Efficiency and Performance of Some Algorithms in Mathematical Programming

by

Navneet Singh

Submitted to the Department of E.E.C.S.
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering & Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

Author .....
Department of E.E.C.S.
February 1, 1996

Certified by .....
Robert M. Freund
Professor
Thesis Supervisor

Accepted by .....
F.R. Morgenthaler
Chairman, Dept. Committee on Graduate Theses

Efficiency and Performance of Some Algorithms in Mathematical Programming

by

Navneet Singh

Submitted to the Department of E.E.C.S.
on February 1, 1996, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering & Computer Science

Abstract

In this thesis, we analyze three optimization algorithms designed to solve mathematical programming problems. We prove useful results about them, and attempt to implement some of the more promising ones. The optimization problems we consider are all members of P, the class of problems solvable in polynomial time. They are reducible to each other and the algorithms we study can be adapted to solve any of the problems. The motivation for this thesis is the pursuit of algorithms for optimization problems that are efficient in theory (in P) as well as efficient in practice. However, to come up with such algorithms is not the goal of this thesis. We hope that by studying the theoretical properties of, as well as examining the practical perfomance of these optimization algorithms, we can get a step closer to devising an algorithm that is efficient, both in theory and in practice.

Thesis Supervisor: Robert M. Freund
Title: Professor

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to my thesis advisor, Prof. Robert M. Freund, for the time, assistance and advice, which he has generously given to me. Coming from a Computer Science background, I had little experience in Operations Research when I first started working with Prof. Freund. He not only gave a non-O.R. undergraduate student an opportunity to do research with him, but he also guided me through every step of my project. Without his continuous supervision and guidance, my progress on the project would not have been possible. His patience and teaching style helped me understand the challenging concepts that I encountered much better. Moreover, I learned a great deal about the field of Operations Research from him, and truly gained an appreciation for it. His enthusiasm for the field is especially contageous.

I am also indebted to my parents for all the encouragement and support that they have given me. They have always been there for me, and if it were not for them, I certainly would not have reached where I have. Finally, I'd like to thank my brother, Raman, for being a wonderful pal.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many optimization problems are equivalent to each other. This is so because they are members of a class of problems called P, the class of polynomial-time-solvable problems. It has been proven that all problems in this class are (polynomially) reducible to each other. Consequently, a known algorithm for a certain kind of problem in P can be adapted to solve a different kind of problem in P. In this thesis, we explore algorithms for different types of problems in P, learn from them, prove useful properties that they might have, and implement some of the more promising ones. We have the assurance, however, that progress toward any one of the algorithms is progress toward the more efficient solution of all the various optimization problems we study.

Once it has been shown that a certain kind of problem is in P, there is no guarantee that there is a practically efficient algorithm that solves the problem. This might seem to contradict what I mentioned above, that an algorithm which solves one type of problem in P can solve any other. Although this is true, algorithms might have an efficient implementation for one kind of problem, and yet, none for another kind of problem. Moreover, efficiency is measured both from a theoretical and a practical perspective. Theory asks the question "What is the worst-case performance of the

algorithm?", while practice asks the question "How does the algorithm perform in frequently encountered situations?"

We explore algorithms for different types of problems, learn from them and point out their strengths and weaknesses. We will also be involved in the software implementation of some of the more promising algorithms. Specifically, we concentrate on three mathematical programming algorithms:
1) Von Neumann's Algorithm for linear feasibility with a convexity constraint
2) the Relaxation Method for linear inequalities, and
3) the Ellipsoid Algorithm.

We first study these algorithms and their various properties, and prove some useful results about them. We then focus on Khachiyan's Ellipsoid Method and implement it for two different problems, system of linear inequalities and the semi-definite programming problem. The program will take as input a system of linear inequalities or a semi-definite program and will show the iterations of the Ellipsoid Algorithm and return a solution if one exists, i.e. if the system is feasible. The motivation for doing all of the above is the pursuit of algorithms for optimization problems that are efficient in theory (in P) as well as efficient in practice. However, to come up with such algorithms is not the goal of this thesis. We hope that by proving relevant properties of, and implementing some promising optimization algorithms to see how they run in practice, we can get a step closer to devising an algorithm that is more efficient, both in theory and in practice.

Chapter 2

Background

2.1 Types of Problems

Although we mentioned in Chapter 1 that all problems in P are equivalent to each other, we state here the different types of problems that the various algorithms solve because these are the "modes" that they work in.

2.1.1 System of Linear Inequalities

Find an n-vector x satisfying

$$\sum_{j=1}^{n} a_{ij}x_j + b_i \geq 0, \quad i = 1, ..., m \tag{2.1}$$

2.1.2 Linear Feasibility with a Convexity Constraint

Find $x = (x_1, x_2, ..., x_n) \geq 0$, such that

$$\sum_{j=1}^{n} P_j x_j = 0, \sum_{j=1}^{n} x_j = 1 \tag{2.2}$$

where $P_j \in R^m$ and $\| P_j \| = 1 \; \forall j$

2.1.3 Semi-Definite Programming

2) Find an n-vector x satisfying

$$\sum_{i=1}^{n} x_i A_i \succ 0 \tag{2.3}$$

where A_i are symmetric matrices.

In other words, find x such that the above summation yields a positive definite matrix.

2.2 Motivation

Given an m x n matrix A, we know that one and only one of the following two systems has a solution:

$$Ax \geq e \tag{2.4}$$

or

$$A^T y = 0,\ y \geq 0,\ e^T y = 1 \tag{2.5}$$

Consider the problem of devising an efficient algorithm that will attempt to solve system (2.4) if it is feasible.

If (2.4) were feasible, the Relaxation Method could be used to find the solution. On the other hand, if (2.5) were feasible, the Von Neumann Algorithm would be applicable. However, the problem is that we do not have an a priori idea of which system is feasible.

Since the two algorithms work on dual problems, it would be nice to know the relationship between the variables local to them. This would allow us to start with the Von Neumann Algorithm on a particular system, solve it if the system is feasible, or detect infeasibility and return the current state of the variables. If the original

system is infeasible, the Relaxation Method could take over on the dual system from the point where the Von Neumann Algorithm left off on the original system.

Chapter 3

The Relaxation Algorithm

Motzkin and Schoenberg's Relaxation Algorithm solves the problem:
Find an n-vector x satisfying

$$\sum_{j=1}^{n} a_{ij}x_j + b_i \geq 0 \tag{3.1}$$

Their claim is that each of the inequalities (3.1) defines a closed half-space:

$$H_i = \{x \in \Re^n | \sum_j a_{ij}x_j + b_i \geq 0\}, \quad i = 1, ..., m \tag{3.2}$$

Using (3.2), the set of all solutions to (3.1), i.e., the feasible set is, of the form $A = \bigcap_{i=1}^{m} H_i$.

At this point, we must introduce a concept, that of pointwise distance. A point p_{j+1} is pointwise-closer to set A than another point p_j if $\mid p_j - a \mid > \mid p_{j+1} - a \mid \forall a \in A$. The terms pointwise-equidistant and pointwise-farther are defined analogously. The Relaxation Algorithm furnishes a sequence of points p_i, such that for $j \geq 1$, p_{j+1} is pointwise closer to A[10].

The Relaxation Algorithm works as follows. If p_k is the current point, the algorithm chooses a violated constraint (i.e. a hyperplane π_i that p_k is on the wrong side of), defines q_k to be the projection of p_k on the hyperplane π_i, and computes p_{k+1} as $p_{k+1} = p_k + \lambda(q_k - p_k)$ where λ is a fixed number between 0 and 2.
This method can be summarized as
$p_{k+1} = p_k + \frac{\lambda A_i(b_i - A_i p_k)}{A_i{}^T A_i}$.
Depending on the choice of λ, we can get some interesting results about the termination of the Relaxation Method. These results are summarized in the following table:

	$0 < \lambda < 1$	$\lambda = 1$	$1 < \lambda < 2$	$\lambda = 2$
Does the sequence converge?	YES Proof 1	YES Proof 2	YES Proof 2	YES Proof 2
Does it terminate?	NO CounterEx 1	NO CounterEx 2	YES Proof 3	YES Proof 4
Can we choose any violated constraint?	NO CounterEx 3	YES Proof 2	YES Proof 2	YES Proof 2

Table 3.1: Summary of the properties of Relaxation Method

3.1 Proofs

3.1.1 Proof 1

The first thing to notice is that the Relaxation Method never produces duplicate points. Also, since the polytope A has non-zero volume, the locus of all points pointwise-equidistant from A is simply a point. We infer that the sequence of points p_i is getting STRICTLY pointwise-closer to A. It follows from this that p_i must converge to SOME point. Say the sequence converges to a point $l \notin A$ and the

distance from l to the farthest violated hyperplane, π, is x. Since the sequence of points converges to l, we know that, after a while, all points will violate π.

We are also guaranteed, because of convergence, that there exists a v, such that $\mid p_{i+1} - p_i \mid < \epsilon$, for all $i \geq v$, and for any choice of ϵ.
But if we choose ϵ less than $\frac{\lambda x}{1+\lambda}$, this will not hold true. Consider a point p_i, $(i > v)$ that violates π. Using the max criterion, it will be moved by at least $\frac{\lambda x}{1+\lambda}$.

Therefore $l \in A$.

$\otimes$

3.1.2 Proof 2

It was established in Proof 1 that the sequence of points produced by the Relaxation Method converges to *some* point, l (inside or outside A). We suppose that $l \notin A$ and reach a contradiction.

Case 1: *$l \notin A$ and l does not lie on any hyperplane*

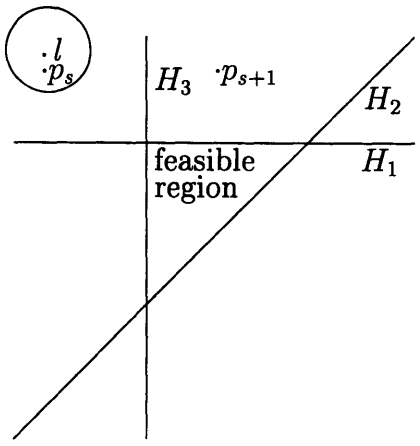

Figure 3.1: Sequence of points must converge to a point in feasible region

If $l \notin A$, then it must violate at least one constraint, i.e. be on the wrong side of a hyperplane. Let d be the distance from l to a closest hyperplane (not necessarily a violated one). We can guarantee, since the sequence of points converges to l, that

after a while all points in the sequence will lie in the sphere $S = \{x| \mid x - l \mid = d/2\}$. Let p_s be such a point inside S. We note that p_s and l violate the same constraints. Now, p_{s+1} will be the projection or reflection of p_s on some hyperplane (since λ is between 1 and 2). In either case, p_{s+1} is outside S and we have a Contradiction.

Case 2: *$l \notin A$ and l lies on exactly one hyperplane*
You simply let d be the distance to the next closest hyperplane and use an approach similar to that of Case 1.

Case 3: *$l \notin A$ and l lies on the intersection of multiple hyperplanes*
We again let d be the distance to the closest hyperplane that l does not lie on, and claim that after a while all points in the sequence will lie in the sphere $S = \{x| \mid x - l \mid = d/2\}$. However, we now need to show that some point in this sphere will violate a constraint other than the ones that l lies on. Or equivalently, if we temporarily limit our constraints to the hyperplanes that l lies on, we need to show that the sequence of points terminates.

We use the property that S is compact, and the theorem -
HEINE-BOREL THEOREM: T is compact $\Longleftrightarrow$ Each open cover of T contains a finite subcover.
We first give a brief background on open sets[11]. Define a topology on a set X to be a collection of its subsets, τ, having the following properties:
1) $\emptyset$ and X are in τ.
2) The union of the elements of any subcollection of τ is in τ.
3) The intersection of the elements of any finite subcollection of τ is in τ.
We say that a subset U of X is an **open set** of X if U belongs to the collection τ.

A direct implication of the Heine-Borel theorem is that in order to show there exists a finite cover, it suffices to show that there exists an infinite open cover, which

is what we do below.

We provide a cover C for S in the following manner.

Let $C_i = \{x \in S \mid x \text{ falls in A after i iterations}\}$. If C is the collection $C_1, C_2, ...,$ then it is straightforward to show that C is a topology on S.

Although C is an infinite cover, the Heine-Borel theorem guarantees that there exists a finite subcover for S since S is compact.

Therefore, p_i terminates in a finite number of steps.

$\otimes$

3.1.3 Proof 3

Let p^i be the sequence of points generated by the Relaxation Method. We know that, since p^i converges to a point l on the boundary of the feasible region A, there exists a v, such that $p^i (i \geq v)$ violate only those constraints that l lies on. Let S be the sphere $S = \{x| \mid x - l \mid = \mid p^v - l \mid\}$. We know that there exists a $w \geq v$ such that $p^i \in S (i \geq w)$. We have now reduced this problem to the termination problem stated in Case 3 of Proof 2. Therefore, p^i terminates in a finite number of steps.

$\otimes$

3.1.4 Proof 4

We are at least guaranteed convergence on the boundary of A (see Proof 2). We suppose that the sequence does not terminate and reach a contradiction. We note that, because the sequence of points converges to a point l on the boundary of A, after a while all points in the sequence will violate only those constraints that l lies on. Let p_{v_0-1} be the last point that violates some other constraint. Let S be the sphere $S = \{x| \mid x - l \mid = \mid p_{v_0} - l \mid\}$. We note that every point $p_v (v > v_0)$ lies on S because it is the reflection of the previous point in a hyperplane through the center of S, namely l. Therefore, p_v can never converge to l.

$\otimes$

3.2 Counterexamples

3.2.1 Counterexample 1

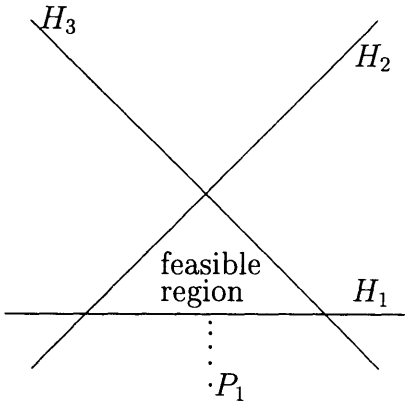

Figure 3.2: Relaxation Method does not terminate for $0 < \lambda < 1$

Start off with P_1. It only violates H_1, and so does every point thereafter.

⊗

3.2.2 Counterexample 2

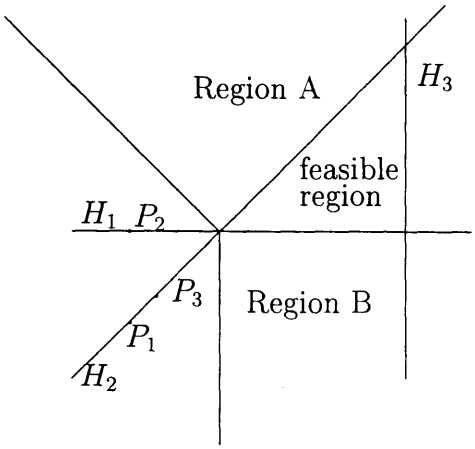

Figure 3.3: Relaxation Method does not terminate for $\lambda = 1$

Since $\lambda = 1$ corresponds to the projection method, the only way a sequence will terminate is if some point is projected on to a hyperplane in the feasible region.
In the above example, there are only 3 hyperplanes. We see that H_3 is never violated. So our only hope is that a point be projected on to H_1 or H_2 in the feasible region. We note that in order for a point to be projected on the feasible side of H_2, it must be in Region B (and similarly for H_1 in Region C).
Therefore, P_i in the figure above serves as a counterexample.

⊗

3.2.3 Counterexample 3

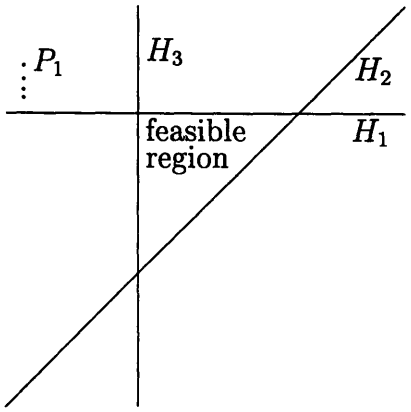

Figure 3.4: Relaxation Method needs max criterion for $0 < \lambda < 1$

Point P_1 violates H_1 and H_3. Using the max criterion, you'd move it closer to H_3. However, if you do not use the max criterion and keep moving it closer to H_1, the sequence of points does not converge to a feasible solution.

⊗

Chapter 4

Von Neumann's Algorithm

4.1 The Problem

Von Neumann's Algorithm tries to solve the homogeneous form of the Linear Feasibility Problem with a convexity constraint, namely, find $x = (x_1, x_2, ..., x_n) \geq 0$, such that

$$\sum_j P_j x_j = 0, \sum_j x_j = 1 \tag{4.1}$$

where $P_j \in R^m$ and $\| P_j \| = 1 \; \forall j$.

The geometric interpretation of this problem is that $P_j \in R^m$ are points lying on a hypersphere S with radius equal to 1 and the center at the origin[6]. The problem then becomes that of assigning non-negative weights x_j to the points P_j so that their weighted vector sum is the origin.

4.2 The Algorithm

The algorithm is initiated with any approximation to the origin:
$u_1 = A_1 = \sum P_j x_{1_j}$ where

$$x_{1_1} = 1, x_{1_j} = 0 \; for \; j \neq 1. \tag{4.2}$$

Then, the algorithm improves its initial guess iteratively with the following steps until it finds a solution that is good enough, i.e., until $\| A_t \| = \epsilon$, where ϵ is the desired precision.

On iteration t,

- Among directions from the origin, find the point P_j which makes the largest angle with the residual vector A_{t-1}.
- Stop if $v = (A_{t-1})^T P_j$ is positive for all j.
- A_t is the closest point to the origin on the line segment joining A_{t-1} to P_j. Update

$$A_t = \lambda A_{t-1} + (1 - \lambda) P_j$$
$$u_t{}^2 = \lambda v + (1 - \lambda)$$
$$x_{t_i} = \lambda x_{t-1_i} \text{ for } i \neq j \text{ and } x_{t_j} = \lambda x_{t-1_j} + (1 - \lambda)$$

where $\lambda = (1 - v)/(u_{t-1}{}^2 - 2v + 1)$

The above can be summarized as the update formula:

$$x_{t+1} = x_t(1 - \frac{\|x_t\|^2 - A_i x_t}{\|A_i - x_t\|^2}) + (\frac{\|x_t\|^2 - A_i x_t}{\|A_i - x_t\|^2}) e_i.$$

Chapter 5

The Ellipsoid Method

The Ellipsoid Method is the first known polynomial-time algorithm for linear programming[4]. However, it has been criticized for being inefficient in practice. Our goal is to explore this conjecture by implementing the Ellipsoid Method for the following two problems, and running some average-size test cases:

1) Find an n-vector x satisfying

$$A^T x \leq b, \tag{5.1}$$

where A^T is $m \times n$ and b is an m-vector.

2) Find an n-vector x satisfying

$$\sum_i x_i A_i \succ 0 \tag{5.2}$$

where A_i are symmetric matrices.
In other words, find x such that the above summation yields a positive definite matrix.

The basic idea of the Ellipsoid Method is to generate a sequence of ellipsoids until the center of the current ellipsoid is a solution. In this way, it is similar to the other algorithms we have come across since it starts out with a guess, the center of the first Ellipsoid, and keeps modifying it as necessary.

5.1 System of Inequalities

For problem (6.1), the method generates a sequence $E_0, E_1, ..., E_k, ...$ in the following manner:

1) Let E_k be represented as $E_k = \{x|(x - x_k)^T J^{-1}(x - x_k) \leq \frac{1}{\delta}\}$. Start with $E_0 = \{x|(x - 0)^T I(x - 0) \leq \frac{1}{\delta}\}$ and $\delta = 2$.

2)Check to see if the center x_k, of the current ellipsoid E_k is a solution. If not, some constraint violated by x_k, say $\alpha^T x \leq \beta$ is chosen and the ellipsoid of minimum volume that contains the half-ellipsoid $\{x \in E_k|\alpha^T x \leq \alpha^T x_k\}$ is constructed. This involves updating x_k and J.

3)Run until you find a feasible point or you hit the stopping criterion determined as follows:

4)Factor $J = M^T DM$ and stop if $D_{ii} > \delta$ for some i.

5)If you stop, reset $\delta = \frac{\delta}{2}$ and restart.

In order to implement the above method in Matlab, I created the following five modules:

•**emethod:** This is the main module that gets the system to be solved from the user, does the necessary initialization, and calls the required modules.

•**iterate:** This module is responsible for performing the iterations in an organized fashion, and for printing the results of each iteration.

•**check:** This module checks to see if the proposed solution, or the center of the current ellipsoid is a feasible solution. If not, it checks to see which constraint the proposed soltion violates and returns that constraint.

•**update:** If the proposed solution does not work, a new ellipsoid must be created. The update module takes care of that by updating the current center, J matrix and the other related parameters.

•**pump:** This module checks to see if the stopping criterion has been reached. If so, it updates the δ and instructs the iterate module to start from afresh with the new δ.

5.2 Semi-Definite Programs

Here we are concerned with problem (5.2), the semi-definite programming problem.

The implementation of this problem is similar in a number of ways to that of the System of Linear Inequalities. However, finding the hyperplane that separates the proposed solution from the feasible set is a great deal more challenging in this case. In what follows we provide a way of computing such a hyperplane.

Consider the feasible set $X = \{x \mid \sum x_i A_i \succ 0\}$, where A_i are symmetric matrices. Define $A \bullet B = tr(AB)$, where $tr(M) = \sum_{i=1}^{n} M_{ii}$[8].

Let the proposed solution be $\bar{x}$. We let $Q = \sum \bar{x}_i A_i$, and factorize $Q = M^T D M$, where $M^T M = I$.

Let $d =$ diagonal of D. If $d > 0$, then $Q \succ 0$ and we are done. If not, there exists a

j such that $d_j \leq 0$. Let E be the matrix with 1 in the j^{th} diagonal element and the rest of the matrix elements 0, and let $P = M^T EM$. Also let $y_i = A_i \bullet P, \ \ i = 1, ..., m$.

Proposition $\bar{x}^T y \leq 0$.

Proof $\bar{x}^T y = \sum \bar{x}_i y_i = \sum \bar{x}_i A_i \cdot P = Q \cdot P = D \cdot E = d_j \leq 0 \quad \otimes$.

Proposition $x^T y > 0 \ \forall x \in X$.

Proof If $x \in X$, $\sum x_i A_i \succ 0$. $x^T y = \sum x_i A_i \cdot P = (\sum x_i A_i) \cdot P \succ 0$ since $\sum x_i A_i \succ 0$ and $P \succeq 0$, and $P \neq 0$. $\otimes$

Therefore, the hyperplane $\{x \mid y^T x = 0\}$ separates $\bar{x}$ from X.

Once we have the separating hyperplane, the implementation is almost identical to that of the System of Linear Inequalities. We simply have one additional module, namely, separate, in addition to the emethod, iterate, check, pump and update modules. The actual matlab code for both problems is included in Appendix A.

Chapter 6

Results and Further Extensions

Our purpose behind studying the Relaxation Method and the Von Neumann Algorithm was the pursuit of devising a single, efficient algorithm that would solve the feasible system among the following pair:

$$Ax \geq e \tag{6.1}$$

and

$$A^T y = 0,\ y \geq 0,\ e^T y = 1 \tag{6.2}$$

We knew that such an algorithm would need a criterion to determine which system was feasible and "switch modes" appropriately. The Von Neumann Algorithm provided us with a way of determining infeasibility. Therefore, it sounded like a good idea to start the algorithm mimicking the Von Neumann Algorithm, stopping with a feasible solution if the latter did, or continuing with mimicking a different algorithm suitable for the dual system if the Von Neumann Algorithm declared the original system infeasible. This is where the Relaxation Method came into action. We proved that it was guaranteed to solve the dual system and stated the condi-

tions under which it terminated in a finite number of iterations, among other things. However, it was not clear where the Relaxation Method would start. The idea was to not re-invent the wheel and, if the Von Neumann Algorithm declared infeasibility, convert the current state of the variable to their corresponding dual variable state.

One of the major extensions of this thesis can be in the area discussed above. Specifically, if the duality relationship between the variables of the Von Neumann Algorithm and the Relaxation Method is articulated, then the goal of devising a combined algorithm can be achieved. Although this combined algorithm would be a very general one, it would not be an efficient one in theory since neither the Von Neumann Algorithm nor the Relaxation Method are polynomial-time. For this reason, we researched the Ellipsoid Method.

The Ellipsoid Method was the first known algorithm for mathematical programming that was in P, i.e. polynomial-time in its worst case. However, this was no guarantee that it would be able to solve the average everyday problems efficiently. Therefore, we tested the Ellipsoid Method to see how it would work in practice. We implemented the algorithm for two types of problems - set of linear nequalities, and semi-definite programming - and then ran it on a set of medium-size problems. On an absolute scale (since we did not have implementations of other algorithms to test it against), the Ellipsoid method behaved well. It certainly solved the problems in a number of iterations that was polynomial in the size of the problems. Another possible extension of this thesis is to determine exactly how the Ellipsoid Method performs relative to the commonly used optimization algorithms.

Appendix A

E-Method Code

A.1 System of Inequalities

A.1.1 emethod1

```
1  function [x] = emethod1

2  % EMETHOD1 Solve Ax <= b using the Ellipsoid Method.
3  %          EMETHOD1 prompts the user for an m x n matrix A and an m-vector b.
4  %          It returns an n-vector x if a solution to Ax <= b exists.
5  %          Otherwise, it declares the system infeasible.
6  %
7  %          See also EMETHOD2

8  global j x a b t s d m n counter del;

9  %%% Prompt the user for the input
10 a = input('Please enter a matrix in the form [row1; row2; ...]');
11 b = input('Now enter the right hand side vector in the form [components]');
```

```
12  m = size(a,1);                %%%   m is the number of rows of a
13  n = size(a,2);                %%%   n is the number of columns
14  del = 1/2;
15  j = (1/del)*eye(n);           %%%   initialize ellipsoid
16  x = zeros(n,1);               %%%   initialize center of ellipsoid
17  t = 1/(n + 1);
18  s = 2/(n + 1);
19  d = (n^2)/(n^2 - 1);

20  iterate1;
```

A.1.2 iterate1

```
1   function [] = iterate1

2   % ITERATE1 Perform the iterations of the Ellipsoid method for the problem
3   %          Ax <= b. Continue performing iterations until the proposed
4   %          solution, x, works

5   global j x a b t s d m n counter del;

6   proposed = '[';
7   i = 1;

8   while i < n+1
9   temp = num2str(x(i));
10  if size(temp,2) < 4 proposed = [proposed, temp];
11  else proposed = [proposed, temp(1,1:3)];
```

```
12 end
13 if i < n proposed = [proposed, ' '];
14 end
15 i = i+1;
16 end

17 proposed=[proposed,']'];
18 fprintf('Iteration %g --- Soln: %s\n',counter,proposed);

19 y = check1;
20 if y == 0 return
21 else update1(y);
22 end

23 pump1;
24 counter = counter + 1;
25 iterate1;
```

A.1.3 check1

```
1 function [y] = check1

2 % CHECK Look for a violated constraint of the form a(i)x > b(i). If one exists,
3 %       return i. If none exists, then x is a solution to the system ax <= b.
4 %       In the latter case, return 0.
5 %
6 %       Works in conjuction with optimization functions for the problem
7 %       ax <= b which have declared x a b and m as global variables
```

```
8  global j x a b t s d m n counter del;


9  i=1;
10 while i < m+1
11 if (dot(a(i,:),x)) ~= (b(i) + .0001)  %%% does x violate a(i)x <= b(i)
12      y = i;
13      return;
14      end
15 i = i+1;
16 end
17 y = 0;
```

A.1.4 pump1

```
1  function [] = pump1


2  % PUMP1 Check to see if the given stopping criterion has been reached.
3  % If so, "pump" the volume of the ellipsoid and restart.


4  global j x a b t s d m n counter del;


5  [M,D] = eig(j);
6  dia = diag(D);
7  minim = min(dia);


8  if ~(minim > del)
9       del = del*2;
10      j = (1/del)*eye(n);
11      x = zeros(n,1);
```

```
12 end
```

A.1.5 update1

```
1  function [] = update1 (i)


2  % UPDATE1 Adjust the proposed solution and the corresponding ellipsoid after
3  %         an unsuccessful iteration of the Ellipsoid method for the problem
4  %         Ax <= b

5  global j x a b t s d m n counter del;

6  v = a(i,:);
7  y = (j'*v')/dot((j'*v'),(j'*v'));
8  w = j*y;

9  x = x - (t*w);
10 j = sqrt(d)*j*(eye(size(j))+(y'*y));
```

A.2 Semi-Definite Programming

A.2.1 emethod2

```
1  function [x] = emethod2


2  % EMETHOD2 Solve the problem Sigma(x(i)A(i)) is p.d. using the Ellipsoid
3  %          Method. EMETHOD2 prompts the user for a collection of i
4  %          symmetric matrices of size n. It returns an i-vector x if a
```

```
5  %          solution to the above system exists. Otherwise, it declares
6  %          the system infeasible.
7  %
8  %          See also EMETHOD


9  global i n a j x v del

10 a = []          %%% a will consist of all input matrices side-by-side

11 %%% Prompt the user for input

12 i = input('How many symmetric matrices are there in the problem?')
13 temp = input('Enter the next matrix and hit return')

14 n = size(temp,2)
15 a = [a temp]

16 del = 2

17 for jk = 2:i
18 temp = input('Enter the next matrix and hit return')
19 a = [a temp]
20 end

21 j = (1/del)*(eye(i))          %%%    initialize ellipsoid

22 x = zeros(i,1)               %%%    initialize center of ellipsoid
```

```
23 pump2

24 iterate2                    %%%    now ready for first iteration of E-method
```

A.2.2 iterate2

```
 1 function [] = iterate2

 2 % ITERATE Perform the iterations of the Ellipsoid method for the problem
 3 %         Sigma(x(i)A(i)) is p.d. Continue performing iterations until
 4 %         the proposed solution, x, works


 5 global i n a j x v

 6 M = check2                   %%%    Check to see if the current solution works

 7 if v == 0 return             %%%    If it works, return to caller function
 8 else separate2(M)
 9 update2                      %%%    If not, update the parameters, and
10 end


11 iterate2                    %%%    Perform iteration with the updated parameters
```

A.2.3 check2

```
 1 function [M] = check2
```

```
2  % CHECK Look for a violated constraint of the form d(i) <= 0 (i.e. an
3  %       eigenvalue of the sum matrix is non-positive). If one exists,
4  %       return i. If none exist, then the current x is a solution, in
5  %       which case, return 0.


6  global i n a j x v ys

7  Q = zeros(n)

8  for jk = 1:i                                    %
9  temp = a(:,[((jk-1)*n + 1):jk*n])               % Q is the sum of the
10 Q = Q + x(jk)*temp                              % individual A(i)
11 end                                             %

12 [M,D] = eig(Q)
13 d = diag(D)

14 for k = 1:n                                     %
15 if d(k) <= 0                                    %
16    v = k                                        % Check for an eigenvalue <= 0
17    return                                       %
18 end                                             %
19 end                                             %

20 v = 0
```

A.2.4 pump2

```
1  function [] = pump2

2  % PUMP2 Check to see if the given stopping criterion has been reached.
3  %       If so, "pump" the volume of the ellipsoid and restart.

4  global i n a j x v del



5  [M,D] = eig(j)
6  dia = diag(D)

7  ind = 1

8  while ind < i+1
9  if dia(ind) > del
10 del = del/2
11 j = (1/del)*eye(i)
12 pump2
13 end
14 ind = ind + 1
15 end
```

A.2.5 separate2

```
1  function [] = separate2(M)

2  % SEPARATE Find the hyperplane that separates the current proposed solution,
3  %          x, from the feasible region
```

```
4  global i n a j x v q


5  E = zeros(n)
6  E(v,v) = 1


7  P = M'*E*M


8  for z = 1:i
9  temp = a(:,[((z-1)*n + 1):z*n])
10 q(z) = trace(temp*P)
11 end


12        %%%%%%%%% q'x = 0 is the desired separating hyperplane. %%%%%%%%%
```

A.2.6 update2

```
1  function [] = update2


2  % UPDATE Adjust the proposed solution and the corresponding ellipsoid after
3  %        an unsuccessful iteration of the Ellipsoid method for the problem
4  %        Sigma(x(i)A(i)) is p.d.
```

```
5  global i n a j x v q


6  B = j*j'

7  al = (q*x)/sqrt(q*B*q')
8  t = (1+i*al)/(i+1)
9  s = 2*t/(1+al)
10 d = (i^2)*(1-(al^2))/((i^2)-1)

11 y = (j'*q')/dot((j'*q'),(j'*q'))
12 w = j*y

13 x = x + (t*w)                              %%% adjust the current solution, and

14 j = sqrt(d)*j*(eye(size(j))-pi*dot(y,y))    %%% the corresponding ellipsoid
```

Bibliography

[1] Shmuel Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:382–392, 1954.

[2] Farid Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[3] Mokhtar Bazaraa. *Non-linear Programming.* Wiley, New York, 1993.

[4] Robert G. Bland, Donald Goldfarb, and Michael J. Todd. The ellipsoid method: A survey. *Operations Research*, 29(6):1039–1091, Nov-Dec 1981.

[5] Bruce P. Burrell and Michael J. Todd. The ellipsoid method generates dual variables. *Mathematics of Operations Research*, 10(4):688–700, November 1985.

[6] George B. Dantzig. An ϵ-precise feasible solution to a linear program with a convexity constraint in $\frac{1}{\epsilon^2}$ iterations independent of problem size. Technical report, Stanford University, 1991.

[7] Curtis Eaves. Piecewise linear retractions by reflexion. *Linear Algebra and its Applications*, 7:93–98, 1973.

[8] Robert M. Freund. Complexity of an algorithm for finding an approximate solution of a semi-definite program with no regularity assumption. M.I.T. Opera-

tions Research Center Working Paper #OR314-95, submitted to *Mathematical Programming*, October 1995.

[9] J.L. Goffin. The relaxation method for solving systems of linear inequalities. *Mathematics of Operations Research*, 5(3):388–414, August 1980.

[10] T.S. Motzkin and I.J. Schoenberg. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:393–404, 1954.

[11] James R. Munkres. *Topology: a first course.* Prentice-Hall, 1975.

[12] James Renegar. Some perturbation theory for linear programming. *Mathematical Programming*, 65:73–91, 1994.

[13] R. Tyrrell Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, N.J., 1970.