0 < \omega < 2$. ∎

2.7 Computational Experience

The ASOR algorithm was tested on a symmetric LCP formulation of a linear programming problem. We state the LP in standard form.

$$\min \quad cx$$

$$\text{st.}$$

$$Ax \geq b$$

$$x \geq 0$$

where $c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}$and $b \in \mathbb{R}^m$. The LCP formulation due to Mangasarian [Mangasarian84] is based on the following observation. Consider the following quadratic programming problem

$$\min \quad cx + \frac{\epsilon}{2} \parallel x \parallel^2$$

st.

$$Ax \geq b$$

$$x \geq 0$$

The unique solution of the above quadratic program is the least 2–norm solution of the LP provided that $\epsilon \in (0, \bar{\epsilon}]$ for some $\bar{\epsilon} > 0$. We take the dual of the quadratic program and get the following dual program

$$\max_{(u,v) \geq 0} \quad cx + \frac{\epsilon}{2} \parallel x \parallel^2 + u(b - Ax) - vx$$

st.

$$c + \epsilon x - A^T u - v = 0$$

where u and v are dual variables in $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively. After making appropriate substitutions we get the following minimization of a quadratic function over nonnegativity constraints

$$\min_{(u,v) \geq 0} \quad \frac{1}{2\epsilon} \parallel A^T u + v - c \parallel^2 - bu$$

Then the LCP formulation is an expression of the Karush–Kuhn–Tucker conditions. The algorithm was tested on the CRYSTAL multi–computer. CRYSTAL is

a set of 20 VAX – 11/750 computers with 2 megabytes of memory each connected by an 80 megabit/sec Proteon ProNet token ring. We randomly generate an LP which is known to be solvable. We choose $\epsilon = .2$ and ω varies between 1.2 and 1.8. Primal feasibilities were calculated to an accuracy of 10^{-5}. The primal objective function values were accurate to four significant digits.

An important question for implementing ASOR is how many updates of ${}^{z}I_j$ should be carried out in processor j before sharing information. Pang and Yang, [Pang & Yang87], suggest using a progressive tolerance. Therefore we iterate in parallel until the difference between successive iterates is less than a certain tolerance based on a progressive accuracy strategy. Pang and Yang report success with implementations on the IBM 4381 and the CRAY X-MP/24. However, they compare their method with an earlier version of a parallel SOR algorithm, [Mangasarian & De Leone86a], which is not as effective as GPSOR.

In our experience we have noted that it is not always wise to select $k_j > 1$. We can think of the parallel sweeps as being part of an inner iteration while that step plus the final line search is an outer iteration. Therefore the more sweeps we do in the inner iteration the more time consuming one outer iteration is. If the marginal improvement due to extra inner iterations is very small it may not be worthwhile to choose $k_j > 1$. Our experience has been that near the optimal point one should use only one sweep.

We incorporated the following strategy. A master node is in charge of determining the number of sweeps at a given iteration. If

$$\frac{\| u^{i+1} - u^i \|}{\| u^i - u^{i-1} \|} > r$$

use one sweep. Otherwise, use two sweeps. In our computations we use $r = .25$. We note that since we were able to balance the load evenly among the processors there was no need to have each node carry out a different number of sweeps. However, this can be changed in case of unbalanced distributions.

Computational results are presented for LPs of size 125 x 500, 250 x 1000, and 1000 x 4000. The two smaller problems have density 5% while the 1000 x 4000 problem has density .8%. Figures 2.1, 2.2, and 2.3 graph time against number of processors. Each graph represents an average over three random problems. We note that in these cases ASOR tends to provide slight improvements over GPSOR. If this algorithm was implemented on a parallel system with a slower communication medium such as a network of workstations these differences may become more pronounced. However, on a shared memory processor the savings of ASOR diminish.

In figures 2.2 and 2.3 we note that two processors solve the problem more than twice as fast as one processor. This is due to the fact that the substitution operator K changes as we move to more processors. Therefore we are running a different algorithm as opposed to simply distributing the program. In these cases there was a drop in total iterations which accounts for the extra speedup. This

leads us to a certain side benefit of studying parallel algorithms. That is we can discover faster serial algorithms! In other words if we use the substitution operator K used in the two processor case but run it serially we should produce a faster serial algorithm. This phenomenon was originally discussed in [Mangasarian & De Leone86b].

Finally we note that further research may provide a more sophisticated criterion to determine how many multiple sweeps should be computed at a given iteration. Such a criterion may provide substantial improvement in the algorithm.

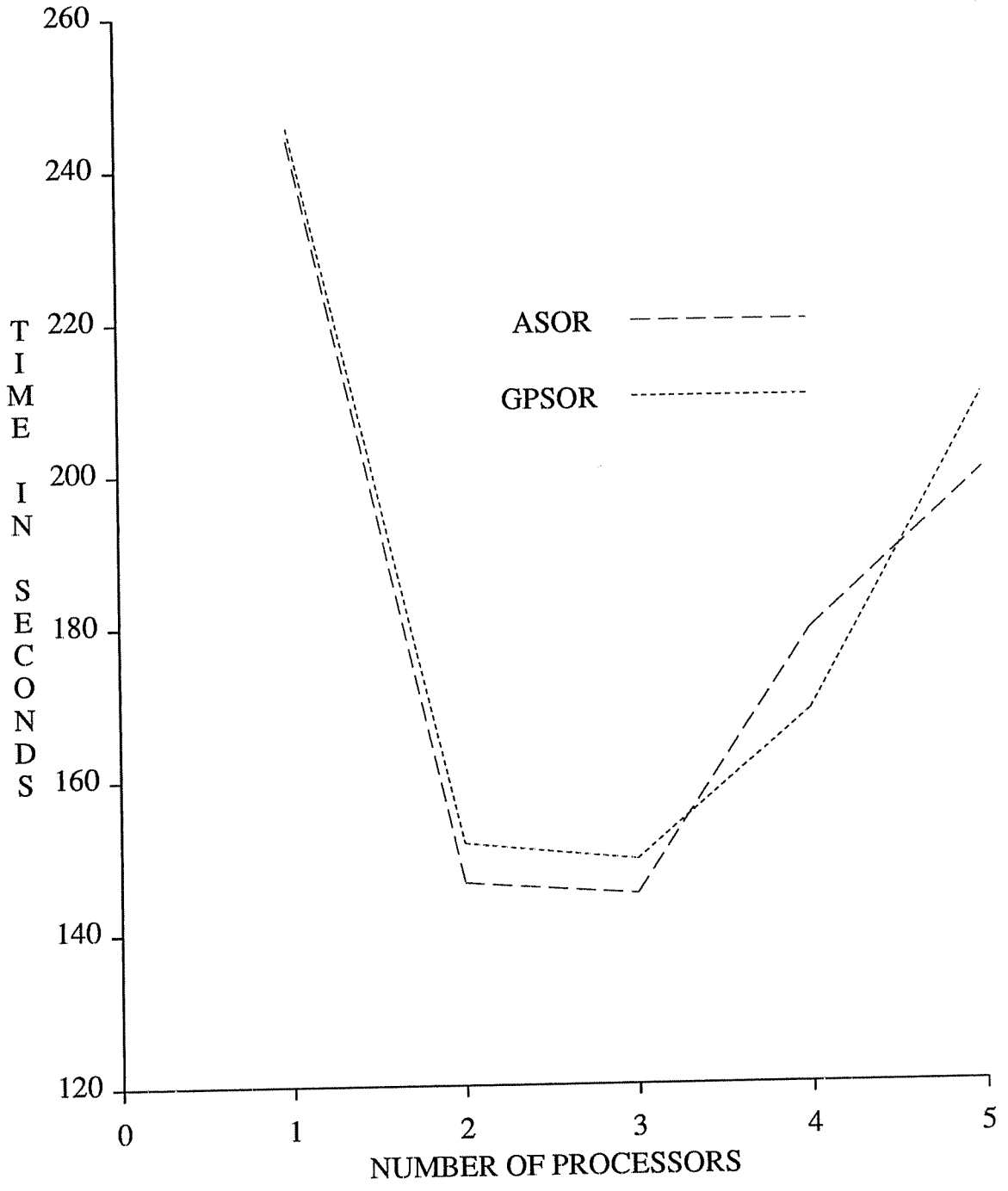

Figure 2.1 Total Time (density=5%, m=125, n=500)

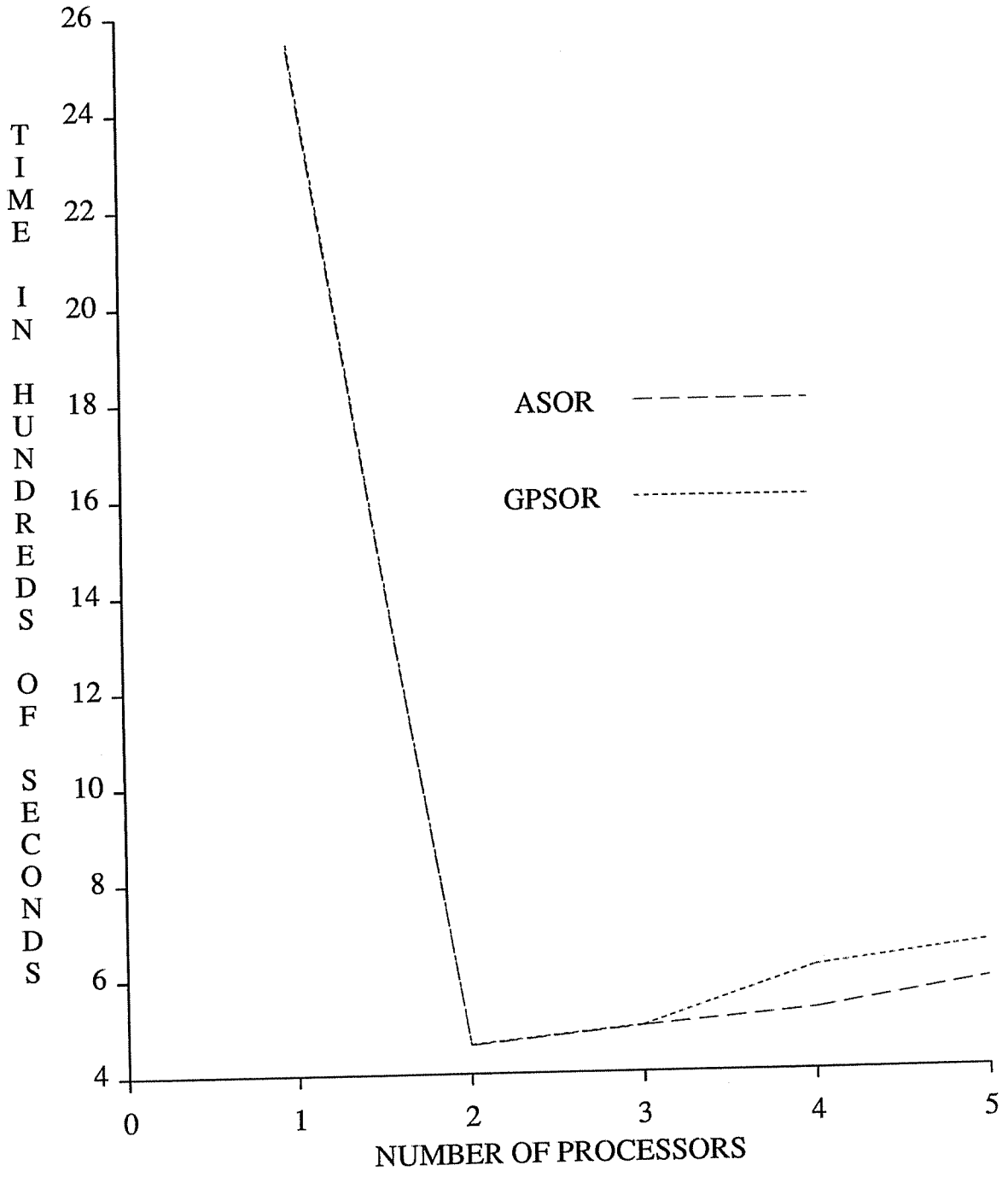

Figure 2.2 Total Time (density=5%, m=250, n=1000)

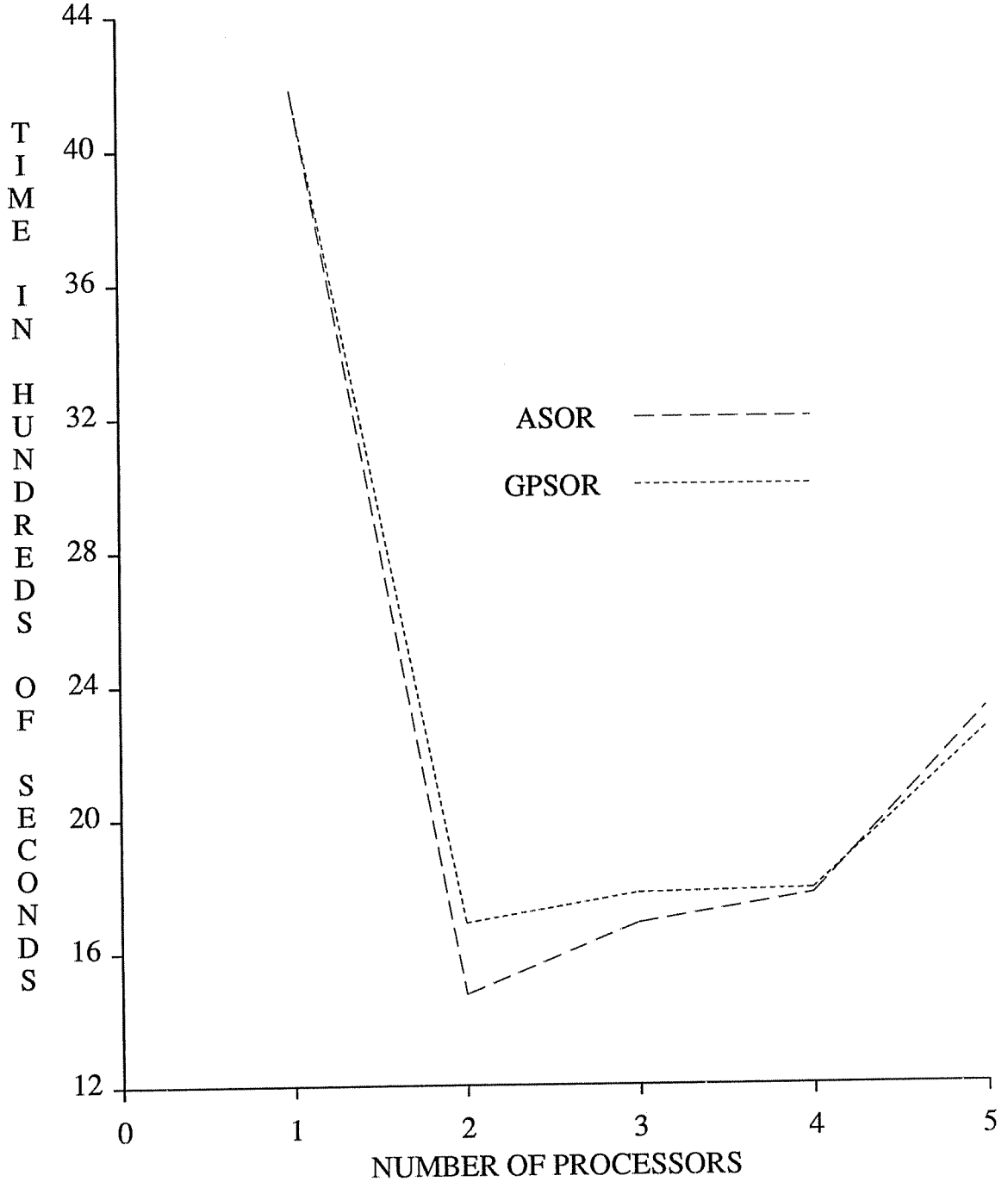

Figure 2.3 Total Time (density=.8%, m=1000, n=4000)

Chapter 3

A PARALLEL PIVOTAL ALGORITHM FOR SOLVING THE LINEAR COMPLEMENTARITY PROBLEM

3.1 INTRODUCTION

In this chapter we propose a parallel distribution of Lemke's algorithm, [Lemke65], for solving LCPs. This distribution is designed to exploit a multi-computer architecture where a number of processors are loosely coupled such as CRYSTAL [DeWitt et al84].

Parallel distribution of algorithms such as Lemke's algorithm allows for solving much larger problems in shorter time. See [Schnabel84] for a discussion on parallel optimization as well as a survey of work done in parallel optimization. Bhide proposed a distributed algorithm for the simplex algorithm [Finkel et al86]. Preliminary testing was done on small LPs. The initial results showed communication overhead dominated the calculations.

Phillips and Rosen propose a parallel algorithm for solving the LCP based on solving a multiple cost row linear program [Phillips & Rosen86]. This algorithm handles an LCP when M is indefinite. However, they suggest using Lemke's algorithm when M is positive semidefinite. They present computational experience on the CRAY X-MP/48.

A major trend in computing has been the creation of large networks of computer workstations. See [Agrawal & Jagadish86] for a model for parallel processing on these networks. These networks have relatively high communication costs. Therefore it is crucial that algorithms designed to run on these networks minimize the need to communicate. The proposed parallel implementation of Lemke's algorithm is tested on the CRYSTAL multi-computer. We also compare computational results with an implementation on a tightly coupled multiprocessor, the Sequent Balance 21000. This machine consists of eight processors running at 10 MHz, with a 8 kbyte cache sharing a global memory via a 32-bit bus, with 8 megabytes of physical memory.

Section 3.2 will discuss Lemke's algorithm. Section 3.3 will discuss the parallel implementation of Lemke's algorithm designed for multi-computers. Finally, Section 3.4 will present a model for predicting speedups. Computational results are compared with the predicted speedups. A comparison of results using the Balance 21000 will be presented.

3.2 LEMKE'S ALGORITHM FOR SOLVING THE LCP

Pivotal methods are based on moving from one vertex to another on a polyhedral set S where

$$S := \{z | z \geq 0, Mz + q \geq 0\} \tag{3.1}$$

Mangasarian has shown that the LCP is equivalent to a constrained minimization of a concave function [Mangasarian78] which we state in the following lemma.

Lemma 3.1. *(LCP equivalence with a concave minimization problem) Let S be defined as in (3.1). Then $\bar{z}$ solves the LCP if and only if*

$$0 = \sum_{i=1}^{n} \bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = \min_{z \in S} \sum_{i=1}^{n} z_i - (z_i - (Mz + q)_i)_+$$

Proof :

First we note that the objective function in the minimization problem is nonnegative on S. Now suppose $\bar{z}$ solves the LCP. We need only show that $0 = \sum_{i=1}^{n} \bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+$.

Case I:

Suppose $\bar{z}_i = 0$. Then

$$\bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = -(-(M\bar{z} + q)_i)_+ = 0$$

since

$$(M\bar{z} + q)_i \geq 0.$$

Case II:

Suppose $\bar{z}_i > 0$. Then

$$\bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+ = \bar{z}_i - (\bar{z}_i)_+ = 0$$

since

$$(M\bar{z} + q)_i = 0.$$

Therefore

$$0 = \sum_{i=1}^{n} \bar{z}_i - (\bar{z}_i - (M\bar{z} + q)_i)_+.$$

Hence $\bar{z}$ solves the minimization problem. We now show that if $\bar{z}$ solves the minimization problem that $\bar{z}$ solves the LCP.

Suppose $\bar{z}$ solves the minimization problem. Then $\bar{z}_i - (\bar{z}_i - (M\bar{z}+q)_i)_+ = 0$ for $i = 1, 2, \ldots, n$.

Case I: Suppose $\bar{z}_i - (M\bar{z} + q)_i \geq 0 \Rightarrow (M\bar{z} + q)_i = 0 \Rightarrow \bar{z}_i \geq 0$.

Case II: Suppose $\bar{z}_i - (M\bar{z} + q)_i < 0 \Rightarrow \bar{z}_i = 0 \Rightarrow (M\bar{z} + q)_i \geq 0$.

Therefore $\bar{z}$ solves the LCP. ∎

The LCP has a solution at a vertex as a consequence of the following lemma from [Mangasarian78].

Lemma 3.2. *If the linear complementarity problem has a solution, it has a solution at a vertex of S.*

Proof :

Because S is contained in the nonnegative orthant, it does not contain any straight lines (that go to infinity on both ends) and hence by Corollary 32.3.2 in [Rockafellar70] the concave minimization problem must have a solution at a vertex of S, which by Lemma 3.1 must solve the LCP. ∎

Lemke's algorithm [Lemke65] is based on the following equivalence obtained by enlarging the space of the problem by adding artificial variables.

$$\left\{\begin{array}{r} Mz+q\geq 0 \\ z\geq 0 \\ z(Mz+q)=0 \end{array}\right\} \iff \left\{\begin{array}{r} w=Mz+ez_0+q\geq 0 \\ w_0=z_0 \\ (z,z_0)\geq 0 \\ (z,z_0)\begin{pmatrix} w \\ w_0 \end{pmatrix}=0 \end{array}\right\}$$

This representation has the advantage that we can easily identify a feasible point. Before we specify the algorithm we need to state the following definition.

Definition 3.1. *The points* $(z,z_0),(w,w_0)$ *are said to be* **almost complementary** *if* $z_iw_i=0$ *except for at most one* i.

We are now ready to specify the algorithm due to Lemke.

Algorithm 3.1. (Lemke's Algorithm)

Step I Let w be basic and z be nonbasic. Therefore $w=q$ and $z=0$. Add an artificial nonbasic variable z_0 and an artificial basic variable $w_0=z_0$. Increase the value of z_0 until the most negative w_i becomes zero. This corresponds to pivoting on the row with most negative q_i and column z_0. The problem is now feasible and almost complementary.

Step II Exchange a nonbasic variable and some basic variable while maintaining feasibility and almost complementarity. Therefore the nonbasic variable that enters the basis is the variable complementary to the variable which just became nonbasic. Continue this step until the process terminates at a solution or an unbounded ray.

The LCP is feasible if S is nonempty. Lemke's algorithm will converge to a solution if the LCP is feasible and M is copositive plus or a P- matrix.

3.3 IMPLEMENTATION OF PARALLEL LEMKE

The time consuming portion of Lemke's algorithm is the pivoting step. The pivoting step is $O(n^2)$, therefore when n is large each iteration will be very time consuming. The idea then is to distribute the pivoting among r processors in the hopes of obtaining a solution r times as fast.

Suppose we are trying to solve the LCP and define

$$A := [M \quad e \quad q]$$

Then A is the pivot matrix and $A_{.(n+2)}$ is the right hand side where n is the dimension of the LCP. Suppose that r is the pivot row and s is the pivot column, then for $i \neq r, j \neq s$ we have

$$a_{ij} \leftarrow a_{ij} - \frac{a_{is}a_{rj}}{a_{rs}}$$

The key to parallelizing Lemke's algorithm is to note that when updating the pivot matrix we need only elements in the pivot row and the pivot column. Therefore there are two natural ways to divide the pivot matrix, in horizontal blocks or in vertical blocks. Then each horizontal or vertical block will be updated simultaneously in separate processors.

If we divide the pivot matrix into horizontal blocks each processor has the portion of the pivot column it needs to update all the elements in its block. Therefore each processor needs only the pivot row to complete the update. Likewise if we divide the pivot matrix into column blocks we need only the pivot column to complete the update.

Because we are considering an implementation on a loosely coupled network with relatively heavy communication costs we would like to avoid unnecessary communication. In Lemke's algorithm we must determine the pivot column and row. The pivot column is always the column corresponding to the complement of the variable which last became nonbasic. Therefore locating the pivot column is a simple matter of keeping track of the location of the variables. This is easily done independently in each processor. However, we find that the horizontal partition results in extra communications to complete the ratio test necessary to determine the appropriate pivot row.

The pivot row is selected by employing a ratio test as follows

$$r = \underset{1 \leq i \leq n}{\operatorname{argmin}} \{-\frac{a_{i,n+2}}{a_{is}} \mid a_{is} < 0\}.$$

If the the pivot matrix A is partitioned in horizontal blocks each processor would find the minimum ratio of the portion of the pivot column it owned. Then a communication to a master processor would be necessary whereby this processor would determine the minimum of all ratios and notify the appropriate processor with another communication. The processor which owned the pivot row

would then proceed to share it with all other processors. Therefore this results in $2\ (\ p\ -\ 1\ )\ +\ 1$ communications, the first $p-1$ communications are needed to send the master the minimum ratio, the next communication to notify the proper processor, and finally the last $p-1$ to send the pivot row.

If the pivot matrix was partitioned in vertical blocks then each processor either waits for or sends the pivot column. Then if each processor holds a copy of $A_{.(n+2)}$ each processor can independently determine the pivot row and proceed with the update. Therefore the vertical division saves p communications.

We suggest the following scheme. Each processor j holds a column block of the matrix A and a copy of $A_{.n+2}$ in its memory. Then processor j proceeds through the following steps.

Step I. *Determine the pivot column s*

If s is in memory

send pivot column

else

receive pivot column

Step II. *Ratio Test*

Step III. *Pivot*

Step IV. *If complementarity is satisfied then*

stop

else

go to step I.

3.4 COMPUTATIONAL EXPERIENCE

Table 3.1 displays the time in seconds to solve three randomly generated problems of dimension 50, 200, and 425 on the CRYSTAL multi–computer. The problems were tested using one to eight processors.

Table 3.1

CRYSTAL TIME IN SECONDS

processors	*Problem Dimension*		
p	50	200	425
1	1.8163	90.8340	629.7770
2	1.0159	46.5438	316.573
3	.7808	31.6945	212.6680
4	.6771	24.6202	161.5050
5	.6261	20.4335	129.9620
6	.5972	17.6500	110.2570
7	.6011	15.7600	96.0057
8	.6164	14.2546	86.1556

To assess the speedup from using p processors we define the following measurement of speedup.

$$S_p := \frac{\text{Computing time on one processor}}{\text{Computing time using p processors.}}$$

An upper bound on S_p would be p. This would occur if there was no overhead due to the parallelism of the algorithm. Therefore a goal in producing an efficient parallel algorithm would be to try to approach a speedup of p for p processors.

Expected speedup based on an elementary operation count for the proposed algorithm will be defined as

$$S_p^e := \frac{\text{Expected time for one iteration on one processor}}{\text{Expected time for one iteration on } p \text{ processors}}$$

Note that we have both serial and parallel components of the algorithm. Finding the pivot column and finding the pivot row are repeated independently in each processor and therefore can be thought of as serial. This step is $O(n)$. Pivoting is done in parallel and takes $\frac{2n^2}{p}$ steps. We also must take into account communication costs. At each iteration one processor sends a pivot column in $k(n)$ buffers to the $p-1$ remaining processors. Here $k(n)$ is the number of buffers needed to send a vector of n real numbers and is given by $k(n) := \lceil \frac{n}{s} \rceil$. Therefore $k(n)$ is the smallest integer greater than or equal to $\frac{n}{s}$, where s is the number of elements that can be transmitted by one buffer. Therefore define

$$S_p^e := \frac{2n^2 + \beta n}{2n^2/p + \beta n + \gamma k(n)(p-1)} = \frac{2n + \beta}{2n/p + \beta + \gamma k(n)(p-1)/n}$$

where β is an unknown parameter and γ is a ratio of time for communication divided by the time for one operation. Using this definition we can maximize speedup for fixed n by using $p = \sqrt{\frac{2}{\gamma k(n)}} \times n$ processors. A least squares fit

over observed data gives us $\beta = 4.4014$ and $\gamma = 143.5467$. Therefore we would expect speedup to be maximized when p is 5.9, 16.7 and 28.9 for dimensions 50, 200,and 425 respectively. Clearly, p must be an integer number. Figure 3.1 shows the percentage of the maximum speedup we obtain for a fixed dimension given the number of processors p. We see that the curve is steep for $p < \bar{p}$ where $\bar{p} = \sqrt{\frac{2}{\gamma k(n)}} \times n$. However, for $p > \bar{p}$ the curve decreases slowly. Therefore when there is a question of rounding $\bar{p}$ one should round up rather than down. Table 3.2 compares expected speedup with the speedups obtained on test problems for problem sizes 50, 200, and 425. Figures 3.2, 3.3 and 3.4 display table 3.2 graphically. Finally figure 3.5 graphs the computed speedups for the different dimensions so we can see that as n grows large the speedup approaches p for $p << n$.

We would expect to get better speedups on a shared memory machine like the Balance 21000. The implementation on Balance 21000 is similar except we save on communication costs because the pivot column is in shared memory. Table 3.3 compares computed speedups on Balance 21000 with computed speedups on CRYSTAL for problem sizes 50,200,425. We see that the CRYSTAL speedups drop off faster as we move to more processors than the Balance 21000 speedups. This drop is noticed almost immediately for the smaller dimensioned problem, however CRYSTAL is competitive with Balance 21000 at dimension 425 through seven processors. Figure 3.6 displays speedup curves of CRYSTAL versus Balance 21000 for the problem of dimension 50. Figure 3.7 displays speedup curves for the problem of dimension 425.

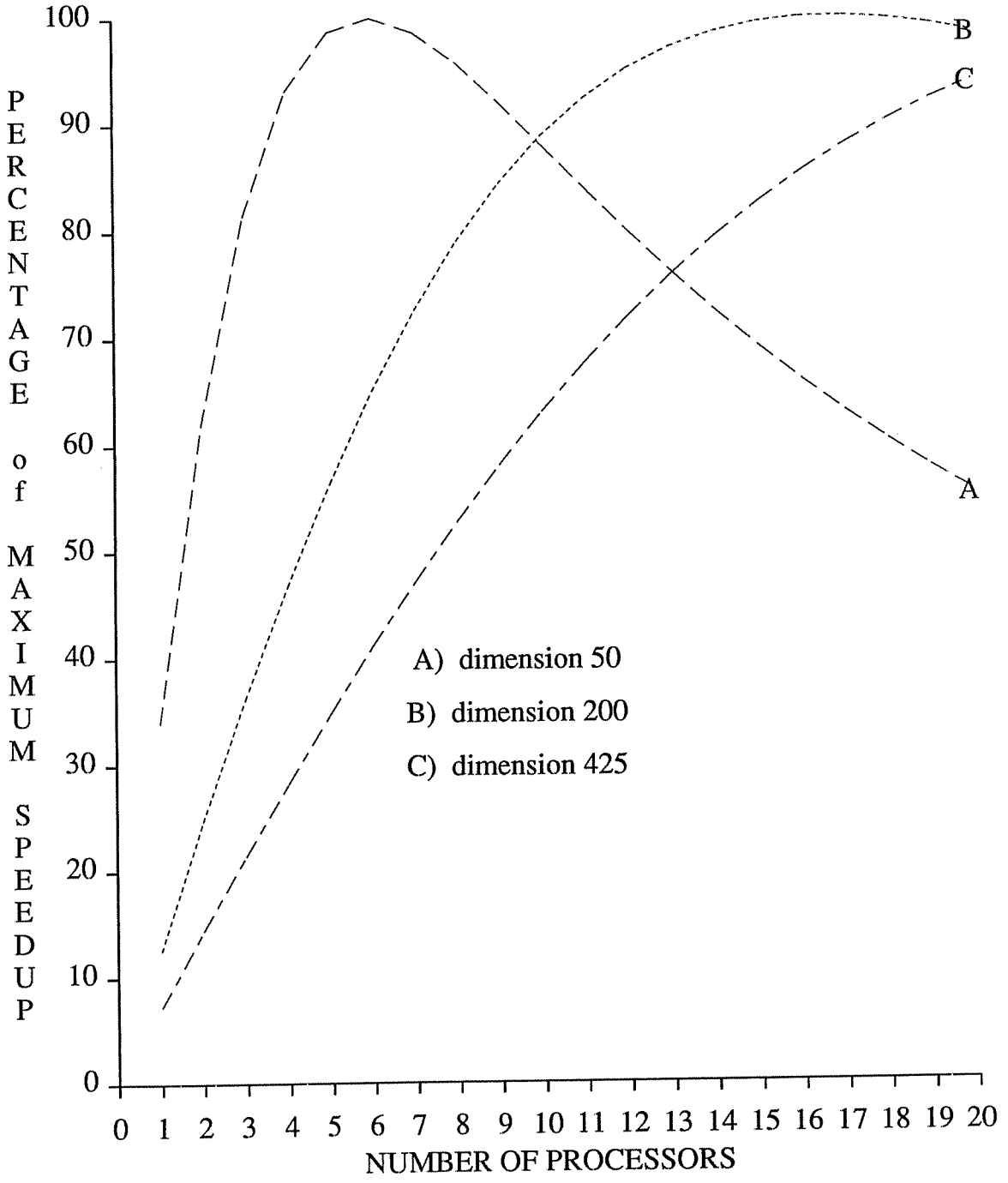

Figure 3.1 Percentage of Maximum Speedup

Table 3.2

EXPECTED VERSUS COMPUTED SPEEDUPS

	50		200		425	
p	S_p	S_p^e	S_p	S_p^e	S_p	S_p^e
2	1.7878	1.8227	1.9515	1.9646	1.9893	1.9850
3	2.3262	2.4009	2.8659	2.8760	2.9613	2.9486
4	2.6824	2.7458	3.6894	3.7198	3.8994	3.8845
5	2.9009	2.9087	4.4453	4.4857	4.8458	4.7875
6	3.0413	2.9467	5.1464	5.1677	5.7119	5.6529
7	3.0216	2.9065	5.7635	5.7634	6.5597	6.4767

Table 3.3

CRYSTAL VERSUS BALANCE SPEEDUPS

	50		200		425	
p	CRYS	Balan	CRYS	Balan	CRYS	Balan
2	1.7878	1.8108	1.9646	1.8891	1.9959	1.9893
3	2.3262	2.3928	2.8760	2.8309	2.9613	2.9762
4	2.6824	3.0454	3.7198	3.6932	3.8994	3.9860
5	2.9009	3.3500	4.4857	4.4933	4.8945	4.9521
6	3.0413	3.7222	5.1677	5.4505	5.7119	5.8777
7	3.0216	3.9411	5.7634	6.3699	6.5597	6.7854

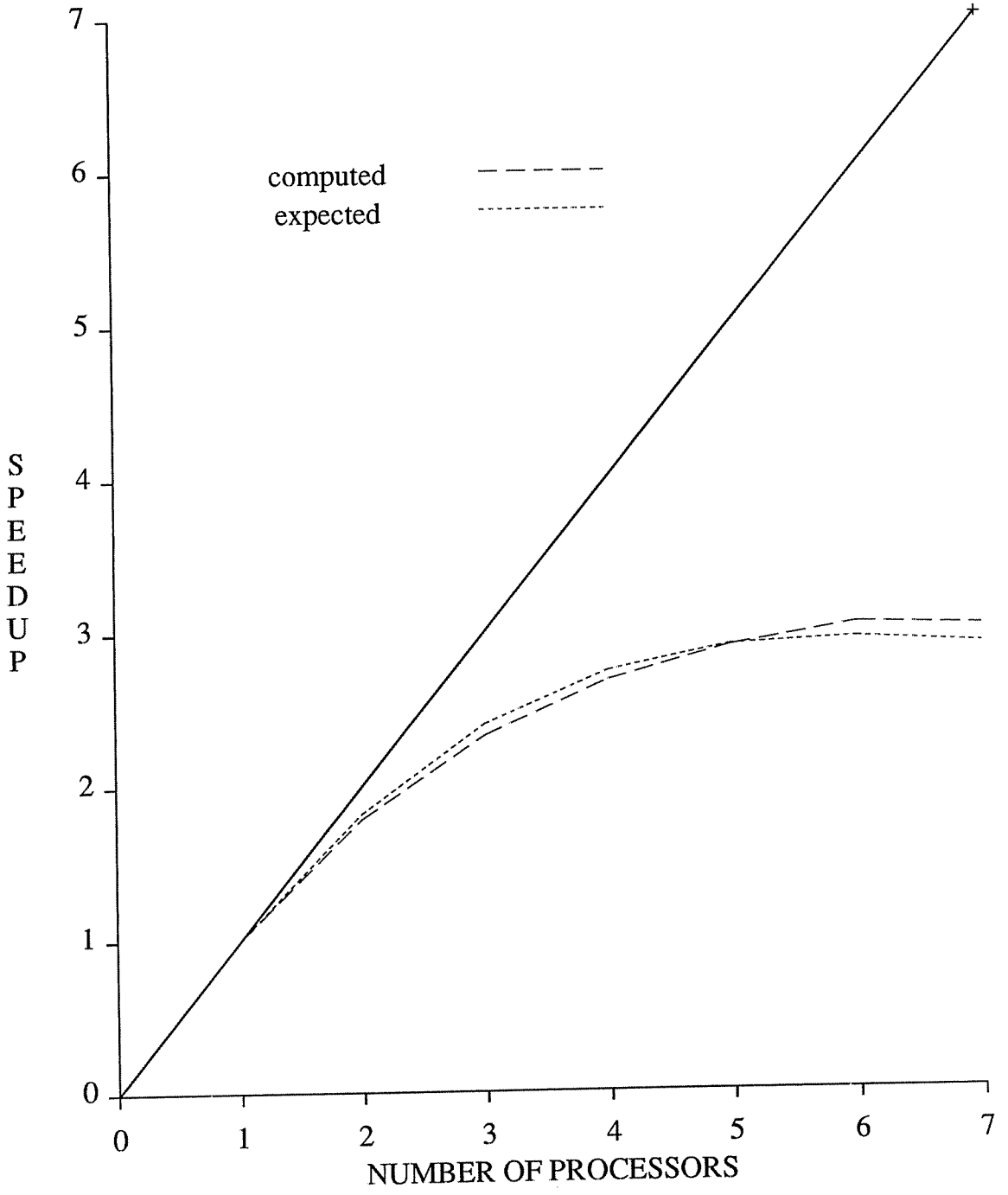

Figure 3.2 CRYSTAL Speedups: Expected versus Computed, Dimension 50

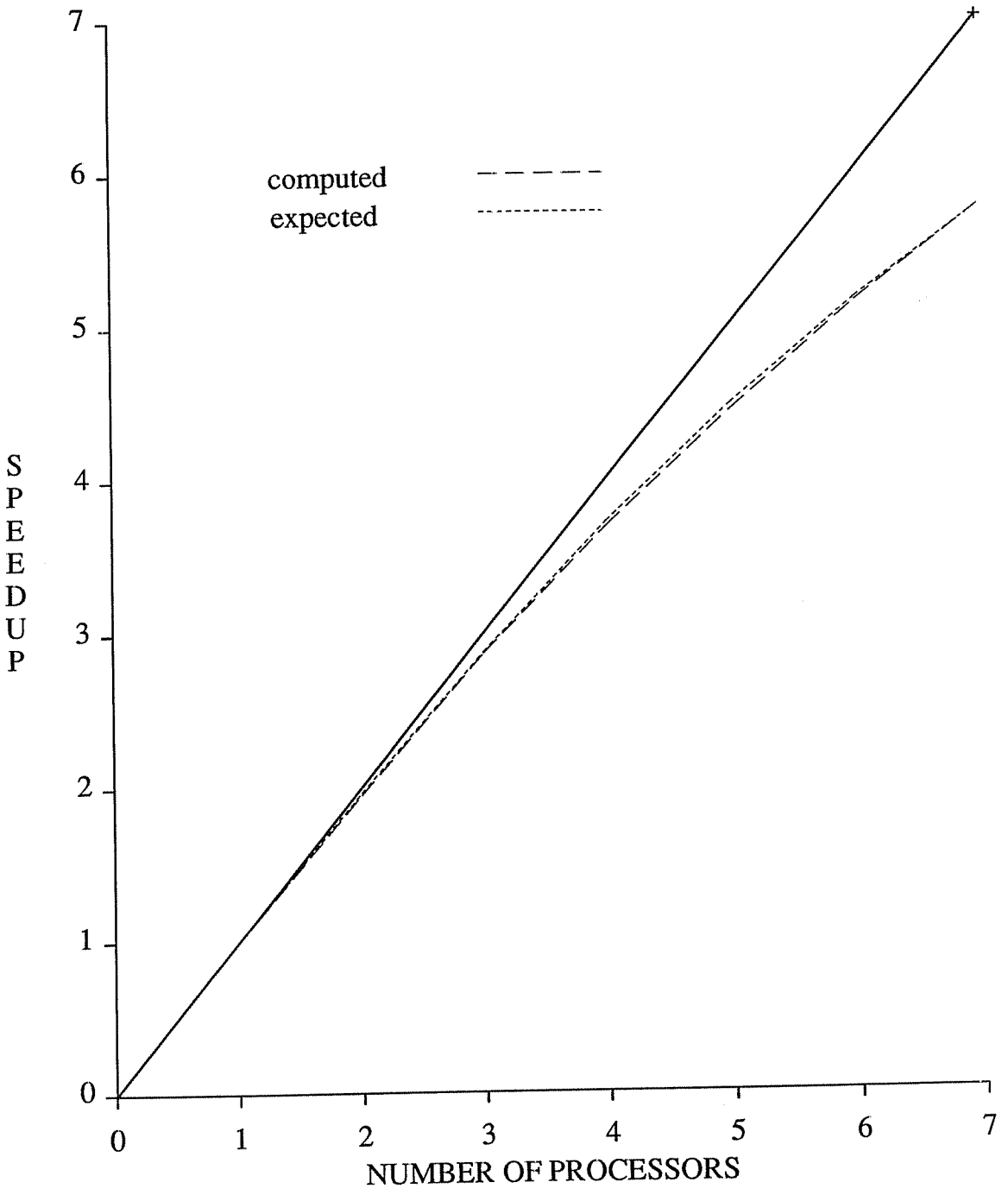

Figure 3.3 CRYSTAL Speedups: Expected versus Computed, Dimension 200

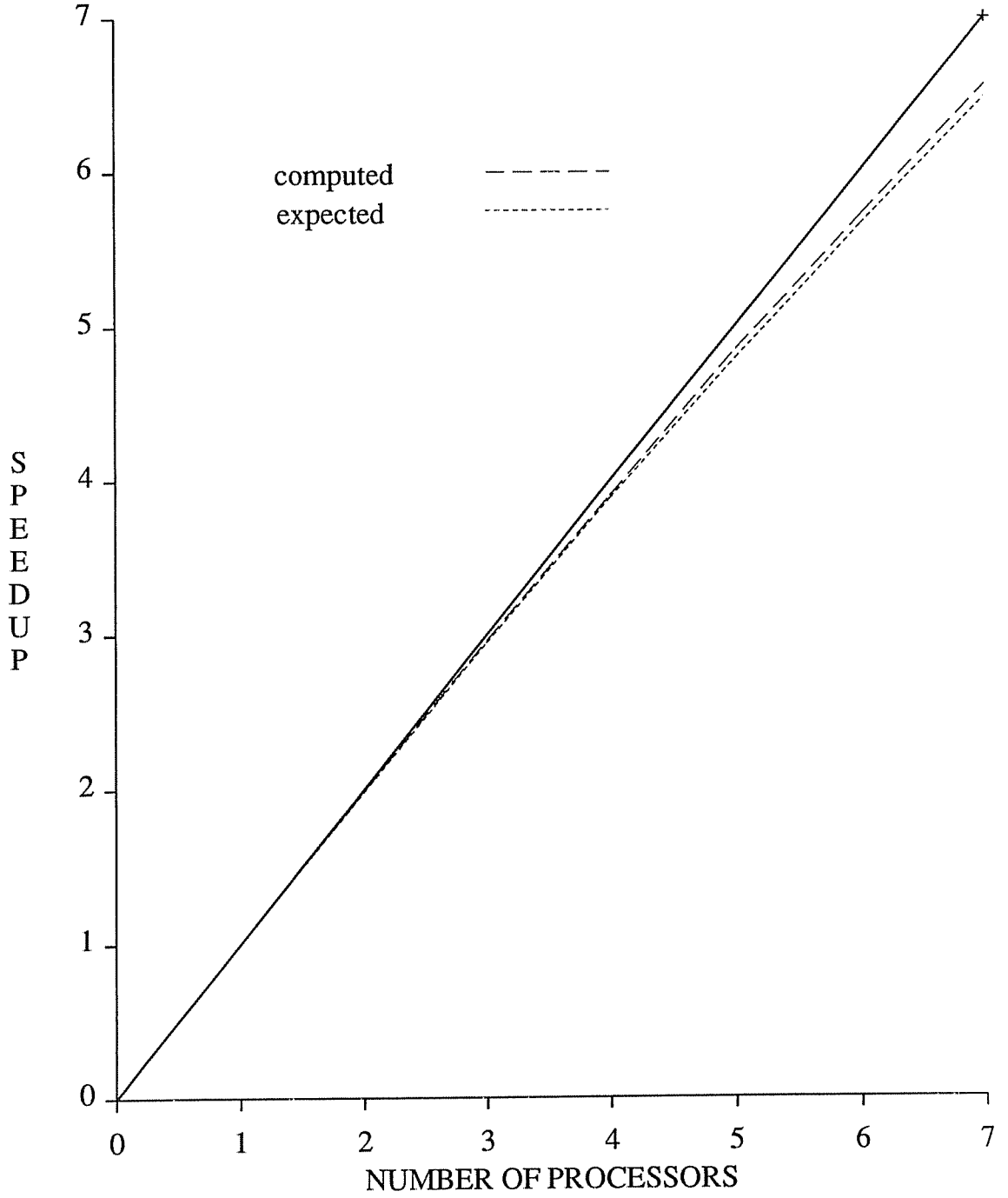

Figure 3.4 CRYSTAL Speedups: Expected versus Computed, Dimension 425

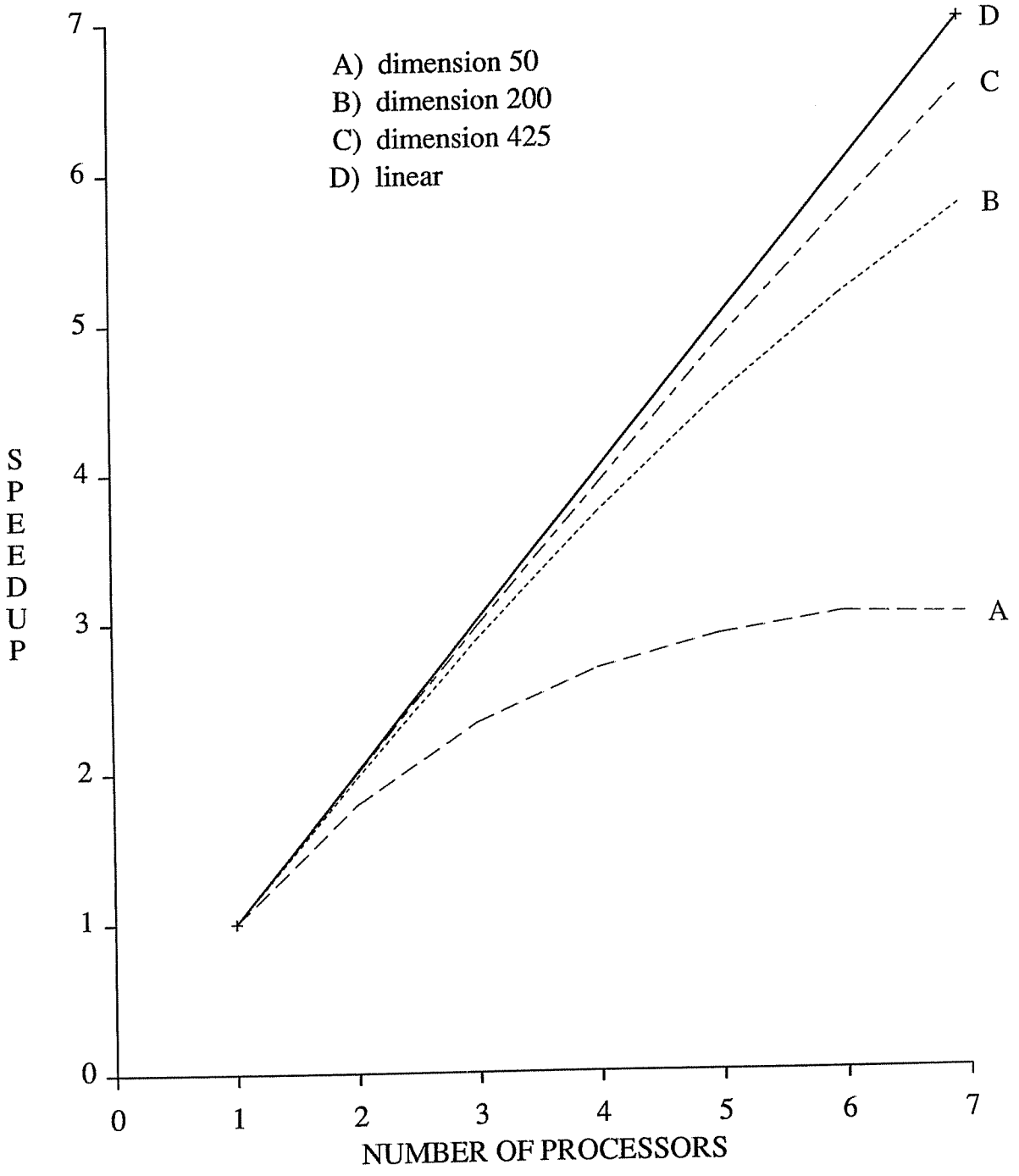

Figure 3.5 CRYSTAL Speedups

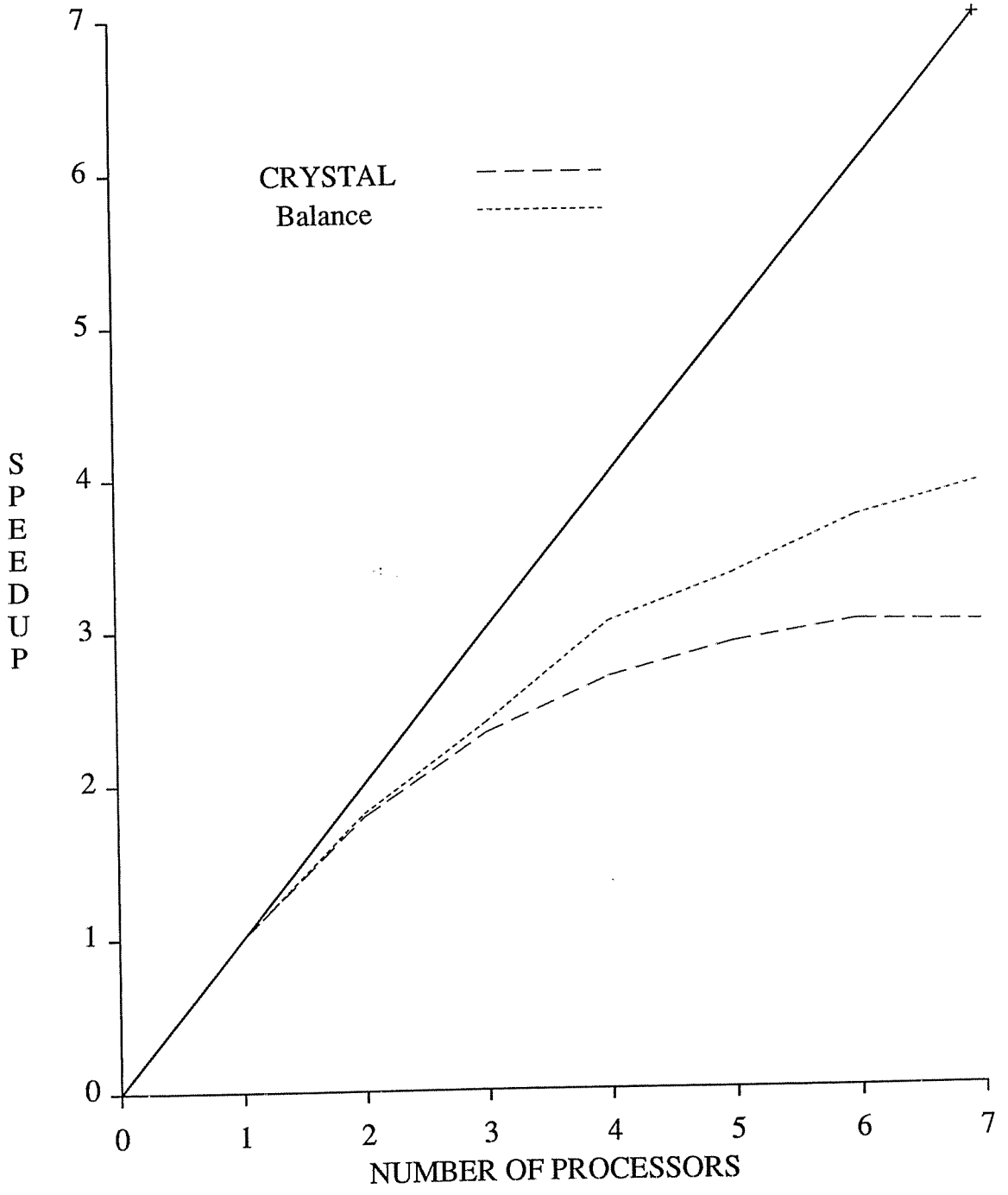

Figure 3.6 CRYSTAL versus Balance Speedups: dimension 50

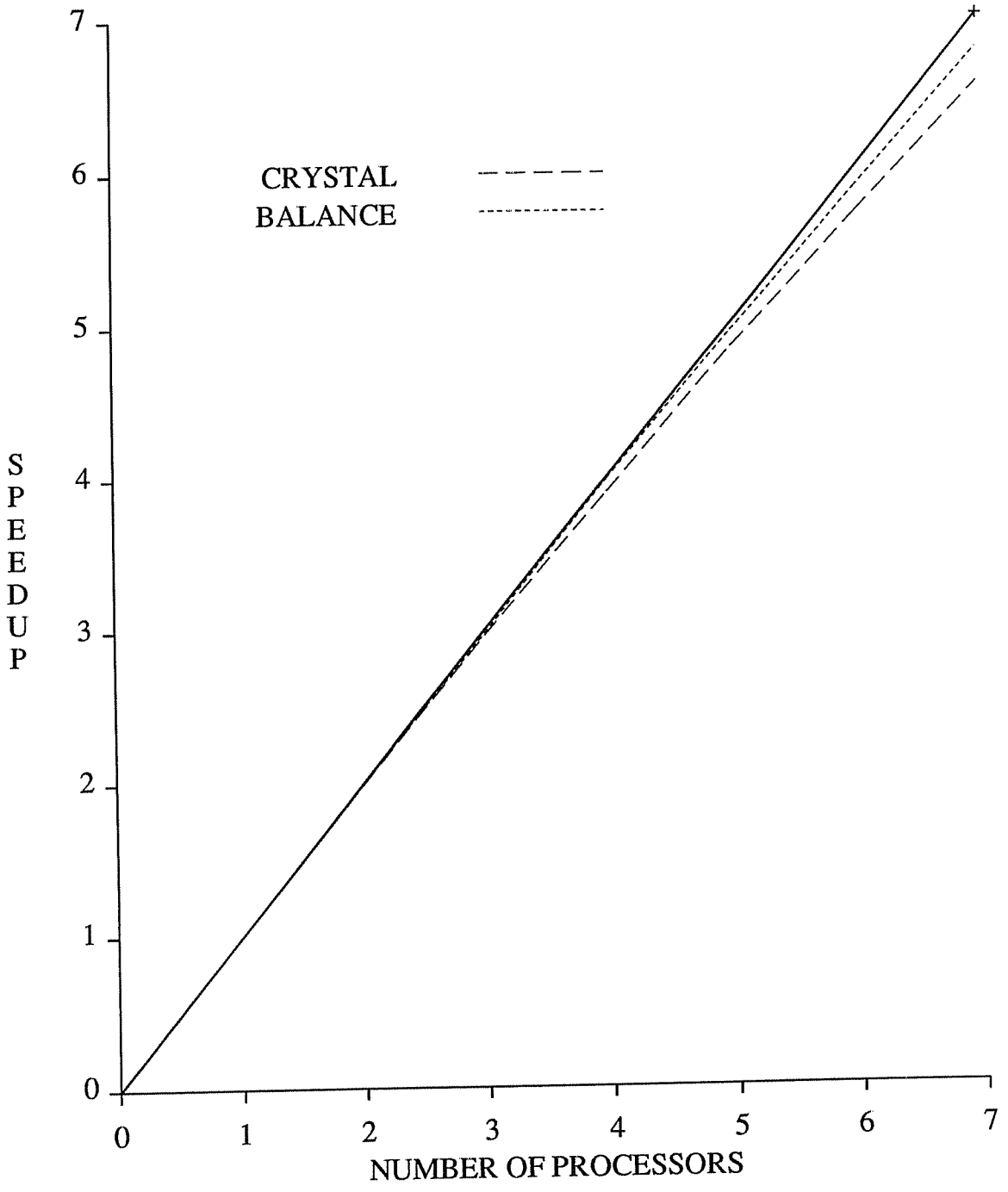

Figure 3.7 CRYSTAL versus Balance Speedups: dimension 425

Chapter 4

TWO–STAGE SUCCESSIVE OVERRELAXATION ALGORITHM

4.1 INTRODUCTION

Successive overrelaxation algorithms are characterized by fast movement into a neighborhood of the solution followed by slow movement close to the solution. An active set strategy would therefore be useful after a few initial iterations of the SOR algorithm. The initial iterates of SOR are designed to identify a good guess for an initial active set. In large–scale programming this step is crucial. Typically active set strategies allow at most one constraint to be added to the active set at each iteration. Lenard, [Lenard79] conducted a computational study of various active set strategies on small quadratic programming problems. In her study, in general the number of changes in the active set appeared to fall in the range of $\frac{n}{3}$ to $\frac{n}{2}$, where n in the number of problem variables. In addition this number approximately corresponds to the number of active constraints at the solution of the test problems. Therefore if the initial active set chosen is not close to the solution active set an enormous number of iterations may be required for large–dimensional problems. Since each iteration may involve solving a large–scale problem, this is not feasible for large–scale programming. Discussion of active set methods can be found in numerous sources including [Gill & Murray74],[Goldfarb72],and [Hoyle86]. By contrast our proposed two–stage method provides a good initial guess of the final active constraint set.

In a related work [Bertsekas82] proposes a projected Newton method. He proposes a superlinearly convergent scheme for solving

$$\min_{z \geq 0} f(z)$$

using the following iteration,

$$z^{i+1} = (z^i - \lambda^i D_i \nabla f(z^i))_+$$

where λ^i is a stepsize chosen by an Armijo–like rule, D_i is a positive definite symmetric matrix which is partly diagonal and f(z) is strictly convex. The portion of D_i which is not diagonal may be chosen to be the inverse of the Hessian of $f(z)$ with respect to the indices corresponding to a subset of the variables. Although Bertsekas uses an active set strategy to partition the variables into two sets, strictly speaking this is not an active set method. Active set methods are forced to remain on a linear manifold until a certain criterion is violated. In contrast Bertsekas' method allows the iterates to move off the active constraint set at any given iteration. This allows for multiple changes in the active set at any given iteration hence avoiding the inherent limitations of manifold suboptimization methods.

We propose here a two–stage SOR method. The idea is to use the SOR algorithm until a certain tolerance is reached and then use the current approximation of the solution to identify a set of constrained variables. We use a second SOR scheme for solving a complementary system of equations which leads to a new feasible point at which the process repeats. The use of the SOR algorithm at the

beginning helps identify a set of active constraints which may be close to the set of active constraints at a solution. The second step is a Bertsekas–like method which is sparsity preserving.

Section 4.2 will specify the algorithm and state optimality conditions. Section 4.3 will establish convergence results. Finally, in Section 4.4 we discuss implementation of the algorithm and computational experience.

4.2 TWO–STAGE SOR ALGORITHM

We consider the problem

$$\min_{z \geq 0} f(z) = \frac{1}{2} zMz + qz \tag{4.1}$$

If M is symmetric positive semi-definite solving problem (4.1) is equivalent to solving the symmetric linear complementarity problem.

The idea of Two–Stage SOR (TSOR) is to use the SOR algorithm to approximate a solution to problem (4.1) and use this approximate solution to partition the variables into two sets. Define

$$\mathcal{I} := \mathcal{I}(\bar{z}) = \{j | \bar{z}_j > 0\} \quad \text{for } \bar{z} \text{ a solution of problem (4.1)}$$

Define the set at the ith iterate as follows

$$\mathcal{I}_i := \{j | z_j^i \leq \epsilon^i\} \quad \text{for some} \quad \epsilon^i \geq 0, \quad \epsilon^i \geq \bar{\epsilon} > 0. \tag{4.2}$$

From this we can partition M and z as follows

$$M := \begin{bmatrix} M_{\mathcal{I}_i} \\ M_{\mathcal{I}_i^c} \end{bmatrix}$$

$$z := \left[{}^{z}\mathcal{I}_i \; {}^{z}\mathcal{I}_i^c \right]$$

where $\mathcal{I}_i^c$ is the complement of $\mathcal{I}_i$ in $\{1, 2, \ldots, n\}$. We guess that $\mathcal{I}_i^c$ is the set of indices of components of z which will be positive in the solution. Therefore we must solve

$$M_{\mathcal{I}_i^c} z + q_{\mathcal{I}_i^c} = 0 \tag{4.3}$$

to satisfy complementarity. We use an SOR algorithm to approximate a solution to (4.3). Note that if M is positive definite, $\epsilon = 0$ and $\mathcal{I}_i^c = \mathcal{I}$, then if we solve (4.3) we solve (4.1) exactly in one step.

We are now ready to specify the TSOR algorithm.

Algorithm 4.1.

Let E be a positive diagonal matrix in $\mathbb{R}^{n \times n}$, let $z^0 \geq 0$. Let $\omega > 0$ such that for some $\gamma_1, \gamma_2 > 0$

$$y((\omega E)^{-1} + K)y \geq \gamma_1 \parallel y \parallel^2 \quad , \forall y \in \mathbb{R}^n \tag{4.4}$$

$$y((\omega E)^{-1} + K - \frac{M}{2})y \geq \gamma_2 \parallel y \parallel^2 \quad , \forall y \in \mathbb{R}^n \tag{4.5}$$

Step 1

For $i \leq l$ where l is a positive integer

$$z^{i+1} = (z^i - \omega E(Mz^i + q + K(z^{i+1} - z^i)))+$$

stop if $z^{i+1} = z^i$.

Step 2

Direction Generation

$$d^i := (p(z^i) - z^i)$$

where

$$p(z^i) := \begin{bmatrix} p^k_{\mathcal{I}_i^c}(z^i) \\ p^1_{\mathcal{I}_i}(z^i) \end{bmatrix}$$

and

$$p^k_{\mathcal{I}_i^c}(z^i) = p^{k-1}_{\mathcal{I}_i^c}(z^i) - \omega E_{\mathcal{I}_i^c\mathcal{I}_i^c}(M_{\mathcal{I}_i^c\mathcal{I}_i^c} p^{k-1}_{\mathcal{I}_i^c}(z^i) + M_{\mathcal{I}_i^c\mathcal{I}_i} z^i_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} + K_{\mathcal{I}_i^c\mathcal{I}_i^c}(p^k_{\mathcal{I}_i^c}(z^i) - p^{k-1}_{\mathcal{I}_i^c}(z^i))) \quad (4.6)$$

$$p^1_{\mathcal{I}_i}(z^i) = (z_{\bar{\mathcal{I}}_i} - \omega E_{\mathcal{I}_i\mathcal{I}_i}(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i}))+ \quad (4.7)$$

Stop if $p(z^i) = z^i$.

Stepsize Generation

$$z^{i+1} = z^i + \lambda^i d^i$$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda}\{f(z^i + \lambda d^i) | z^i + \lambda d^i \geq 0\} \quad (4.8)$$

Go to step 2.

We are now ready to state optimality conditions for Algorithm 4.1.

Theorem 4.1. *(TSOR optimality conditions) Let* $M,K,E \in \mathbb{R}^{n\times n}, q \in \mathbb{R}^n, \omega >$ 0 *and* E *be a positive diagonal matrix.*

(a) If z *solves the LCP and* $(\omega E_{\mathcal{F}\mathcal{F}})^{-1} + K_{\mathcal{F}\mathcal{F}}$ *is nonsingular, where* $\mathcal{F}$ *is any subset of* $\{1,2,\ldots,n\}$*, then* $p(z) = z$ *where* $p(z)$ *is the solution of (4.6) and (4.7).*

(b) If $p(z)$ *solves (4.6) and (4.7) and* $p(z) = z$ *and if* $(\omega E)^{-1} + K - \frac{M}{2}$ *and* $(\omega E)^{-1} + K$ *are positive definite then* z *solves the LCP.*

Proof :

(a) Suppose z solves the LCP. $\mathcal{I} = \{j | z_j \leq \epsilon\}$. It is easy to verify that

$$z_{\mathcal{I}} = (z_{\mathcal{I}} - \omega E_{\mathcal{I}\mathcal{I}}(M_{\mathcal{I}}z + q_{\mathcal{I}}))_+$$

Consider $p^1_{\mathcal{I}c}(z) = z_{\mathcal{I}c} - \omega E_{\mathcal{I}c\mathcal{I}c}(M_{\mathcal{I}c}z + q_{\mathcal{I}c} + K(p^1_{\mathcal{I}c}(z) - z_{\mathcal{I}c}))$. By rearranging terms we get

$$((\omega E_{\mathcal{I}c\mathcal{I}c})^{-1} + K_{\mathcal{I}c\mathcal{I}c})p^1_{\mathcal{I}c}(z) = ((\omega E_{\mathcal{I}c\mathcal{I}c})^{-1} + K_{\mathcal{I}c\mathcal{I}c})z_{\mathcal{I}c}.$$

Therefore since $((\omega E_{\mathcal{I}c\mathcal{I}c})^{-1} + K_{\mathcal{I}c\mathcal{I}c}$ is nonsingular $p^1_{\mathcal{I}c}(z) = z$. This implies that $p^k_{\mathcal{I}c}(z) = z$ for every k. Therefore $p(z) = z$.

(b) Now suppose $p(z) = z$. One can easily show that

$$z_{\mathcal{I}} \geq 0, M_{\mathcal{I}}z + q_{\mathcal{I}} \geq 0, z_{\mathcal{I}}(M_{\mathcal{I}}z + q_{\mathcal{I}}) = 0$$

We now show that

$$z_{\mathcal{I}c} \geq 0, M_{\mathcal{I}c}z + q_{\mathcal{I}c} \geq 0, z_{\mathcal{I}c}(M_{\mathcal{I}c}z + q_{\mathcal{I}c}) = 0 \tag{4.9}$$

is satisfied. It is easy to show that if $p^k_{\mathcal{I}^c}(z) = z_{\mathcal{I}^c}$ and $k = 1$ then (4.5) is satisfies. We will show that this is true for arbitrary k. Assume that $p^k_{\mathcal{I}^c}(z) = z$ for some $k \geq 2$ but $p^1_{\mathcal{I}^c}(z) \neq z_{\mathcal{I}^c}$. Then $z_{\mathcal{I}^c}$ is an accumulation point of the iteration

$$z^{i+1} = z^i - \omega E_{\mathcal{I}^c\mathcal{I}^c}(M_{\mathcal{I}^c\mathcal{I}^c} z^i_{\mathcal{I}^c} + M_{\mathcal{I}^c\mathcal{I}} z_{\mathcal{I}} + q_{\mathcal{I}^c} + K_{\mathcal{I}^c\mathcal{I}^c}(z^{i+1}_{\mathcal{I}^c} - z^i_{\mathcal{I}^c}))$$

Then since $(\omega E_{\mathcal{I}^c\mathcal{I}^c})^{-1} + K_{\mathcal{I}^c\mathcal{I}^c} - \frac{M_{\mathcal{I}^c\mathcal{I}^c}}{2}$ is positive definite and by the argument in Theorem 2.1 of [Mangasarian77],

$$M_{\mathcal{I}^c} z + q_{\mathcal{I}^c} = 0$$

But then by part (a) $z = p(z)$. In particular $z_{\mathcal{I}^c} F = p^1_{\mathcal{I}^c}(z)$. This contradicts the assumption. Therefore (4.5) is satisfied. ∎

4.3 CONVERGENCE OF TSOR

In order to prove the convergence result we first need the following two Lemmas.

Lemma 4.1. *Let $f(z) = \frac{1}{2} zMz + qz$. Let the sequence $\{z^i\}$ be generated by the following SOR iteration*

$$z^{i+1} = z^i - \omega E(Mz + q + K(z^{i+1} - z^i))$$

Then the sequence $\{f(z^i)\}$ is non-increasing.

Proof :

The proof follows from a simple modification of the proof of Theorem 2.1 in [Mangasarian77]. ∎

Suppose the sequence generated by the TSOR algorithm has a convergent subsequence. Lemma 4.2 will establish the boundedness of the directions associated with that convergent subsequence.

Lemma 4.2. *If $z^{ij} \to \bar{z}$ then d^{ij} is bounded.*

Proof :

We have $d^{ij} = (p(z^{ij}) - z^{ij})$.

Since the sets $\mathcal{I}^{ij}$ are finite without loss of generality assume $\mathcal{F} = \mathcal{I}^{i1} = \mathcal{I}^{i2} = \ldots = \mathcal{I}^{ij} = \ldots$. We will show that the components of $p(z^{ij})$, $p^k_{\mathcal{F}^c}(z^{ij})$ and $p^1_{\mathcal{F}}(z^{ij})$ are bounded. We have

$$p^1_{\mathcal{F}}(z^{ij}) = (z^{ij}_{\mathcal{F}} - \omega E_{\mathcal{F}\mathcal{F}}(M_{\mathcal{F}} z^{ij} + q_{\mathcal{F}}))_+$$

Suppose $p^1_{\mathcal{F}}$ is not bounded. Then we get the following contradiction.

$$0 \neq \bar{p}_{\mathcal{F}} = \lim_{j\to\infty} \frac{p^1_{\mathcal{F}}(z^{ij})}{\| p^1_{\mathcal{F}}(z^{ij}) \|} = \lim_{j\to\infty} \frac{(z^{ij} - \omega E_{\mathcal{F}\mathcal{F}}(M_{\mathcal{F}} z^{ij} + q_{\mathcal{F}}))_+}{\| p^1_{\mathcal{F}}(z^{ij}) \|} = 0$$

Therefore $p^1_{\mathcal{F}}(z^{ij})$ is bounded.

Now suppose $p^k_{\mathcal{F}^c}(z^{ij})$ is not bounded. Consider the case when $k = 1$. Then we have

$$0 \neq \bar{p} = \lim_{j\to\infty} \frac{p^1_{\mathcal{F}^c}(z^{ij})}{\| p^1_{\mathcal{F}^c}(z^{ij}) \|} = -\omega E_{\mathcal{F}^c\mathcal{F}^c} K_{\mathcal{F}^c\mathcal{F}^c} \bar{p}$$

Therefore $((\omega E_{\mathcal{F}^c\mathcal{F}^c})^{-1} + K_{\mathcal{F}^c\mathcal{F}^c})\bar{p} = 0, \bar{p} \neq 0$ This contradicts the positive definiteness of $(\omega E)^{-1} + K$. Therefore $\{p^1_{\mathcal{F}^c}(z^{i_j})\}$ is bounded. The result follows for arbitrary k by induction. ∎

We are now ready to prove the main result of this section.

Theorem 4.3. *(TSOR convergence) Let M be symmetric and positive semidefinite. Either the sequence $\{z^i\}$ generated by Algorithm 4.1 terminates at a solution of the LCP or each accumulation point of $\{z^i\}$ solves the LCP.*

Proof :

The sequence $\{z^i\}$ terminates only if for some i, $p(z^i) = z^i$, in which case by Theorem 4.1, z^i solves the LCP. Suppose now $\{z^i\}$ does not terminate and the $\bar{z}$ is an accumulation point of $\{z^i\}$. Let

$$f_{\mathcal{I}_i^c}(z_{\mathcal{I}^c}) := f(z_{\mathcal{I}_i^c}, z^i_{\mathcal{I}_i})$$

and

$$f_{\mathcal{I}_i}(z_{\mathcal{I}}) := f(z^i_{\mathcal{I}_i^c}, z_{\mathcal{I}_i})$$

We have

$$\begin{aligned}
-\nabla_{\mathcal{I}_i^c} f(z^i) d^i_{\mathcal{I}_i^c} &= -\nabla_{\mathcal{I}_i^c} f(z^i)(p^k_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c}) \\
&= f_{\mathcal{I}_i^c}(z_{\mathcal{I}^c}) - f_{\mathcal{I}_i^c}(p^k_{\mathcal{I}_i^c}(z^i)) + \frac{1}{2} \parallel p^k_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c} \parallel^2_{M_{\mathcal{I}_i^c\mathcal{I}_i^c}} \\
&\geq f_{\mathcal{I}_i^c}(z_{\mathcal{I}^c}) - f_{\mathcal{I}_i^c}(p^1_{\mathcal{I}_i^c}(z^i)) + \frac{1}{2} \parallel p^k_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c} \parallel^2_{M_{\mathcal{I}_i^c\mathcal{I}_i^c}}
\end{aligned}$$

(By Lemma 4.1)

$$\geq f_{\mathcal{I}_i^c}(z^i_{\mathcal{I}^c}) - f_{\mathcal{I}_i^c}(p^1_{\mathcal{I}_i^c}(z^i))$$

(By positive semidefiniteness of M)

$$= -\Big((p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c}) + \omega E_{\mathcal{I}_i^c\mathcal{I}_i^c}\big(M_{\mathcal{I}_i^c} z^i + q_{\mathcal{I}_i^c}$$
$$+ K_{\mathcal{I}_i^c\mathcal{I}_i^c}(p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c})\big)$$
$$\times (\omega E_{\mathcal{I}_i^c\mathcal{I}_i^c})^{-1}(p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c})\Big)$$
$$- \parallel p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c} \parallel^2_{M_{\mathcal{I}_i^c\mathcal{I}_i^c}/2 - K_{\mathcal{I}_i^c\mathcal{I}_i^c} - (\omega E_{\mathcal{I}_i^c\mathcal{I}_i^c})^{-1}}$$

(By Taylor Series Expansion)

$$= \parallel p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c} \parallel^2_{(\omega E_{\mathcal{I}_i^c\mathcal{I}_i^c})^{-1} + K_{\mathcal{I}_i^c\mathcal{I}_i^c} - M_{\mathcal{I}_i^c\mathcal{I}_i^c}/2}$$

(By (4.6))

$$\geq \tilde{\gamma} \parallel p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c} \parallel^2$$

(By (4.5))

Now, we find a similar result for the second case.

$$-\nabla f_{\mathcal{I}_i}(z^i) d^i_{\mathcal{I}_i} = -\nabla f_{\mathcal{I}_i}(z^i)(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i})$$
$$= -(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i})(p^1_{\mathcal{I}_i}(z^i) - z^i_{\mathcal{I}^i})$$
$$= -\Big((p^1_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) + (\omega E_{\mathcal{I}_i\mathcal{I}_i})(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i})\Big)$$
$$\times \Big((\omega E_{\mathcal{I}_i\mathcal{I}_i})^{-1}(p^1_{\mathcal{I}_i}(z^i) - z^i_{\mathcal{I}_i})\Big)$$

$$+ \| p^1_{\mathcal{I}_i}(z^i) - z^i_{\mathcal{I}_i} \|_{(\omega E_{\mathcal{I}_i\mathcal{I}_i})^{-1}}$$

$$\geq \| p^1_{\mathcal{I}_i}(z^i) - z^i_{\mathcal{I}^i}) \|^2_{(\omega E_{\mathcal{I}_i\mathcal{I}_i})^{-1}}$$

(By Lemma 2.2. of [Mangasarian77])

$$\geq \hat{\gamma} \| p^1_{\mathcal{I}_i}(z^i) - z^i_{\mathcal{I}_i} \|^2$$

(Since E is a positive diagonal)

Therefore we have

$$\begin{aligned} -\nabla f(z^i)d^i &= -\nabla_{\mathcal{I}_i^c} f(z^i) d_{\mathcal{I}_i^c} - \nabla_{\mathcal{I}_i} f(z^i) d_{\mathcal{I}_i} \\ &\geq \tilde{\gamma} \| p^1_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c} \|^2 + \hat{\gamma} \| p^1_{\mathcal{I}_i}(z^i) - z^i_{\mathcal{I}_i} \|^2 \\ &\geq \gamma \| p(z^i) - z^i \|^2 \end{aligned}$$

where $\gamma = \min\{\tilde{\gamma}, \hat{\gamma}\}$.

Now, suppose that $\{z^{i_j}\} \to \bar{z}$. By Lemma 4.2 we have $\{p(z^{i_j})\}$ is bounded. Therefore without loss of generality we can assume $\{z^{i_j}, p(z^{i_j})\} \to (\bar{z}, \bar{p})$. We have $\bar{d} = \bar{p} - \bar{z}$. Then

$$z^{i_j} + \lambda d^{i_j} \geq 0 \text{ for } 0 \leq \lambda \leq \bar{\lambda}^{i_j}$$

where

$$\bar{\lambda}^{i_j} = \min_{k \in \mathcal{I}^c_{i_j}} \{1, z^{i_j}_k / d^{i_j}_k \mid z^{i_j}_k + d^{i_j}_k < 0\}.$$

Since $\{\bar{\lambda}^{i_j}\}$ is bounded without loss of generality we can assume that $\bar{\lambda}^{i_j} \to \bar{\lambda}$. Therefore

$$f(z^{i_j} + \lambda d^{i_j}) \geq f(z^{i_j+1}) \geq f(z^{i_j+1}) \quad \text{for } 0 \leq \lambda \leq \bar{\lambda}^{i_j}$$

and thus

$$f(\bar{z}+\lambda\bar{d}) \geq f(\bar{z}) \quad \text{for} \quad 0 \leq \lambda \leq \bar{\lambda} \tag{4.7}$$

We claim that $\bar{\lambda} \neq 0$. Since each $\mathcal{I}^c_{i_j}$ is a finite set at least one set $\mathcal{F}$ occurs infinitely often. Therefore without loss of generality assume $\mathcal{I}^c_{i_j} = \mathcal{F}$ for all j. Now, suppose $\bar{\lambda}^{ij} \to 0$. Then

$$\lim_{j\to\infty} \frac{-z_*^{ij}}{d_*^{ij}} = 0$$

where

$$\frac{-z_*^{ij}}{d_*^{ij}} = \min_{k\in\mathcal{F}}\{-z_k^{ij}/d_k | z_k^{ij} + d_k < 0\}$$

since $\{d^i\}$ is bounded then $\bar{\lambda}^{ij} \to 0$ implies that $-z_*^{ij} \to 0$ However $z_*^{ij} \in \mathcal{F}$ and therefore $z_*^{ij} > \bar{\epsilon}$ for all j by (4.2). This contradiction implies that $\bar{\lambda} \neq 0$. Therefore from (4.7) we have

$$\nabla f(\bar{z})\bar{d} \geq 0$$

Then

$$0 \geq -\nabla f(\bar{z})\bar{d} = \lim_{j\to\infty} -\nabla f(z^{ij})d^{ij} \geq \lim_{j\to\infty} \gamma \parallel p(z^{ij}) - z^{ij} \parallel^2 \geq 0$$

Therefore $\parallel p(\bar{z}) - \bar{z} \parallel^2 = 0$. By Theorem 4.1 we have $\bar{z}$ solves the LCP. ∎

We have the following special case of TSOR.

Corollary 4.1. *Let M be symmetric positive semidefinite. In addition let $K = L$ and $E = D^{-1} > 0$ where L is the strictly lower triangular part of M and D is the diagonal part of M. Then conditions (4.4) and (4.5) are satisfied for $0 < \omega < 2$.*

4.4 IMPLEMENTATION AND COMPUTATIONAL RESULTS

There are two obvious questions that come to mind when implementing the TSOR Algorithm. The first is what criterion should be used to switch from Step 1 to Step 2 of the algorithm. The second question is how do we choose k when computing $p^k_{\mathcal{I}_i^c}(z^i)$.

The criterion for switching must be such that it provides a good initial guess of the active set at a solution. The criterion we implemented checks whether the set $\mathcal{I}_i^c$ has changed every l iterations. When that set has not changed over l iterations then the algorithm proceeds to Step 2. The size of l is important. If we wait too long to make the switch we lose the full advantage of moving to Step 2. If we switch too early we may have failed to identify an appropriate set and may spend a lot of time switching sets. Since the iterations in Step 1 are much cheaper than those in Step 2 we would prefer to stay in Step 1 under these conditions. We found that checking every 10 iterations was appropriate for very sparse problems, such as less than 1% density. Checking after every 5 iterations seemed to improve results on the denser problems.

The second question involves deciding on how much work to do to find the direction in Step 2. Notice, that if M is positive definite and we solve exactly the system

$$M_{\mathcal{I}_i^c\mathcal{I}_i^c} z_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c\mathcal{I}_i} z^i_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} = 0 \tag{4.10}$$

then we are essentially using Bertsekas' method without explicitly defining D_i. However, this would involve an infinite number of SOR iterates. We could solve

(4.10) to a stringent tolerance. This would provide a good direction if we have identified the set correctly. If we have failed to identify the set correctly it may not be worthwhile to do so much work for one iteration. We suggest the following scheme

0. Start with a loose tolerance, γ^i and a stringent tolerance $\bar{\gamma}$. Define final tolerance ϵ.
1. Solve until $\| p^k_{\mathcal{I}_i^c}(z^i) - p^{k-1}_{\mathcal{I}_i^c}(z^i) \| < \gamma^i$
2. Compute z^{i+1}. If $\| (-Mz-q)_+, z(Mz+q)) \| < \epsilon$ then go to 5.
3. Identify $\mathcal{I}_{i+1}$. If $\mathcal{I}_{i+1} = \mathcal{I}_i$ then $\gamma^{i+1} = \bar{\gamma}$ else $\gamma^{i+1} = \alpha\gamma^i$ for $0 < \alpha < 1$.
4. Go to 1.
5. Stop

In addition, we have included an upper bound on the size of k to avoid too much time being spent on a given iteration.

The algorithm was tested on randomly generated LCPs. We randomly generate a matrix A and define

$$M := AA^T$$

A random solution $\bar{z}$ is generated and q is selected such that

$$\bar{z}_i > 0 \Longrightarrow q_i = -M_i\bar{z}$$

$$\bar{z}_i = 0 \Longrightarrow q_i > -M_i\bar{z}$$

The performance of TSOR is compared with the standard SOR algorithm,

$$z^{i+1} = (z^i - \omega E(Mz^i + q + K(z^{i+1} - z^i)))_+$$

where

$$E = D^{-1}, K = L, \omega > 0$$

The convergence criterion used is

$$\| (-Mz^i - q)_+, z^i(Mz^i + q) \| < .5 \times 10^{-4}$$

Tables 4.1, 4.2, 4.3,and 4.4 display the results for varying dimensions of a positive definite M. Tables 4.5, 4.6, and 4.7 display results for positive semidefinite M. The problem density indicates the fraction of non-zero elements in the matrix M. The solution density indicates the fraction of non-zero elements in the generated solution $\bar{z}$. We note that in Step 2 of the algorithm the time to complete one iteration will increase as the number of non-zero elements in the solution increases. Therefore we tested various densities of the solution vector. The columns *iter* and *time* indicate the number of iterations and the total time in seconds for convergence using the standard SOR Algorithm. The next four columns describe the results from the TSOR Algorithm applied to the same problem. The first column labeled *total iter* indicates the total number of iterations to reach convergence. The next column labeled *SOR iter* indicates the total number of SOR iterations before switching to Step 2. The column labeled *inner iter* indicates the cumulative number of SOR iterations needed to calculate the directions in Step 2 on the

algorithm. The column labeled *time* indicates the total time in seconds for the TSOR Algorithm to satisfy the convergence criterion. Finally the column labeled *factor* compares the two algorithms by measuring

$$\frac{\text{Time for SOR Algorithm}}{\text{Time for TSOR Algorithm}}$$

We make the following observations about the results for positive definite M shown in Tables 4.1, 4.2, 4.3 and 4.4.

(1) The most dramatic improvements occur when the solution density is small. The bulk of the time spent on one iteration in Step 2 is calculating $p^k_{\mathcal{I}_i^c}(z^i)$. We note that when the set $\mathcal{I}_i^c$ has few elements, determining the direction at the ith iteration is much faster. Therefore we might expect the most dramatic improvements to occur in this case.

(2) The largest number of Step 2 iterations needed for convergence was 19,and often we need less than five extra iterations. However the number of inner SOR iterations needed to generate these iterations rises as the density of the solution rises. This can be attributed to the complementarity condition which causes the system of equations

$$M_{\mathcal{I}_i^c\mathcal{I}_i^c} z_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c\mathcal{I}_i} z_{\mathcal{I}_i} = 0$$

to be of much larger dimension for problems with denser solutions. Therefore the number of SOR iterations needed to solve the system to a certain tolerance is larger than a smaller size problem.

(3) SOR is not as effective on very sparse problems as it is on denser problems. TSOR inherits some of the same difficulties as regular SOR since we are using the SOR iterate as the basis for generating the direction. However in TSOR we consider

$$M_{\mathcal{I}_i^c \mathcal{I}_i^c} z_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z^i_{\mathcal{I}_i}$$

The matrix $M_{\mathcal{I}_i^c \mathcal{I}_i^c}$ may not be as sparse as M and therefore the SOR iterate to generate the direction may be more effective. Note that when we exclude the dramatic improvements in the first two lines of Table 4.2 and the first line in Table 4.3 the improvement factor tends to be better for sparse problems.

To test for the positive semidefinite case we generated a matrix of rank $\frac{4}{5} \times n$. Tables 4.5, 4.6 and 4.7 display results from these tests. In Tables 4.5 and 4.6 we use the same stopping criterion as previously stated with the exception of problems with solution density .6 and .8 in Table 4.6. These problems were run to a tolerance of $.5 \times 10^{-3}$ and $.5 \times 10^{-2}$ respectively.

We note that although Table 4.5 seems to mirror the positive definite results, a comparison of Tables 4.6 and 4.2 reveals that the positive semidefinite case requires more second stage iterations. This can be explained by the fact that in the positive semidefinite case the LCP has infinitely many solutions and as a result there may be more changes in the active sets. In addition the resulting system of equations may not be of full rank and in fact is not even guaranteed to be consistent.

Finally, we note that the principal advantage of this method over a direct implementation of Bertsekas method is the ability to solve huge problems without storage difficulties and in a reasonable amount of time. The largest problem we have tested is a positive semidefinite LCP of dimension 10,000. Table 4.7 displays the results of that run and provides a comparison with the SOR algorithm. Bertsekas has solved special positive definite problems with 10,000 variables using his algorithm. The special nature of his problems allows him to use Riccati equations to efficiently compute the Newton direction. However, we know of no method other than TSOR, that can handle general positive semidefinite problems of this size.

Table 4.1

COMPARISON OF SOR AND TSOR
***M* POSITIVE DEFINITE: DIMENSION 1500**

		SOR		TSOR				
problem	*solution*		*time*	*total*	*SOR*	*inner*	*time*	*speedup*
density	*density*	*iter*	*seconds*	*iter*	*iter*	*iter*	*seconds*	*factor*
.03131	.25	34	91.76	19	15	28	67.36	1.3622
.03131	.5	50	138.68	29	25	19	108.86	1.2738
.03131	.7	90	250.13	45	40	38	198.68	1.2589
.03131	.8	150	419.81	60	55	86	358.46	1.1711
.00427	.25	317	167.76	34	30	200	34.85	4.8139
.00427	.5	1000*	516.50	55	40	750	179.96	2.8699
.00427	.7	1499	783.40	79	60	907	348.05	2.2508

Table 4.2

COMPARISON OF SOR AND TSOR
***M* POSITIVE DEFINITE: DIMENSION 2000**

		SOR		TSOR				
problem	*solution*		*time*	*total*	*SOR*	*inner*	*time*	*speedup*
density	*density*	*iter*	*seconds*	*iter*	*iter*	*iter*	*seconds*	*factor*
.02443	.25	1000*	3877.65	31	30	4	124.30	31.1958
.02443	.4	1000*	3884.70	41	40	5	168.92	22.9977
.02443	.6	75	297.75	43	40	22	216.43	1.3757
.02443	.7	143	565.20	74	70	62	442.02	1.2786
.02443	.8	300	1202.33	95	90	100	671.87	1.7895
.00799	.25	50	70.53	22	20	20	36.95	1.9088
.00799	.4	300	424.95	54	50	169	133.80	3.1760
.00799	.6	1000*	1426.93	59	50	401	338.10	4.2204
.00799	.8	1000*	1436.06	81	70	550	624.46	2.2996

* failed to converge after maximum iterations is reached

Table 4.3

COMPARISON OF SOR AND TSOR
***M* POSITIVE DEFINITE: DIMENSION 2500**

		SOR		TSOR				
problem density	*solution density*	*iter*	*time seconds*	*total iter*	*SOR iter*	*inner iter*	*time seconds*	*speedup factor*
.01429	.25	1000*	3711.53	31	30	3	119.63	31.0251
.01429	.4	50	188.40	32	30	32	145.83	1.2919
.01429	.6	100	379.86	44	40	37	237.56	1.5989
.01429	.7	150	570.90	51	40	124	488.05	1.1697
.01429	.8	250	955.08	77	70	174	743.12	1.2852
.00477	.25	100	142.48	22	20	49	42.65	3.3407
.00477	.5	1000	1415.73	73	70	150	175.97	8.0454
.00477	.7	1000	1437.76	98	80	807	793.97	1.8108

Table 4.4

COMPARISON OF SOR AND TSOR
***M* POSITIVE DEFINITE: DIMENSION 5000**

		SOR		TSOR				
problem density	*solution density*	*iter*	*time seconds*	*total iter*	*SOR iter*	*inner iter*	*time seconds*	*speedup factor*
.00376	.25	50	213.70	31	30	4	138.00	1.5485
.00376	.4	650	2743.88	47	40	350	501.98	5.4661
.00376	.6	450	1938.53	65	50	750	1548.47	1.2519
.00376	.8	400	1749.70	94	80	648	2210.8	.7914
.00191	.25	150	361.83	32	30	100	111.30	3.2509
.00191	.4	150	364.95	52	50	77	173.53	2.1031
.00191	.6	550	1343.76	100	80	920	1219.266	1.1021

* failed to converge after maximum iterations is reached

Table 4.5

COMPARISON OF SOR AND TSOR

***M* POSITIVE SEMIDEFINITE: DIMENSION 1000**

		SOR		TSOR				
problem	*solution*		*time*	*total*	*SOR*	*inner*	*time*	*speeup*
density	*density*	*iter*	*seconds*	*iter*	*iter*	*iter*	*seconds*	*factor*
.07106	.25	50	133.55	31	30	4	85.52	1.5616
.07106	.4	1000	2655.43	32	30	15	97.98	27.1009
.07106	.6	100	272.91	57	50	39	224.26	1.2169
.07106	.8	2000*	5509.71	121	110	453	1190.95	4.6263
.03162	.25	50	62.33	22	20	16	31.38	1.9861
.03162	.4	50	63.11	41	40	6	54.10	1.1665
.03162	.6	150	190.66	44	40	87	105.3833	1.8092
.03162	.8	1000*	1282.88	100	90	454	524.083	2.4478

Table 4.6

COMPARISON OF SOR AND TSOR

***M* POSITIVE SEMIDEFINITE: DIMENSION 2000**

		SOR		TSOR				
problem	*solution*		*time*	*total*	*SOR*	*inner*	*time*	*speedup*
density	*density*	*iter*	*seconds*	*iter*	*iter*	*iter*	*seconds*	*factor*
.01023	.25	300	522.08	51	50	42	98.21	5.3156
.01023	.4	350	611.50	65	60	214	189.78	3.2221
.01023	.6	1550	2751.82	106	90	751	725.35	3.7938
.01023	.8	4000*	7172.35	237	180	2801	3677.75	1.9502
.00393	.25	550	434.45	28	20	289	66.6166	6.5216
.00393	.4	1950	1558.08	89	50	1950	511.97	3.0433
.00393	.6	2000*	1611.56	112	80	1680	673.13	2.3941
.00393	.8	2000	1629.33	259	210	2439	1584.98	1.0279

* failed to converge after maximum iterations is reached

Table 4.7

COMPARISON OF SOR AND TSOR
***M* POSITIVE SEMIDEFINITE: DIMENSION 10000**

		SOR		TSOR				
problem density	*solution density*	*iter*	*time seconds*	*total iter*	*SOR iter*	*inner iter*	*time seconds*	*speedup factor*
.00038	.25	4000*	10480.83	111	40	3533	2024.30	5.1770
.00038	.25	1900	4969.20	78	40	1900	1132.95	4.3860
.00038	.4	2000	5205.53	107	60	2350	2215.06	2.3500
.00129	.25	10000*	61560.50	73	40	1650	1716.06	35.8731
.00129	.40	7400	45667.65	164	120	2200	3994.11	11.4337

* failed to converge after maximum iterations is reached

Chapter 5

A HYBRID ALGORITHM FOR SOLVING THE LINEAR COMPLEMENTARITY PROBLEM

5.1 INTRODUCTION

In this chapter we propose an efficient hybrid algorithm to find the exact solution of

$$\min_{z \geq 0} \ \frac{1}{2} zMz + qz$$

when M is symmetric and positive definite. This algorithm is very fast for medium-sized problems where the dimension n is of the order of 100.

The proposed algorithm is based on a projected Newton method presented by Bertsekas, [Bertsekas82]. Bertsekas presents a superlinearly convergent algorithm for solving

$$\min_{z \geq 0} \ f(z)$$

when $f(z)$ is a strictly convex twice differentiable function.

The basis for Bertsekas' method is to find a feasible descent direction and to take a step along that direction. In order to find his descent direction Bertsekas partitions z^i into two sets. We define

$$\mathcal{F}^+(z^i) = \{j | z^i_j = 0, \frac{\partial f(z^i)}{\partial z^i_j} > 0\}$$

To generate a feasible descent direction he takes a gradient step projected on the nonnegative orthant for those variables with indices in $\mathcal{F}^+(z^i)$. For the remaining variables he takes a Newton step projected on the same orthant.

A very important aspect of our proposed approach is the identification of active constraints which we will carry out by a preprocessing procedure which will greatly speed up the original algorithm proposed by Bertsekas. In particular we make the following changes in Bertsekas' algorithm. First we redefine the division of the variables as follows.

$$\mathcal{I}^+(z^i) = \{j | z_j^i = 0\}$$

This new partition saves checking the gradient at each iteration. It also provides a conservative estimate of the positive variables at the solution. Since the bulk of the work at each iteration is done calculating the Newton direction we try to make this set as small as possible.

Secondly, we note that the Newton step involves taking the inverse of a portion of the Hessian. This step is $O(m_i^3)$ where m_i is the cardinality of the complement of $\mathcal{I}^+(z^i)$ in $\{1, 2, \ldots, n\}$. In the quadratic case we see that if $\mathcal{I}^+(z^0) = \mathcal{I}^+(\bar{z})$ where $\bar{z}$ is the solution then we identify the solution in one step. However, if we have a poor estimate of $\mathcal{I}^+(z^i)$ then a number of expensive iterations may be taken before identifying the solution. Therefore, we propose an SOR algorithm as a preprocessing step. The SOR iteration is cheap and moves quickly into a neighborhood of the solution. Therefore after a finite number of SOR iterations

we might expect a good estimate of $\mathcal{I}^+(\bar{z})$, thereby insuring few Newton steps. This preprocessing step makes a very significant contribution to the speedup of the algorithm.

The final modification involves determining the step size. Bertsekas uses an Armijo–like stepsize. Because of the quadratic nature of the LCP a line search step is very cheap. Therefore we use a line search instead of the proposed Armijo–like stepsize.

In Section 5.2 we will formally state the proposed hybrid algorithm. In Section 5.3 we'll discuss convergence properties. Finally in Section 5.4 we'll present some computational results.

5.2 THE HYBRID ALGORITHM

The set $\mathcal{I}^+(z^i)$ as previously defined has an undesirable property. That is given a sequence $\{z^i\}$ of interior points that converge to a boundary point $\bar{z}$ the set $\mathcal{I}^+(z_i)$ may be strictly smaller than the set $\mathcal{I}^+(\bar{z})$. Therefore, as suggested by Bertsekas we redefine the set as follows. Let

$$w^i = \| z^i - (z^i - (Mz^i + q)_+) \|, \quad \epsilon^i = \min\{\epsilon, w^i\} \quad \text{for some } \epsilon > 0$$

$$\mathcal{I}_i := \{j | 0 \leq z_j^i \leq \epsilon^i\}$$

$$\mathcal{I}_i^c := \{j | 1 \leq j \leq n, j \notin \mathcal{I}_i\}$$

We are now ready to state the algorithm.

Algorithm 5.1

Let E be a positive diagonal matrix in $\mathbb{R}^{n\times n}$ let $z^0 \geq 0$. Let $\omega > 0$ such that $y((\omega E)^{-1} + K - \frac{M}{2})y \geq \gamma \parallel y \parallel^2$ for $\gamma > 0$.

Step 1

For $i \leq l$ where l is a positive integer

Preprocess

$$z^{i+1} = (z^i - \omega E(Mz^i + q + K(z^{i+1} - z^i)))+$$

Stop if $z^{i+1} = z^i$.

Step 2

Direction Generation

$$d^i := (p(z^i) - z^i)$$

where

$$p(z^i) := \begin{bmatrix} p_{\mathcal{I}_i}(z^i) \\ p_{\mathcal{I}_i^c}(z^i) \end{bmatrix}$$

$$p_{\mathcal{I}_i}(z^i) = (z^i_{\mathcal{I}_i} - \omega E_{\mathcal{I}_i\mathcal{I}_i}(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i}))+$$

$$p_{\mathcal{I}_i^c}(z^i) = -(M_{\mathcal{I}_i^c\mathcal{I}_i^c})^{-1}(q_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c\mathcal{I}_i} z^i_{\mathcal{I}_i})$$

stop if $d^i = 0$.

Stepsize Generation

$$z^{i+1} = z^i + \lambda^i d^i$$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda>0}\{f(z^i + \lambda d^i) | z^i + \lambda d^i \geq 0\}$$

and

$$f(z) := \frac{1}{2} zMz + qz$$

Go to step 2.

We state the following optimality conditions.

Theorem 5.1. *(Hybrid optimality conditions) Let* M, K, $E \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$, $\omega > 0$ *such that* E *is a positive diagonal matrix.*

(a) If z^i *solves the LCP and* M *has nonsingular principal minors then* $p(z^i) = z^i$.

(b) If $p(z^i) = z^i$ *then* z^i *solves the LCP.*

Proof :

(a) Suppose z^i solves the LCP. Then

$$w^i = \| z^i - (z^i - (Mz^i + q))_+ \| = 0 \Longrightarrow \epsilon^i = 0$$

Therefore

$$\mathcal{I}_i = \{j | z^i_j = 0\}$$

Then clearly, $0 = z_j^i = p_j(z^i)$ for $j \in \mathcal{I}_i$. Now $j \in \mathcal{I}_i^c \Longrightarrow M_j z^i + q_j = 0$. Since M has nonsingular principal minors $z^i_{\mathcal{I}_i^c}$ is the unique solution of

$$M_{\mathcal{I}_i^c \mathcal{I}_i^c} z^i_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z^i_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} = M_{\mathcal{I}_i^c \mathcal{I}_i^c} z^i_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} = 0$$

Therefore $p_{\mathcal{I}_i^c}(z^i) = z^i_{\mathcal{I}_i^c}$.

(b) Now suppose $p(z^i) = z^i$. We have $z_j^i = (z_j^i - \omega E_{jj}(M_j z^i + q_j))_+$ for $j \in \mathcal{I}_i$. It is easy to show that if $z_j^i > 0$ then $M_j z^i + q_j = 0$ and if $z_j^i = 0$ then $M_j z^i + q_j > 0$. We also have $M_{\mathcal{I}_i^c \mathcal{I}_i^c} z_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} = 0$ and $p_{\mathcal{I}_i^c}(z^i) = z_{\mathcal{I}_i^c} \geq 0$. Therefore z^i solves the LCP. ∎

5.3 CONVERGENCE OF THE HYBRID ALGORITHM

In this section we will show that the algorithm terminates in a finite number of steps when M is positive definite. We first show that every accumulation point solves the LCP.

Theorem 5.2. *Let E be a positive diagonal matrix in $\mathbb{R}^{n \times n}$ and $\omega > 0$. If M is symmetric and positive definite then every accumulation point solves the LCP.*

Proof :

The sequence $\{z^i\}$ terminates only if for some i, $p(z^i) = z^i$, in which case by Theorem 5.1, z^i solves the LCP. Suppose now $\{z^i\}$ does not terminate and that $\bar{z}$ is an accumulation point of $\{z^i\}$. Let

$$\mathcal{I}_i = \{j |\ 0 \leq z_j \leq \epsilon^i\}$$

We partition

$$z := [z_{\mathcal{I}_i}, z_{\mathcal{I}_i^c}]$$

Also let

$$f(z) = \frac{1}{2} zMz + qz$$

For ease of notation we shall write

$$z_{\mathcal{I}_i} := z^i_{\mathcal{I}_i}$$

and similarly

$$z_{\mathcal{I}_i^c} := z^i_{\mathcal{I}_i^c}$$

We will first demonstrate that in Step 2 we derive a feasible descent direction.

Now

$$\begin{aligned}
-\nabla_{\mathcal{I}_i} f(z^i) d^i_{\mathcal{I}_i} &= -\nabla_{\mathcal{I}_i} f(z^i)(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) \\
&= -(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i})(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) \\
&= -(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} + \omega E_{\mathcal{I}_i \mathcal{I}_i}(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i})) \\
&\times (\omega E_{\mathcal{I}_i \mathcal{I}_i})^{-1}(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) \\
&\quad + \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2_{(\omega E_{\mathcal{I}_i \mathcal{I}_i})^{-1}} \\
&\geq \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2_{(\omega E_{\mathcal{I}_i \mathcal{I}_i})^{-1}} \\
&\quad \text{(By Lemma 2.1 in [Mangasarian77]} \\
&\geq \gamma_1 \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2
\end{aligned}$$

where γ_1 is the smallest diagonal element of $(\omega E)^{-1}$.

Now consider the direction corresponding to $\mathcal{I}_i^c$. We have

$$\begin{aligned}
-\nabla_{\mathcal{I}_i^c} f(z^i) d^i_{\mathcal{I}_i^c} &= -\nabla_{\mathcal{I}_i^c} f(z^i)(p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c}) \\
&= -(M_{\mathcal{I}_i^c} z^i + q_{\mathcal{I}_i^c})(p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c}) \\
&= -(M_{\mathcal{I}_i^c} z^i + q_{\mathcal{I}_i^c})(p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c}) \\
&\qquad + (M_{\mathcal{I}_i^c \mathcal{I}_i^c} p_{\mathcal{I}_i^c}(z^i) + M_{\mathcal{I}_i^c \mathcal{I}_i} z_{\mathcal{I}_i} + q_{\mathcal{I}_i^c})(p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c}) \\
&= \| p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c} \|^2_{M_{\mathcal{I}_i^c \mathcal{I}_i^c}} \\
&\geq \gamma_2 \| p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c} \|^2
\end{aligned}$$

where γ_2 is the smallest eigenvalue of $M_{\mathcal{I}_i^c \mathcal{I}_i^c}$.

Therefore

$$-\nabla f(z^i) d^i \geq \gamma_1 \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2 + \gamma_2 \| p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c} \|^2 \geq \gamma \| p(z^i) - z^i \|^2$$

where $\gamma = \min\{\gamma_1, \gamma_2\}$.

Now, suppose that $\{z^i k\} \to \bar{z}$. We claim that $p(z^i k)$ is bounded. If it were not either $p_{\mathcal{I}_{i_k}^c}(z^i)$ or $p_{\mathcal{I}_{i_k}}(z^i)$ is unbounded. Suppose $p_{\mathcal{I}_{i_k}}(z^i)$ is unbounded. Since there are a finite number of indices we can assume without loss of generality that $\mathcal{F} := \mathcal{I}_{i_k}$ is the same for all k. We have

$$\bar{p}_{\mathcal{F}} = \lim_{k \to \infty} \frac{p_{\mathcal{F}}(z^i k)}{\| p_{\mathcal{F}}(z^i k) \|} = \lim_{k \to \infty} \frac{(z^{i_k}_{\mathcal{F}} - \omega E_{\mathcal{F}\mathcal{F}}(M_{\mathcal{F}} z^{i_k}_{\mathcal{F}} + q_{\mathcal{F}}))_+}{\| p_{\mathcal{F}}(z^i k) \|} = 0$$

But this is a contradiction since $\bar{p}_{\mathcal{F}}$ must lie on the unit ball. Now suppose that $p_{\mathcal{F}c}(z^ik)$ is unbounded. We have

$$\frac{M_{\mathcal{F}c\mathcal{F}c}p_{\mathcal{F}c}(z^ik) + M_{\mathcal{F}c\mathcal{F}}z^{i}_{\mathcal{F}}k + q_{\mathcal{F}}}{\| p_{\mathcal{F}}(z^ik) \|} = 0$$

Therefore, by taking the limit as k tends to infinity we have $M_{\mathcal{F}c\mathcal{F}c}\bar{p}_{\mathcal{F}c} = 0 \Longrightarrow \bar{p}_{\mathcal{F}c} = 0$, since M is positive definite. But this contradicts the fact that $\bar{p}_{\mathcal{F}c}$ lies on the unit ball. Therefore $p(z^ik)$ is bounded. Then without loss of generality we assume $\{z^ik, p(z^ik)\} \rightarrow (\bar{z}, \bar{p})$. This gives us $\bar{d} = \bar{p} - \bar{z}$.

Then

$$z^ik + \lambda d^ik \geq 0 \text{ for } 0 \leq \lambda \leq \bar{\lambda}^ik$$

where

$$\bar{\lambda}^ik = \min_{l \in \mathcal{F}c}\{1, \frac{-z^{i}_{l}k}{d^{i}_{l}k} | z^{i}_{l}k + d^{i}_{l}k < 0\}.$$

Since $\bar{\lambda}^ik$ is bounded without loss of generality we can assume that $\bar{\lambda}^ik \rightarrow \bar{\lambda}$. We have

$$f(z^ik + \lambda d^ik) \geq f(z^ik{}^{+1}) \geq f(z^ik{}^{+1}) \quad \text{for } 0 \leq \lambda \leq \bar{\lambda}^ik$$

Therefore

$$f(\bar{z} + \lambda\bar{d}) \geq f(\bar{z}) \quad \text{for} \quad 0 \leq \lambda \leq \bar{\lambda}$$

Suppose $\bar{\lambda} = 0$. Then for k large enough we have

$$\bar{\lambda}^ik = \frac{-z^{i}_{*}k}{d^{i}_{*}k}$$

where

$$\frac{-z_*^{i}k}{d_*^{i}k} = \min_{l \in \mathcal{F}^c} \{ \frac{-z_l^{i}k}{d_l^{i}k} | z_l^{i}k + d_l^{i}k < 0 \}$$

Now since d^ik is bounded we have $z_*^{i}k \rightarrow 0$. But $z_*^{i}k > \epsilon_{i_k}$. Therefore without loss of generality we have $\epsilon_{i_k} \rightarrow 0$. But this implies that

$$\lim_{k \rightarrow \infty} w^ik = \lim_{k \rightarrow \infty} \| z^ik - (z^ik - (Mz^ik + q)_+) \| = \| \bar{z} - (\bar{z} - (M\bar{z} + q)_+) \| = 0$$

Then $\bar{z}$ solves the LCP.

If $\bar{\lambda} > 0$ then

$$\nabla f(\bar{z})\bar{d} \geq 0$$

. Then

$$0 \geq -\nabla f(\bar{z})\bar{d} = \lim_{j \rightarrow \infty} -\nabla f(z^ik)d^ik \geq \lim_{k \rightarrow \infty} \gamma \| p(z^ik) - z^ik \|^2 \geq 0$$

Therefore $\| p(\bar{z}) - \bar{z} \|^2 = 0$. By Theorem 5.1 we have $\bar{z}$ solves the LCP. ∎

We can show the entire sequence converges using results developed in [Mangasarian77].

Theorem 5.3. *Suppose M is symmetric and positive definite then the sequence $\{z^i\}$ generated by Algorithm 5.1 converges to the unique solution of the LCP.*

Proof :

We have

$$f(z^i) \leq f(z^0) = c \quad \forall\, i$$

$$z^i \geq 0 \quad \forall\, i$$

By Lemma 2.3 in [Mangasarian77] and the positive definiteness of M we have the sequence $\{z^i\}$ is bounded. Therefore $\{z^i\}$ has an accumulation point which by Theorem 5.2 solves the LCP. Since $\bar{z}$ is the unique solution of the LCP and the sequence $\{z^i\}$ is bounded every subsequence has a unique accumulation point $\bar{z}$ which implies that the sequence converges to $\bar{z}$. ∎

To show that the algorithm converges after a finite number of iterations we need a nondegeneracy assumption at the solution. We assume

$$\bar{z} + M\bar{z} + q > 0 \tag{5.1}$$

With this assumption we state the following result.

Theorem 5.4. *Assume M is positive definite and (5.1) is satisfied at $\bar{z}$ the unique solution of the LCP. Then the algorithm identifies I and $\bar{I}$ at the solution in a finite number of steps.*

Proof :

From theorem 5.3 we have convergence of the algorithm therefore $z^i \rightarrow \bar{z}$ and $Mz^i + q \rightarrow M\bar{z} + q$. Let

$$\mathcal{I} = \{j | \bar{z}_j = 0\}$$

We have

$$w_i = \| z^i - (z^i - (Mz^i + q))_+ \|$$

For i large enough $\epsilon_i = w_i$ and ϵ_i becomes arbitrarily small. Now, since $M_j z^i + q \rightarrow M_j \bar{z} + q > 0$ for $j \in \mathcal{I}$ then we have

$$(z_j^i - (M_j z^i + q))_+ = 0$$

for large enough i. Therefore

$$z_j^i \leq w_i = \epsilon_i \quad \forall j \in I$$

Therefore $\mathcal{I}_i \supseteq \mathcal{I}$ for i large enough. Likewise, for $j \in \mathcal{I}^c$ for large enough i we have

$$z_j^i > \epsilon_i$$

Therefore for $i > K$, for some $K > 0$ we have identified the sets I and $\mathcal{I}^c$. ∎

From this we see that after identifying the appropriate set we can set those variables in the set $\mathcal{I}$ to zero and it reduces to taking a Newton step in a subspace. Since the function is quadratic this will identify the solution in one step.

5.4 COMPUTATIONAL RESULTS

We tested the hybrid SOR on randomly generated problems. We first generate a random $n \times n$ matrix A, where n is the dimension of the problem. Then we define

$$M = AA^T$$

We then generate a random solution of a prescribed density and choose q such that the random solution solves the resulting LCP.

Table 5.1

NUMBER OF ITERATIONS: HYBRID VERSUS PIVOTAL

	Hybrid		Pivotal
solution density	*Step 1*	*Step 2*	*iter*
.25	20	1	60
.50	30	1	206
.80	30	4	38
.25	20	1	99
.50	20	1	113
.80	30	1	26

Table 5.1 displays the results from a test designed to determine the importance of Step 1 of the hybrid SOR algorithm, the SOR preprocessing step. We tested problems of dimension 100 using Algorithm 5.1 as stated and using just Step 2 of Algorithm 5.1. We label this latter algorithm as a pivotal algorithm in Table 5.1. The column labeled *solution density* displays the fraction of non–zero elements in the solution vector. Under the label *Hybrid* there are two columns labeled *Step*

1 and *Step 2* which give the total number of SOR iterations and total number of pivotal iterations, respectively, to solve the problem using the hybrid algorithm. Under the label *Pivotal* we see the total number of pivotal iterations when we skip the SOR preprocessing step 1.

We note that in all but one case we needed only one pivotal iteration to find the exact solution when we used SOR preprocessing. In addition, in all but the final case the total number of iterations, SOR iterations plus pivotal iterations, is less than the total number of iterations when we skip the preprocessing step. We note that the SOR iterations are significantly cheaper than the pivotal iteration so that even in the final case it is advantageous to use the preprocessing step.

Table 5.2 displays the results from testing problems ranging from dimension 100 to dimension 800. The same problems were also solved using the SOR algorithm, that is Step 1 of Algorithm 5.1, for comparison purposes. The last two columns display time in seconds to solve the problems using the Hybrid and SOR algorithms. We terminate the SOR algorithm when the following condition is satisfied

$$\| (-Mz^i - q)_+, z^i(Mz^i + q) \|_\infty < .5 \times 10^{-4}$$

We make the following observations.

1) The hybrid SOR algorithm performs best when the solution density is small. This is because the pivotal iteration is comparatively cheaper than one in which the solution is of high density. Therefore if we know a priori that the

solution is likely to have many zeroes this algorithm is a fast effective method for determining the exact solution.

2) In only five problems, numbers 5, 6, 12, 14 and 15, are the times to solve the problem using the hybrid algorithm versus using the SOR algorithm different by a factor greater than two. Of those problems 5,6 and 14 are solved much faster by the hybrid algorithm. The SOR algorithm was significantly better in only problems 12 and 15. In both cases the solution density was high. In most cases the times were comparable. Since the hybrid algorithm provides an exact solution, this algorithm proves to be better than the SOR algorithm for medium size problems.

Finally we note that the hybrid method provides an efficient way to identify the exact solution for large LCPs when it's known that the density of the solution is likely to be small. Table 5.3 displays results of testing large problems with this property. Storage considerations alone make it impossible to apply Lemke's algorithm, [Lemke65]. The hybrid algorithm can find the solution in a matter of seconds.

Table 5.2

COMPARISON OF THE HYBRID METHOD WITH SOR

				Hybrid				**SOR**
problem number	*dimension*	*problem density*	*solution density*	*total iter*	*SOR iter*	*pivotal iter*	*time seconds*	*time seconds*
1	100	.2959	.25	21	20	1	2.40	4.08
2	100	.2959	.50	31	30	1	3.45	4.08
3	100	.2959	.80	34	30	4	17.61	16.96
4	200	.0182	.25	21	20	1	1.36	2.28
5	200	.0182	.50	21	20	1	2.51	11.55
6	200	.0182	.80	31	30	1	5.76	46.70*
7	300	.1519	.25	31	30	1	19.25	24.73
8	300	.1519	.50	31	30	1	31.91	25.15
9	300	.1519	.80	41	40	1	79.08	76.83
10	500	.0585	.25	21	20	1	17.05	29.86
11	500	.0585	.50	31	30	1	60.62	30.36
12	500	.0585	.80	81	80	1	273.25	122.93
13	800	.0086	.25	21	20	1	10.90	11.06
14	800	.0086	.50	31	30	1	46.15	209.95
15	800	.0086	.70	75	70	5	704.56	225.03

* failed to converge after maximum iterations is reached

Table 5.3

HYBRID ALGORITHM ON LARGE-SCALE LCPs

				Hybrid Method			
problem number	*dimension*	*problem density*	*solution density*	*total iter*	*SOR iter*	*pivotal iter*	*Hybrid time*
16	1000	.0094	.10	21	20	1	11.68
17	1000	.0094	.25	21	20	1	23.82
18	5000	.0008	.01	21	20	1	34.07
19	5000	.0008	.05	21	20	1	44.87
20	5000	.0008	.07	21	20	1	52.15
21	10000	.0006	.01	21	20	1	85.45
22	10000	.0006	.02	21	20	1	90.55
23	10000	.0006	.03	21	20	1	99.17

Chapter 6

A HYBRID METHOD FOR SOLVING POSITIVE SEMIDEFINITE LCPs

6.1 INTRODUCTION

In this chapter we consider the problem

$$\min_{z \geq 0} \ \frac{1}{2} z M z + q z$$

for symmetric positive semidefinite M. In Chapter 5 we described an effective hybrid algorithm for solving the problem for positive definite M. This method involved partitioning the matrix M into two subsets, $M_{\mathcal{I}_i}$ and $M_{\mathcal{I}_i^c}$ and then solving

$$M_{\mathcal{I}_i^c \mathcal{I}_i^c} z_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z^i_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} = 0 \tag{6.1}$$

When M is positive definite (6.1) is guaranteed to have a unique solution. However, if M is positive semidefinite the solution may not be unique and in fact may have no solution. Therefore, in order to apply a hybrid method to the important class of positive semidefinite matrices a new approach is necessary.

One approach is to perturb the matrix M to make it positive definite using the Tihonov regularization. This approach was suggested by [Subramanian85]. The disadvantage of this approach is that the perturbation parameter must be driven to zero which may lead to computational instability.

Another approach is to use the proximal point algorithm [Rockafellar76a] and [Rockafellar76b]. In this method we also perturb the matrix M to make it positive definite. In addition we perturb the vector q by a term depending on the previous iterate. This method also involves solving a sequence of quadratic programming problems.

We propose an iterative procedure based on the algorithm presented in Chapter 5 which does not involve solving a sequence of quadratic programming problems. At each iteration we solve

$$(M_{\mathcal{I}_i^c \mathcal{I}_i^c} + \delta^i I) z_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} - \delta^i z^i_{\mathcal{I}_i^c} = 0 \tag{6.2}$$

where $\delta^i > 0$. This problem is guaranteed to have a solution and provides us with a descent direction for the original problem. We note that (6.2) is simply a proximal point iteration on the system of equations (6.1). However, we need only take one step to determine a descent direction. In addition, there will be no restriction on δ^i other than being a positive number.

In Section 6.2 we discuss the proximal point algorithm. In Section 6.3 we will present the proposed modified proximal point algorithm. In Section 6.4 we will provide convergence results. Finally, in Section 6.5 we will present computational results.

6.2 PROXIMAL POINT ALGORIHTM

The proximal point algorithm was developed for solving general variational inequalities with a monotone operator [Rockafellar76a]. We will restrict our discussion here to the problem of solving the linear complementarity problem. For various applications of the proximal point algorithm see [Ha80],[Cheng81], and [De Leone85].

We consider the problem

$$\min_{z \geq 0} \ \frac{1}{2} zMz + qz \tag{6.3}$$

In Chapter 5 we discussed a method that finds an exact solution when M is symmetric and positive definite. However, when M is positive semidefinite algorithm 5.1 is not guaranteed to converge. We propose applying the proximal point algorithm to overcome this difficulty. Having the point z^i, the proximal point algorithm finds a new point z^{i+1} which solves the following problem

$$\min_{z \geq 0} \ f(z, z^i) = \frac{1}{2} zMz + qz + \frac{\delta^i}{2} \parallel z - z^i \parallel^2 \tag{6.4}$$

where $\{\delta^i\}$ is a sequence of real numbers bounded away from zero. We note that since $f(z, z^i)$ is strongly convex then problem (6.4) has a unique solution.

The equivalent LCP is as follows. Find $z \in \mathbb{R}^n$ such that

$$z \geq 0$$

$$(M + \delta^i I)z + q - \delta^i z^i \geq 0$$

$$z((M + \delta^i I)z + q - \delta^i z^i = 0$$

Since $M + \delta^i I$ is positive definite we propose using Algorithm 5.1 to find each successive iterate in the proximal point algorithm. We now formally state the proposed proximal point algorithm.

Algorithm 6.1.

1. Let $z^0 = 0$
2. Let z^{i+1} be the unique optimal solution of problem 6.4
3. Stop if $z^{i+1} = z^i$ else go to 2

From [Rockafellar76] we have the following convergence theorem.

Theorem 6.1. *Assume (6.3) has a solution, then the sequence* $\{z^i\}$ *generated by Algorithm 6.1 converges to a solution of (6.3).*

6.3 THE MODIFIED HYBRID ALGORITHM

In this section we state a variation of Algorithm 5.1, as described in Chapter 5, for positive semidefinite LCPs. As in Chapter 5 we define the sets as follows.

$$w^i = \| z^i - (z^i - (Mz^i + q))_+ \|, \ \epsilon^i = \min\{\epsilon, w^i\} \quad \text{for some } \epsilon > 0$$

$$\mathcal{I}_i := \{j | 0 \leq z_j^i \leq \epsilon^i\}$$

$$\mathcal{I}_i^c := \{j | 1 \leq j \leq n, j \notin \mathcal{I}_i\}$$

We are now ready to state the algorithm.

Algorithm 6.2

Let E be a positive diagonal matrix in $\mathbb{R}^{n\times n}$ let $z^0 \geq 0$. Let $\omega > 0$ such that $y((\omega E)^{-1} + K - \frac{M}{2})y \geq \gamma \parallel y \parallel^2$ for $\gamma > 0$ and $\forall y$.

Step 1

For $i \leq l$ where l is a positive integer

$$z^{i+1} = (z^i - \omega E(Mz^i + q + K(z^{i+1} - z^i)))+$$

Stop if $z^{i+1} = z^i$.

Step 2

Direction Generation

Let $\delta^i > \bar{\delta} > 0$

$$d^i := (p(z^i) - z^i)$$

where

$$p(z^i) := \begin{bmatrix} p_{\mathcal{I}_i}(z^i) \\ p_{\mathcal{I}_i^c}(z^i) \end{bmatrix}$$

$$p_{\mathcal{I}_i}(z^i) = (z_{\mathcal{I}_i} - \omega E_{\mathcal{I}_i\mathcal{I}_i}(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i}))+$$

$$p_{\mathcal{I}_i^c}(z^i) = -(M_{\mathcal{I}_i^c\mathcal{I}_i^c} + \delta^i I)^{-1}(q_{\mathcal{I}_i^c} - \delta^i z_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c\mathcal{I}_i} z_{\mathcal{I}_i})$$

Stop if $d^i = 0$.

Stepsize Generation

$$z^{i+1} = z^i + \lambda^i d^i$$

where

$$f(z^i + \lambda^i d^i) = \min_{\lambda>0}\{f(z^i + \lambda d^i) | z^i + \lambda d^i \geq 0\}$$

where

$$f(z) := \frac{1}{2} zMz + qz$$

Go to Step 2.

We state the following optimality conditions.

Theorem 6.2. *(Modified Hybrid Optimality Conditions) Let* M, K, $E \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$, $\omega > 0$ *such that* E *is a positive diagonal matrix.*

(a) If z^i *solves the LCP and* M *is positive semidefinite, then* $p(z^i) = z^i$.

(b) If $p(z^i) = z^i$, *then* z^i *solves the LCP.*

Proof :

(a) Suppose z^i solves the LCP. Then

$$w^i = \| z^i - (z^i - (Mz^i + q))_+ \| = 0 \Longrightarrow \epsilon^i = 0$$

Therefore

$$\mathcal{I}_i = \{i | z^i_j = 0\}$$

Then clearly, $0 = z_j = p_j(z^i)$ for $j \in \mathcal{I}_i$. Now $j \in \mathcal{I}_i^c \Longrightarrow M_j z^i + q_j = 0$. Therefore

$$M_{\mathcal{I}_i^c} z^i_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} = M_{\mathcal{I}_i^c \mathcal{I}_i^c} z^i_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} = 0$$

Now by Algorithm 6.2 direction generation

$$(M_{\mathcal{I}_i^c \mathcal{I}_i^c} + \delta^i I) p_{\mathcal{I}_i^c}(z^i) + q_{\mathcal{I}_i^c} - \delta^i z_{\mathcal{I}_i^c} = 0$$

and hence $p_{\mathcal{I}_i^c}(z^i)$ is unique. But we also have

$$0 = M_{\mathcal{I}_i^c} z^i_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} = (M_{\mathcal{I}_i^c \mathcal{I}_i^c} + \delta^i I) z^i_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} - \delta^i z_{\mathcal{I}_i^c}$$

Therefore $p_{\mathcal{I}_i^c}(z^i) = z_{\mathcal{I}_i^c}$.

(b) Now suppose $p(z^i) = z^i$. We have $z^i_j = (z^i_j - \omega E_{jj}(M_j z^i + q_j))_+$ for $j \in I_i$. It is easy to show that if $z^i_j > 0$ then $M_j z^i + q_j = 0$ and if $z^i_j = 0$ then $M_j z^i + q_j \geq 0$. We also have from $p(z^i) = z^i$ that

$$0 = (M_{\mathcal{I}_i^c \mathcal{I}_i^c} + \delta^i I) z_{\mathcal{I}_i^c} + q_{\mathcal{I}_i^c} - \delta^i z_{\mathcal{I}_i^c} = M_{\mathcal{I}_i^c \mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z_{\mathcal{I}_i} + q_{\mathcal{I}_i^c}$$

and $p_{\mathcal{I}_i^c}(z^i) = z_{\mathcal{I}_i^c} \geq 0$. Therefore z^i solves the LCP. ∎

6.4 CONVERGENCE OF THE MODIFIED HYBRID ALGORITHM

The following theorem will show that every accumulation point of Algorithm 6.2 solves the LCP. In addition the proof of this theorem demonstrates that every Step 2 iteration of Algorithm 6.2 provides a descent direction for the original problem. This is in contrast to the approach suggested in Section 6.2. In this

algorithm the Step 2 iterations of Algorithm 5.1 are inner iterations. Each of these iterations provide a descent direction for the current perturbed quadratic program which may not be a descent direction for the original problem. We are now ready to state the convergence theorem for Algorithm 6.2.

Theorem 6.3. *(Convergence) Let E be a positive diagonal matrix in $\mathbb{R}^{n\times n}$ and $\omega > 0$. If M is symmetric and positive semidefinite then every accumulation point of Algorithm 6.2 solves the LCP.*

Proof :

The sequence $\{z^i\}$ terminates only if for some i, $p(z^i) = z^i$, in which case by Theorem 6.2, z^i solves the LCP. Suppose now $\{z^i\}$ does not terminate and that $\bar{z}$ is an accumulation point of $\{z^i\}$. Let

$$\mathcal{I}_i = \{j | \ 0 \leq z^i_j \leq \epsilon^i\}$$

We partition

$$z := [z_{\mathcal{I}_i}, z_{\mathcal{I}_i^c}]$$

We will first demonstrate that in Step 2 we derive a feasible descent direction. For convenience of notation we write

$$z_{\mathcal{I}_i} = z^i_{\mathcal{I}_i}$$

and similarly

$$z_{\mathcal{I}_i^c} = z^i_{\mathcal{I}_i^c}$$

Now

$$
\begin{aligned}
-\nabla_{\mathcal{I}_i} f(z^i) d^i_{\mathcal{I}_i} &= -\nabla_{\mathcal{I}_i} f(z^i)(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) \\
&= -(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i})(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) \\
&= -(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} + \omega E_{\mathcal{I}_i \mathcal{I}_i}(M_{\mathcal{I}_i} z^i + q_{\mathcal{I}_i})) \\
&\quad \times (\omega E_{\mathcal{I}_i \mathcal{I}_i})^{-1}(p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i}) + \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2_{(\omega E_{\mathcal{I}_i \mathcal{I}_i})^{-1}} \\
&\geq \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2_{(\omega E_{\mathcal{I}_i \mathcal{I}_i})^{-1}} \\
&\qquad \text{(By Lemma 2.1 in [Mangasarian77])} \\
&\geq \gamma \| p_{\mathcal{I}_i}(z^i) - z_{\mathcal{I}_i} \|^2
\end{aligned}
$$

for γ the smallest diagonal element of $(\omega E)^{-1}$.

Now consider the direction corresponding to $\mathcal{I}_i^c$. We have

$$
\begin{aligned}
-\nabla_{\mathcal{I}_i^c} f(z^i) d^i_{\mathcal{I}_i^c} &= -\nabla_{\mathcal{I}_i^c} f(z^i)(p_{\mathcal{I}_i^c}(z^i) - z^i_{\mathcal{I}_i^c}) \\
&= -(M_{\mathcal{I}_i^c \mathcal{I}_i^c} z^i_{\mathcal{I}_i^c} + M_{\mathcal{I}_i^c \mathcal{I}_i} z_{\mathcal{I}_i} + q_{\mathcal{I}_i^c})(p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c}) \\
&\quad + \left((M_{\mathcal{I}_i^c \mathcal{I}_i^c} + \delta^i I) p_{\mathcal{I}_i^c}(z^i) + M_{\mathcal{I}_i^c \mathcal{I}_i} z_{\mathcal{I}_i} + q_{\mathcal{I}_i^c} - \delta^i z_{\mathcal{I}_i^c}\right) \\
&\quad \times (p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c}) \\
&= \| p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c} \|^2_{M_{\mathcal{I}_i^c \mathcal{I}_i^c} + \delta^i I} \\
&= \delta^i \| p_{\mathcal{I}_i^c}(z^i) - z_{\mathcal{I}_i^c} \|^2
\end{aligned}
$$

Now suppose that $\{z^i k\} \to \bar{z}$. We claim that $p(z^i k)$ is bounded. If not then either $p_{\mathcal{I}^c_{i_k}}(z^i k)$ or $p_{\mathcal{I}_{i_k}}(z^i k)$ is unbounded. Since there are a finite number of indices we can assume without loss of generality that $\mathcal{F} = \mathcal{I}_{i_k}$ for all k.

We have

$$\bar{p}_{\mathcal{F}} = \lim_{k\to\infty} \frac{p_{\mathcal{F}}(z^i k)}{\| p_{\mathcal{F}}(z^i k) \|} = \lim_{k\to\infty} \frac{(z^{i}_{\mathcal{F}} k - \omega E_{\mathcal{F}\mathcal{F}}(M_{\mathcal{F}} I z^i k + q_{\mathcal{F}}))+}{\| p_{\mathcal{F}}(z^i k) \|} = 0$$

But this contradicts the fact that $\bar{p}_{\mathcal{F}}$ lies on the unit ball. Now suppose that $p_{\mathcal{F}c}(z^i k)$ is unbounded. We have

$$\frac{(M_{\mathcal{F}c\mathcal{F}c} + \delta^i k I)p_{\mathcal{F}c}(z^i k) + M_{\mathcal{F}c\mathcal{F}} z^{i}_{\mathcal{F}} k + q_{\mathcal{F}c} - \delta^i k z^{i}_{\mathcal{F}c} k}{\| p_{\mathcal{F}c}(z^i k) \|} = 0$$

Taking limits we have $(M_{\mathcal{F}c\mathcal{F}c} + \bar{\delta} I)\bar{p}_{\mathcal{F}c} = 0 \Longrightarrow \bar{p}_{\mathcal{F}c} = 0$, since M is positive semidefinite. But this contradicts the fact that $\bar{p}_{\mathcal{F}c}$ lies on the unit ball. Therefore $p(z^i k)$ is bounded.

Without loss of generality we assume $\{z^i k, p(z^i k)\} \to (\bar{z}, \bar{p})$. This gives us $\bar{d} = \bar{p} - \bar{z}$.

Then

$$z^i k + \lambda d^i k \geq 0 \text{ for } \; 0 \leq \lambda \leq \bar{\lambda}^i k$$

where

$$\bar{\lambda}^i k = \min_{l \in \mathcal{F}c} \{1, \frac{-z^i k}{d^{i}_{l} k} | z^{i}_{l} k + d^{i}_{l} k < 0\}.$$

Since $\bar{\lambda}^i k$ is bounded, without loss of generality we can assume that $\bar{\lambda}^i k \to \bar{\lambda}$. We have

$$f(z^i k + \lambda d^i k) \geq f(z^i k^{+1}) \geq f(z^i k{+1}) \quad \text{for } 0 \leq \lambda \leq \bar{\lambda}^i k$$

Therefore

$$f(\bar{z} + \lambda \bar{d}) \geq f(\bar{z}) \quad \text{for} \quad 0 \leq \lambda \leq \bar{\lambda}$$

Suppose that $\bar{\lambda} = 0$. Then for k large enough we have

$$\bar{\lambda}^i k = \frac{-z_*^i k}{d_*^i k}$$

where

$$\frac{-z_*^i k}{d_*^i k} = \min_{l \in \mathcal{F}^c} \{ \frac{-z_l^i k}{d_l^i k} | z_l^i k + d_l^i k < 0 \}$$

But $z_*^i k > \epsilon^i k$. Therefore without loss of generality we have $\epsilon^i k \to 0$. But this implies that

$$\lim_{k \to \infty} w^i k = \lim_{k \to \infty} \| z^i k - (z^i k - (M z^i k + q)_+) \| = \| \bar{z} - (\bar{z} - (M\bar{z} + q)_+) \| = 0$$

Then $\bar{z}$ solves the LCP.

If $\bar{\lambda} > 0$ then

$$\nabla f(\bar{z}) \bar{d} \geq 0$$

We let

$$\alpha^i k = \min\{\delta^i k, \gamma\}$$

Then since $\alpha^i k$ is bounded we can assume without loss of generality that $\alpha^i k \to \bar{\alpha} > 0$. Then

$$0 \geq -\nabla f(\bar{z}) \bar{d} \geq \lim_{j \to \infty} \alpha^i k \| p(z^i k) - z^i k \| = \bar{\alpha} \| p(\bar{z}) - \bar{z} \| \geq 0$$

Thus $p(\bar{z}) = \bar{z}$, and by Theorem 6.2 $\bar{z}$ solves the LCP. ∎

6.5 COMPUTATIONAL RESULTS

We tested Algorithm 6.1, the proximal point algorithm, and Algorithm 6.2, the modified hybrid algorithm, on random positive semidefinite LCPs. The problems were generated to have rank $\frac{4}{5} \times n$, where n is the dimension of the problem. For the proximal point algorithm we used a constant parameter of $\delta^i = .001$, which experimentally proved to be a good choice. We used Algorithm 5.1 of Chapter 5 to solve for each iterate.

In the modified hybrid algorithm we start with a parameter $\delta^i = 10^{-5}$. We then update δ^i as follows.

$$If \quad \mathcal{I}_{i+1} = \mathcal{I}_i \quad then \quad \delta^{i+1} = \max(.1\delta^i, 10^{-10})$$

Our stopping criterion for both methods was

$$\| (-Mz^i - q)_+, z^i(Mz^i + q) \|_\infty < .5 \times 10^{-4}$$

Table 6.1 displays the results of tests on twenty problems run on a VAXstation II. Tests were done for varying problem dimension, density and solution density. Under the heading Proximal Point we have two columns labeled *outer iter* and *inner iter*. The former is the number of proximal point iterations. The latter is the number of Algorithm 5.1 iterations. The last column under this heading gives the total time in seconds to reach the stopping criterion. The results for Algorithm 6.2 are displayed under the heading Modified Hybrid. The column labeled SOR

iter displays the number of SOR preprocessing iterations. The second column labeled *piv iter* displays the number of Step 2 iterations. Finally, the last column gives time in seconds.

We see from Table 6.1 that the modified hybrid algorithm is consistently faster than the proximal point algorithm. We also note in half of the problems run only a single costly step 2 pivotal iteration is needed when using the modified hybrid algorithm. This seems to indicate that the direction obtained by solving the perturbed system of equations provides us with a good descent direction for the original problem.

Table 6.1

PROXIMAL POINT ALGORITHM VERSUS THE MODIFIED HYBRID ALGORITHM

				Proximal Point			Modified Hybrid		
prob	*prob*	*prob*	*sol*	*outer*	*inner*	*time*	*SOR*	*piv*	*time*
#	*dim*	*density*	*density*	*iter*	*iter*	*seconds*	*iter*	*iter*	*seconds*
1	100	.3875	.25	2	41	10.06	20	1	2.93
2	100	.3875	.50	2	50	10.61	30	1	4.76
3	100	.3875	.75	6	132	78.86	20	10	38.21
4	100	.0729	.25	3	71	7.00	20	2	1.56
5	100	.0729	.50	24	292	37.43	30	1	1.58
6	100	.0729	.75	5	162	51.93	40	11	22.06
7	200	.3792	.25	2	41	43.10	20	1	14.25
8	200	.3792	.50	2	61	66.76	40	1	30.00
9	200	.3792	.75	3	111	199.23	70	4	114.15
10	200	.0393	.25	4	61	15.63	20	1	2.16
11	200	.0393	.50	30	372	205.68	50	3	13.50
12	200	.0393	.75	500*	5142	7382.43	40	10	77.15
13	300	.3577	.25	2	50	107.93	40	1	59.73
14	300	.3577	.50	2	61	172.73	40	1	75.21
15	300	.3577	.75	4	121	936.03	50	6	427.43
16	300	.0427	.25	2	51	32.03	30	1	5.35
17	300	.0427	.50	10	191	406.53	50	8	101.26
18	300	.0427	.75	14	232	831.15	60	6	154.78
19	400	.0347	.25	4	61	64.26	20	1	7.58
20	400	.0347	.50	17	221	632.23	50	3	83.20

* failed to converge after maximum iterations is reached

Chapter 7

COMPARATIVE COMPUTATIONAL RESULTS

7.1 INTRODUCTION

In this chapter we will compare computational experience by using the various algorithms proposed in this thesis. These tests will be performed on positive semidefinite symmetric LCPs of medium–sized dimension. We recall that in Chapter 2 a parallel version of an SOR algorithm for solving LCPs was presented. Here we use the serial version of the algorithm, where the substitution operator K is chosen to be the strictly lower triangular part of M

In Chapter 3 we presented a parallel version of an algorithm due to Lemke. This algorithm is currently widely used for solving LCPs. We use a serial implementation of the Algorithm coded in Fortran 77 .

In Chapter 4 we proposed a two–stage SOR algorithm (TSOR) designed to solve large sparse symmetric positive semidefinite LCPs. In Chapter 5 we proposed a finite algorithm to solve positive definite symmetric LCPs. Although this algorithm is not guaranteed to converge for positive semidefinite LCPs we will test the algorithm to see how often it converges. This algorithm is actually a hybrid SOR/Newton algorithm. Here we will call the algorithm *Hybrid.* Finally, in Chapter 6 we modified the hybrid algorithm to solve positive semidefinite symmetric LCPs. We will denote this algorithm *Modhyb.*

In Section 7.2 we will describe the specifics of the tests we ran. In Section 7.3 we will present our results. Finally, in Section 7.4 we will suggest some further topics of research.

7.2 TESTING SPECIFICATIONS

We tested the algorithms on random positive semidefinite symmetric LCPs. The matrix M was generated so that it had rank $\frac{4}{5} \times n$ where n is the dimension of the matrix M. A random vector was generated and q was chosen so that a prescribed random vector solves the LCP.

We identified several factors that might affect the relative performance of the algorithm. Those included

1) *problem dimension* n

2) *problem density* (fraction of nonzeros in M)

3) *prescribed solution density* (fraction of nonzeros in the solution)

Thus, our tests involved varying these factors one at a time while holding the other factors fixed.

All algorithms were coded using Fortran 77 and were run on the VAXworkstation II. All floating point operations were done in double precision which provides about 16 figure decimal accuracy. We note that Lemke's algorithm provides an exact solution. The rest of the algorithms were run until the following criterion was reached

$$\| (-Mz^i - q)_+, z^i(Mz^i + q) \|_\infty < .5 \times 10^{-4}$$

In all the algorithms presented in this thesis we use the same parameter specifications as well as any heuristics presented in the previous chapters.

7.3 PERFORMANCE RESULTS

Table 7.1 displays results when we held problem dimension and problem density fixed at 100 and .07 respectively. We then allowed the generated solution density to vary between .25 and .75. We display CPU time in seconds.

Table 7.1
EFFECT OF SOLUTION DENSITY ON ALGORITHM TIME

solution density	*Hyb time*	*Modhyb time*	*SOR time*	*TSOR time*	*Lemke time*
.25	1.45*	1.56	7.00	2.01	14.43*
.25	.93*	1.03	1.40	.70	12.43*
.25	.92*	1.02	8.43	1.72	12.82*
.50	1.55*	1.58	19.53	4.48	20.38*
.50	2.63*	1.88	4.18	5.98	25.07*
.50	1.23*	1.35	1.48	.95	19.56*
.75	f	22.06	52.56	87.98	34.03*
.75	11.45	5.15	265.68	176.63	32.78*
.75	7.30	11.90	365.88	115.55	33.95*

* obtained an exact solution

f– failed because tried to solve an inconsistent system of equations

We make the following observations.

1) In all methods the time to solve the problem increases as the solution density increases.

2) When the solution density is small relative to the rank of the matrix the hybrid algorithm tends to identify the exact solution quickly. However, as the solution density increases the modified hybrid algorithm is more reliable.

3) The SOR and TSOR algorithms performance degenerates significantly when the solution density increases.

4) Excluding the case when the hybrid algorithm failed we see that the hybrid and modified hybrid algorithm consistently find the solution faster than Lemke's algorithm. However, we note that the gap between the times for Lemke and hybrid and modified hybrid narrows as the solution density rises.

Table 7.2 displays results when we held problem dimension and generated solution density fixed at 100 and .5 respectively. We then allowed the problem density to vary between .03 and .70. We display the times to reach the convergence criterion in seconds.

Table 7.2

EFFECT OF PROBLEM DENSITY ON ALGORITHM TIME

problem density	*Hyb time*	*Modhyb time*	*SOR time*	*TSOR time*	*Lemke time*
.03	1.16*	1.20	.87	.70	19.82*
.03	1.00*	1.15	.90	.50	18.88*
.03	1.02*	1.70	F	2.45	20.00*
.38	6.55*	6.48	12.27	10.00	27.55*
.38	3.82*	3.86	6.10	5.18	23.25*
.38	4.73*	4.68	5.83	4.56	22.37*
.70	8.13*	8.03	23.38	14.05	29.10*
.70	9.96	9.80	12.22	11.66	26.67*
.70	7.93	7.78	12.08	10.05	25.05*

* obtained an exact solution

F– failed to converge after reaching maximum iterations

The Lemke algorithm implementation uses no sparsity–preserving techniques. Therefore as the problem density rises the time to solve does not change appreciably. However, the time for the remaining algorithms rises significantly as the density increases. The SOR algorithm times rise most rapidly while the hybrid and modified hybrid algorithm times rise most slowly. This can be attributed to the fact that the SOR algorithm only uses non–zero elements in its floating point calculations. Therefore at each iteration there are approximately $d \times n^2$ arithmetic operations where d is the density of M. The hybrid and modified hybrid algorithms follow this pattern in the preprocessing step but then require $O(m_i^3)$ steps, where m_i is the cardinality of $\mathcal{I}_i^c$, regardless of the density of M.

We also point out that although the gap between the time to solve the problems using the hybrid algorithm and the time to solve using Lemke's algorithm narrows as the density grows the hybrid algorithm outperforms Lemke's algorithm at all densities tested. We add that even if sparsity preserving techniques were employed in Lemke's algorithm unless the original matrix had some special structure one would expect the pivoted matrix to fill up after several pivots.

Table 7.3 displays results when we held problem density and generated solution density fixed at .07 and .5 respectively. We then allowed the problem dimension to vary between 100 and 400. We display the times to reach the convergence criterion in seconds.

Table 7.3

EFFECT OF PROBLEM DIMENSION ON ALGORITHM TIME

problem dimension	*Hyb time*	*Modhyb time*	*SOR time*	*TSOR time*	*Lemke time*
100	4.77*	2.95	7.06	3.33	23.02*
100	3.85*	5.56	55.50	23.90	20.98*
100	1.71*	2.66	23.33	5.90	22.93*
250	11.48*	26.36	F	98.05	364.16*
250	10.58*	29.60	F	38.11	340.10*
250	14.92*	15.15	15.93	9.77	387.57*
400	35.45*	56.80	46.55	22.90	1377.00*
400	35.36	66.36	65.20	38.25	1654.38*
400	46.98	80.78	130.75	38.66	1608.58*

* obtained an exact solution

F– failed to converge after reaching maximum iterations

We make the following observations.

1) TSOR seems to be more robust than SOR. We note that in two cases SOR failed to reach the desired tolerance before reaching the prescribed maximum iteration. However, TSOR was able to solve these problems in a reasonable amount of time.

2) We note that although the modified hybrid algorithm is very close to the hybrid algorithm in the smaller dimensional problems, as we reach the larger dimensional problems the hybrid algorithm is more effective. We expect this is because more Newton iterations are required in the modified hybrid algorithm because of the perturbation term in this algorithm. As the dimension rises this may add significant computing time.

3) Lemke's algorithm on average tends to be of $O(n^3)$. The Newton step in the hybrid algorithm is of order m_i^3 where m_i is the cardinality of $\mathcal{I}_i^c$. Thus for $n >> m_i$ we would expect a much better performance of the hybrid algorithm. The preprocessing step in the hybrid algorithm, consisting of a relatively small number of SOR sweeps, is of order n^2. The overhead due to the preprocessing step is proportionately larger when the dimension is smaller. This may explain the relatively better performance of the hybrid algorithm compared with Lemke's algorithm when n is larger.

7.4 FURTHER RESEARCH

In this section we describe some directions of further research based on the results of this thesis.

1) In Chapter 2 we described an asynchronous parallel successive overrelaxation algorithm for solving symmetric positive semidefinite LCPs. We obtained some success in speeding up the GPSOR algorithm, [Mangasarian & De Leone86], by using a combination of multiple sweeps and single sweeps. However, we have not identified the best combination, if one exists. A research topic might be to develop a more sophisticated criterion for determining the number of sweeps at a given iteration. We suggest this criterion might be based on the relative rate of improvements of the objective function at a given iteration.

2) In Chapter 3 we described an effective parallel distribution of Lemke's algorithm. Bhide, [Finkel etal86] began some work using this idea to parallelize

the simplex algorithm. We note that in Lemke's algorithm at each iteration there is a unique choice of the pivot column. However, in the simplex algorithm there may be multiple choices for the pivot column. In serial codes various strategies have been developed in choosing the pivot column. These strategies can greatly enhance the algorithm. It would be interesting to develop parallel counterparts of these strategies. These strategies must balance the algorithmic improvements due to a good pivot column choice with the extra communication costs which are likely to occur.

3) In Chapters 4, 5 and 6 we presented three related algorithms. The idea of these algorithms is to partition the current estimate of the solution into two sets, $z^i_{\mathcal{I}_i}$ and $z^i_{\mathcal{I}_i^c}$ where the set $\mathcal{I}_i$ is the set of indices in which $z_{\mathcal{I}_i}$ is near zero and the set $\mathcal{I}_i^c$ is the remaining indices. The idea of these algorithms is to concentrate the work on updating $z_{\mathcal{I}_i^c}$, while taking a very simple step to update $z_{\mathcal{I}_i}$. In Algorithms 5.1 and 6.2 the update of $z_{\mathcal{I}_i^c}$ involves solving a system of equations. In Algorithm 4.1 we make multiple SOR iterations to determine an appropriate direction. Therefore extra work is done to determine a good direction over a certain set of variables. We note that if the set of positive variables is small this can result in an extremely fast algorithm. In fact we demonstrated in Chapter 5 that a certain class of very large problems, (10,000 variables) could be solved exactly in a matter of seconds. Because of the complementary nature of the problem we know that if the solution density is large then the dual variables

have a small solution density. Therefore, it would be interesting to develop a dual version of these algorithms which would then take advantage of cases where the solution density is high.

4) In Chapter 6 we avoided the difficulties of the hybrid algorithm applied to a positive semidefinite problem by using a proximal point perturbation on the system of equations that we solved at each iteration. We found that one step of the proximal point iteration gave us a descent direction for the original problem. An interesting question is whether a proximal point perturbation applied to TSOR may help stabilize the algorithm when applied to a positive semidefinite LCP. In other words, in the TSOR algorithm we identify a certain set of variables and perform multiple SOR iterations without the projection on this set of variables. We are actually solving a system of equations inexactly using the SOR iteration. However, as in Chapter 6 when M is positive semidefinite this system is not guaranteed to have a solution. The SOR iteration will not be stable in this case. Therefore it may take a long time to solve to a given tolerance and yet fail to provide a good descent direction to justify this extra work. If we perturb the system we are guaranteed a solution to the system of equations. Therefore it would be interesting to determine if we can perform multiple SOR iterations on the perturbed set of equations and still obtain a descent direction for the original problem.

Bibliography

Agrawal, R. and Jagadish, H. V. (1986) Parallel Computation on Loosely–Coupled Workstations, Computer Technology Research Laboratory Technical Report, AT&T Bell Laboratories.

Barlow, R. H. and Evans, D. J. (1982) Parallel Algorithms for the Iterative Solution to Linear Systems, The Computer Journal, Vol. 25, No. 1,56-60

Bertsekas, D. P. (1982) Projected Newton Methods for Optimization Problems with Simple Constraints, SIAM J. Control and Optimization, 20,221-246.

Chang, M. D. (1986) A Parallel Primal Simplex Variant for Generalized Networks, Department of General Business, University of Texas at Austin.

Chen, R. J.(1987) Parallel Algorithms for a Class of Convex Optimization Problems, Computer Sciences Department, University of Wisconsin–Madison.

Cheng, Y. C. (1981) Iterative Methods for Solving Linear Complementarity and Linear Programming Problems, Computer Sciences Department, University of Wisconsin–Madison.

Conrad, V. and Wallach, Y. (1977) Iterative Solution of Linear Equations on a Parallel Processor System, IEEE Transactions on Computers, C-26,9,838-847.

Cryer, C. W. (1971) The Solution of a Quadratic Programming Problem using Systematic Overrelaxation, SIAM J. Control 9,385-392.

De Leone, R. (1985) Sequential Overrelaxation SOR For Large Sparse Linear Programs, Internal Report CRAI.

DeWitt, D. J., Finkel, R. and Solomon, M. (1984) The CRYSTAL Multicomputer: Design and Implementation Experience, Computer Sciences Technical Report #553, University of Wisconsin–Madison.

Feijoo, B. (1985) Piecewise–Linear Approximation Methods and Parallel Algorithms in Optimization, Computer Sciences Technical Report #598, University of Wisconsin–Madison.

Finkel, R., Barzideh, B., Bhide, C. W., Lam, M.O., Nelson, D. Polisetty, R., Rajaraman, S., Steinberg, I., Venkatesh, G. A. (1986) Experience with Crystal, Charlotte and Lynx Second Report, Computer Sciences Technical Report #649, University of Wisconsin–Madison.

Gill, P. E. and Murray, W. (1974) Methods for Large–scale Linearly Constrained Problems. *In* "Numerical Methods for Constrained Optimization" (Gill, Murray, eds.), 93-148. Academic Prss, London and New York.

Goldfarb, D. (1972) Extensions of Newton's Method and Simplex Methods for Solving Quadratic Programs, *In* "Numerical Methods for Non-Linear Optimization" (F. A. Lootsma, ed.), 239-254. Academic Press, London and New York.

Ha, C. D. (1980) Decomposition Methods for Structured Convex Programming, Industrial Engineering Department, University of Wisconsin–Madison.

Hogan, W. W. (1975) Energy Policy Models for Project Independence, Comput. & Ops Res., 2, 251-271.

Hoyle, S. C. (1986) A Single-Phase Method for Quadratic Programming, Systems Optimization Laboratory Technical Report #86-9, Stanford University.

Lenard, M. L. (1979) A Computational Study of Active Set Strategies in Nonlinear Programming with Linear Constraints, Math. Prog. 16, 81-97.

Lemke, C. E. (1965) Bimatrix Equilibrium Points and Mathematical Programming, Management Science, 11, 681-689.

Mangasarian, O. L. (1977) Solution of Symmetric Linear Complementarity Problems by Iterative Mehods, Journal of Optimization Theory and Applications, 22,465-484.

Mangasarian, O. L. (1978) Characterization of Linear Complementarity Problems as Linear Programs, Mathematical Programming Study 7, North Holland, Amsterdam, The Netherlands, 74–87.

Mangasarian, O. L. (1984) Sparsity–preserving SOR Algorithms for Separable Quadratic and Linear Programs, Computers and Operations Research 11,105-112.

Mangasarian, O. L. and De Leone, R. (1986a) Parallel Successive Overrelaxation Methods for Symmetric Linear Complementarity Problems and Linear Programs, Computer Sciences Technical Report #647, University of Wisconsin–Madison.

Mangasarian, O. L. and De Leone, R. (1986b) Parallel Gradient Projection Successive Overrelaxation for Symmetric Linear Complementarity Problems and Linear Programs, Computer Sciences Technical Report #659, University of Wisconsin–Madison.

Mangasarian, O. L. and Shiau, T. H. (1986) Lipschitz Continuity of Solutions of Linear Inequalities, Programs and Complementarity Problems, SIAM Journal on Control and Optimization 24, 1986.

Medhi, D. (1987) Decomposition of Structured Large–Scale Optimization Problems and Parallel Optimization, Computer Sciences Department, University of Wisconsin–Madison.

Missirlis, N. M. (1985) A Parallel Iterative System Solver, Linear Algebra and its Applications 65,25-44.

Murty, K. C. (1983) "Linear Programming", Wiley, New York.

Pang, J. S. and Yang, J. M. (1987) Two–Stage Parallel Iterative Methods for the Symmetric Linear Complementarity Problem, School of Management, The University of Texas at Dallas.

Phillips, A. T. and Rosen, J. B. (1986) Multitasking Mathematical Programming Algorithms, Computer Science Department Technical Report #86-10, University of Minnesota.

Rockafellar, R. T. (1970) "Convex Analysis", Princeton, University Press, Princeton, New Jersey.

Rockafellar, R. T. (1976a) Monotone Operators and the Proximal Point Algorithm, SIAM J. Control Opt., Vol 14, No. 5.

Rockafellar, R. T. (1976b) Augmented Lagrangians and Applications of the Proximal Point Algorithm in Convex Programming, Mathematics of Operations Research, Vol 1, No. 2, 97-116.

Schnabel, R. B. (1984) Parallel Computing in Optimization, Department of Computer Science Technical Report #CU-CS-282-84, University of Colorado.

Shiau, T. H. (1983) Iterative Linear Programming for Linear Complementarity and Related Problems, Computer Sciences Techincal Report # 507 , University of Wisconsin–Madison.

Subramanian, P. K. (1985) Iterative Methods of Solution for Complementarity Problems, Computer Sciences Technical Report #607, University of Wisconsin–Madison.