SOLVING LARGE SCALE COMBINATORIAL OPTIMIZATION PROBLEMS

by

Steven John Benson

An Abstract

Of a thesis submitted in partial fulfillment of the requirements
for the Doctor of Philosophy degree in Applied
Mathematical and Computational Sciences
in the Graduate College of The
University of Iowa

July 1998

Thesis supervisor: Professor Yinyu Ye

ABSTRACT

I use the positive semidefinite relaxation to approximate combinatorial and quadratic optimization problems subject to linear and boolean constraints. I present a dual potential reduction algorithm and show how it exploits structure and sparsity. Coupled with randomized and heuristic methods, I report computational results for approximating combinatorial problems with dimension up to 7000.

Abstract approved: ______________________
Thesis supervisor

____
Title and department

____
Date

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Steven John Benson

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Applied Mathematical and Computational Sciences at the July 1998 graduation.

Thesis committee: ______________________
Thesis supervisor

____
Member

____
Member

____
Member

____
Member

SOLVING LARGE SCALE COMBINATORIAL OPTIMIZATION PROBLEMS

by

Steven John Benson

A thesis submitted in partial fulfillment of the requirements
for the Doctor of Philosophy degree in Applied
Mathematical and Computational Sciences
in the Graduate College of The
University of Iowa

July 1998

Thesis supervisor: Professor Yinyu Ye

To my parents

ACKNOWLEDGEMENTS

I am especially indebted to Professor Yinyu Ye. As an advisor, he pointed me toward exciting areas of research, offered challenging opportunities, and set an example of professionalism. As an instructor, he taught classes that were always clear, focused, and very worthwhile. I feel very fortunate to have had the opportunity to work with him.

I am also grateful to the excellent faculty members at the University of Iowa. This list includes Professors Atkinson, Han, Hethcote, and Ye. Their talents and willingness to teach others make the University of Iowa an excellent institution for graduate studies in pure and applied mathematics.

Finally, I want to acknowledge the friends I had in Iowa City. Wally LoFaro, Eric Rawdon, Joe Stickles, and many others made the work more fun and my stay in Iowa a positive one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
POSITIVE SEMIDEFINITE RELAXATION OF COMBINATORIAL OPTIMIZATION PROBLEMS

1.1 Combinatorial Optimization

Combinatorial optimization problems ask for the combination of variables that optimize an objective criteria. These problems are generally nonconvex and are examples of a very difficult class of problems labeled NP hard. The only known methods for finding the optimal answer are direct searches. Because the number of possible solutions grows exponentially with the number of variables, problems with only a few dozen variables offer so many possibilities that even the fastest computers cannot find the best answer in a reasonable amount of time. Branch and bound algorithms can pare the search, but usually not enough. Therefore considerable attention has been paid to approximating the solution with an answer of very high quality.

1.1.1 Graph Partitioning Problems

Several combinatorial problems ask for a partition of the vertices of a graph into two sets. Each partition defines a cut which is the sum of the weighted edges that connect vertices in one set with vertices in another set. For example, placing vertices A,B, and C of Figure 1 together in one set and vertices D,E,F,G, and H together in the other are put in the other set gives a cut of 9: the two edges that connect the two sets have weights 4 and 5. The maximum cut problem looks for the partition with the largest cut.

Assigning each vertex a $+1$ or -1 will partition a graph into two sets. Letting

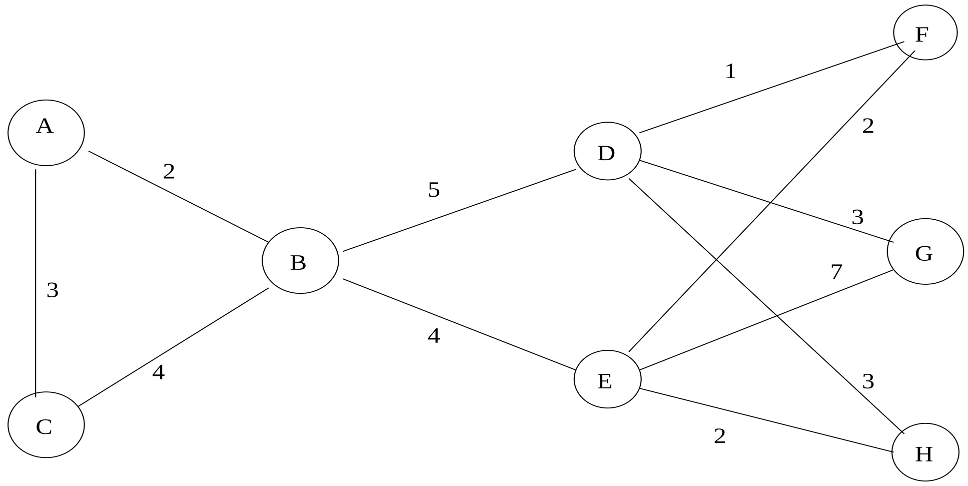

Figure 1: Undirected graph $\tilde{C}$.

$v \in \Re^n$ represents the n vertices of the graph with adjacency matrix $\tilde{C}$, the maximum cut problem can be formulated as:

$$\begin{aligned} &\text{Maximize} \quad \frac{1}{2}\sum_{i<j} \tilde{C}_{i,j}\,(1 - v_i v_j) \\ &\text{Subject to} \quad v \in \{-1,1\}^n, \end{aligned} \tag{1.1}$$

Additional constraints may also be present. For example, the problem may require two particular vertices, s and t must be in different sets. Impose this constraint by letting $e_{st}^T v = 0$ where e_{st} is a vector of zeros except at indices s and t, which equal 1. Alternatively, the partition may require the vertices be split evenly into two sets. That constraint can be written as $e^T v = 0$ where e is the vector with each element equal to one. It can be formulated as:

$$\begin{aligned} &\text{Maximize} \quad \frac{1}{2}\sum_{i<j} \tilde{C}_{i,j}\,(1 - v_i v_j) \\ &\text{Subject to} \quad v \in \{-1,1\}^n, \\ &\qquad\qquad\quad e^T v = 0. \end{aligned} \tag{1.2}$$

Another type of graph partitioning problem is the maximum cover problem.

Let

$$\mathrm{E} = \{(i, j) \text{ such that an edge connects } v_i \text{ and } v_j\}.$$

The maximum cover problem ask for the largest set of vertices such that no edge connects two vertices in the set. In Figure 1, for example, vertices A, D, and E form a cover, but since an edge connects A and B, any set of vertices containing both A and B is not a cover. One formulation of the problem adds a super-vertex v_0 with no edges connecting and asks to solve the following problem.

$$\begin{aligned} &\text{Maximize} && \frac{1}{2}\left(\sum_{i=1}^{n} v_i^2 + v_0 v_i\right) \\ &\text{Subject to} && v \in \{-1, 1\}^{n+1}, \\ & && |v_0 + v_i + v_j| = 1 \text{ if } (v_i, v_j) \in \mathrm{E} \end{aligned} \tag{1.3}$$

A graph with no edges has an optimal solution of n, achieved by assigning all vertices the same sign. On the other hand, a fully connected graph would have a maximum maximum cover of 0. Assigning no vertex has the same sign as v_0 achieves this maximum objective value.

1.1.2 Nonconvex Optimization

Another example of a combinatorial optimization problem arises in nonconvex optimization. The box constrained quadratic problem (Box QP) can be posed as:

$$\begin{aligned} &\text{Minimize} && v^T C v \\ &\text{Subject to} && -e \le v \le e. \end{aligned} \tag{1.4}$$

The quadratic function Q is nonconvex and be be a second order approximation of a non-quadratic function. As a subproblem in a nonlinear optimization problem, the box may represent a trust region.

In nonconvex optimization, locally optimal point may not be a globally optimal. Figure 2 shows that the minimum of nonconvex quadratic functions often occur at corners. For an n variable problem, the number of corners is 2^n, making an exhaustive

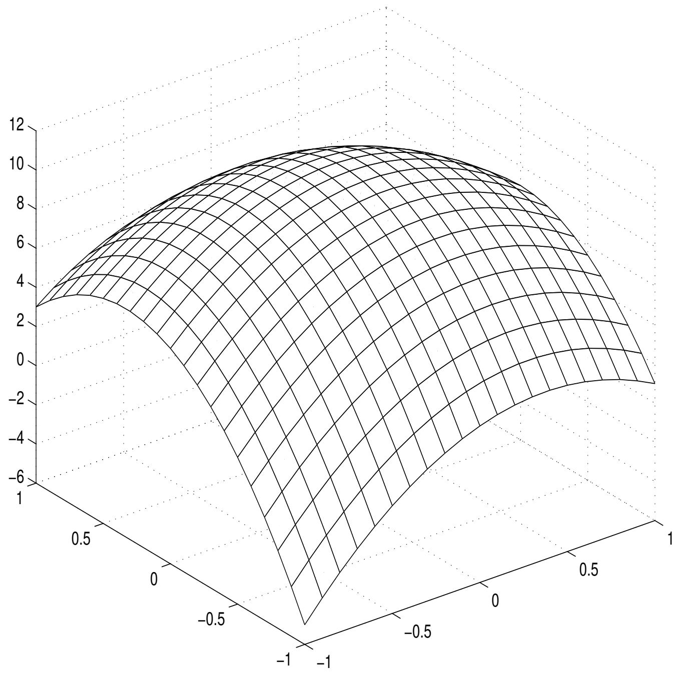

Figure 2: Nonconvex quadratic function

search practically impossible for large problems.

1.2 Positive Semidefinite Relaxation

One method for approximating the solutions to these problems is the positive semidefinite relaxation. The positive semi-definite relaxation was first proposed by Lovász and Shrijver [23]. Shor [32] proposed its use for 0-1 quadratic programming, and the field has received further contributions in papers by Sherali and Adams [31], Fujie and Kojima [13], Helmberg and Rendl [17], Karisch, Rendl, and Clausen [19], and Polijak, Rendl and Wolkowicz [30].

For two matrices $A, B \in \Re^{n\times n}$, let $A \bullet B = \sum_{i,j=1}^{n} A_{ij}B_{ij} = \mathrm{tr}(A^T B)$ where $\mathrm{tr}(\cdot)$ is the trace operator. Let $X = vv^T$ and use the identity $v^T C v = C \bullet (vv^T)$ to rewrite the box QP problem in the equivalent representation:

$$\begin{aligned} &\text{Minimize} && C \bullet X \\ &\text{Subject to} && \operatorname{diag}(X) \leq e \\ &&& \operatorname{Rank}(X) = 1. \end{aligned} \tag{1.5}$$

where $\operatorname{diag}(X)$ refers to the diagonal elements of X.

For the maximum cut problem (1.1), rewrite the objective function as

$$\begin{aligned} \frac{1}{2}\sum_{i<j} \tilde{C}_{ij}(1 - v_i v_j) &= \frac{1}{4}\sum_{i,j=1}^{n} \tilde{C}_{ij}(1 - v_i v_j) \\ &= \frac{1}{4}\left(\sum_{i,j=1}^{n} \tilde{C}_{ij} - \tilde{C} \bullet (vv^T)\right) \\ &= \frac{1}{4}\left(\operatorname{Diag}(\tilde{C}e) - \tilde{C}\right) \bullet (vv^T) \end{aligned} \tag{1.6}$$

The operator $\operatorname{Diag}(\cdot)$ turns a vector into a diagonal matrix. The first and last lines use the fact that the diagonal elements of vv^T equal one. The Laplacian matrix is defined by

$$C = \frac{1}{4}\left(\operatorname{Diag}(\tilde{C}e) - \tilde{C}\right), \tag{1.7}$$

let the matrix $X = vv^T$, and rewrite the maximum cut problem as

$$\begin{aligned} &\text{Maximize} && C \bullet X \\ &\text{Subject to} && \operatorname{diag}(X) = e \\ &&& \operatorname{Rank}(X) = 1. \end{aligned} \tag{1.8}$$

Linear constraints of the form $|h^T v| = \kappa$ with $\kappa \geq 0$ can be rewritten as $(hh^T) \bullet (vv^T) = \kappa^2$. The equal cut problem (1.2), for instance, can be written as

$$\begin{aligned} &\text{Maximize} && C \bullet X \\ &\text{Subject to} && \operatorname{diag}(X) = e \\ &&& (ee^T) \bullet X = 0 \\ &&& \operatorname{Rank}(X) = 1. \end{aligned} \tag{1.9}$$

The positive semidefinite relaxation of these problems relaxes the rank constraint to include all positive semidefinite matrices. The larger set is a convex cone.

The problem has a linear objective function and affine constraints, so this linearization of the nonlinear problem is polynomially solvable [29].

The optimal value of the relaxed problem will give a bound for the combinatorial problem. In some problems, only a bound is needed, but if a rank one solution is needed, the solution to the relaxed problem can be projected back into this set of rank one matrices. Recently, there have been several theoretical results on approximating combinatorial and quadratic optimization problems using semidefinite programming (see, e.g., Goemans and Williamson [16], Nesterov [28], and Ye [36]).

1.3 Positive Semidefinite Programming

Let $\mathcal{S}^n$ represent the space of symmetric $n \times n$ matrices, and for any $X \in \mathcal{S}^n$, let $X \succeq 0$ signify that X is positive semidefinite. A positive semidefinite program, in standard form is:

$$(SDP) \qquad \begin{array}{ll} \text{Minimize} & C \bullet X \\ \text{Subject to} & A_i \bullet X = b_i, \; i = 1, \ldots, m. \\ & X \succeq 0. \end{array} \tag{1.10}$$

where $A_i \in \mathcal{S}^n$, $C \in \mathcal{S}^n$, and $b \in \Re^m$ are given and $X \in \mathcal{S}^n$ is unknown. Furthermore, the A_i's are linearly independent.

Some of the combinatorial problems mentioned in Section 1.1 call for a maximum to the objective function. Those problems will be converted into this standard form by minimizing the negative of their objective value.

The dual of (SDP) can be written as:

$$(DSDP) \qquad \begin{array}{ll} \text{Maximize} & b^T y \\ \text{Subject to} & \sum_{i=1}^{m} y_i A_i + S = C, \quad S \succeq 0, \end{array} \tag{1.11}$$

where y_i, $i = 1, \ldots, m$ are scalar variables.

Also let the linear operator $\mathcal{A} : \mathcal{S}^n \to \Re^m$ be defined as

$$\mathcal{A}X = \begin{pmatrix} A_1 \bullet X \\ A_2 \bullet X \\ \vdots \\ A_m \bullet X \end{pmatrix}.$$

Since $(\mathcal{A}X)^T y = \sum_{i=1}^{m} (A_i \bullet X) y_i = (\sum_{i=1}^{m} y_i A_i) \bullet X$, the adjoint operator $\mathcal{A}^T : \Re^m \to \mathcal{S}^n$ is

$$\mathcal{A}^T y = \sum_{i=1}^{m} y_i A_i.$$

These two positive semidefinite programs are related by the following well known duality theorems. A detailed proof can be found in [29].

Theorem 1.1. *(Weak Duality) For any feasible point X of the primal program (SDP) and any feasible point (y, S) of its dual program (DSDP), $C \bullet X \geq b^T y$.*

Theorem 1.2. *(Strong Duality) Provided that (PSD) and (DSDP) are both feasible and there is a strictly interior point to either (PSD) or (DSDP), there is no duality gap. That is, if primal and dual optimal solutions are (X^*) and (y^*, S^*) respectively, then $C \bullet X^* = b^T y^*$.*

The duality theorems prove that an optimal solution, (X^*, y^*, S^*), $\mathrm{tr}X^*S^* = 0$. Since the trace of a matrix is the sum of its eigenvalues and the product of two positive semidefinite matrices will have nonnegative eigenvalues, all eigenvalues of X^*S^* must be zero. Of course, the only matrix with that property is the zero matrix. Hence $X^*S^* = 0$. Necessary and sufficient conditions for an optimal solution (X^*, y^*, S^*) are given by:

$$\mathcal{A}X^* - b = 0 \tag{1.12}$$

$$\mathcal{A}^T y^* + S^* - C = 0 \tag{1.13}$$

$$X^*S^* = 0 \tag{1.14}$$

Positive semidefinite matrices X and S satisfying the first two equalities are called feasible. This pair of semidefinite programs can be solved in polynomial time, and there are actually several polynomial time algorithms.

Primal dual algorithms have proven to be the most effective method for solving large linear programming programs. These methods use the primal and dual points to generate search directions $\Delta X, \Delta y$, and ΔS. In the past several years, there has been a lot of research concerning ways extend these methods to positive semidefinite programming. The two problems are both examples of convex optimization problems over the intersection of a cone and an affine set [29].

Starting with a feasible initial point, most primal dual methods use the symmetrized Newton equations:

$$\mathcal{A}\Delta X = 0 \tag{1.15}$$

$$\mathcal{A}^T\Delta y + \Delta S = 0 \tag{1.16}$$

$$\mathcal{E}\Delta X + \mathcal{F}\Delta S = \sigma\mu I - \mathcal{H}_P(XS) \tag{1.17}$$

where $\mu = X \bullet S/n$ and ρ is a centrality parameter. Feasible points that solve the perturbed optimality equation

$$XS = \mu I$$

comprise the central path, and most primal dual algorithms try to follow this central path to an optimal point.

The symmetrization operator is

$$\mathcal{H}_P(N) = \frac{1}{2}\left(PNP^{-1} + P^{-1}NP\right)$$

for an invertible matrix P. The linear operators $\mathcal{E}$ and $\mathcal{F}$ are defined by

$$\mathcal{E}(N) = \frac{1}{2}\left(PNSP^{-1} + P^{-1}SNP\right)$$

$$\mathcal{F}(N) = \frac{1}{2}\left(PXNP^{-1} + P^{-1}NXP\right)$$

This system of equations (1.15) - (1.17) has $2n^2 + m$ variables and equations.

Directly solving this system would be extremely difficult for a large problem: a maximum cut problem with 1000 vertices would lead to a system of over a million equations. To reduce the size of the system, use (1.16) to solve for ΔS and plug it into (1.17). Solving for ΔX in (1.17) allows (1.15) to be written as the Schur complement equation

$$M\Delta y = h$$

where $M_{ij} = A_i \bullet \mathcal{E}^{-1}\mathcal{F}(A_j)$ for $i, j = 1, \ldots, m$ and

$$h = \mathcal{A}(\mathcal{E}^{-1}(\sigma\mu I - \mathcal{H}_P(XS))).$$

The other variables subsequently get computed using

$$\Delta S = -\mathcal{A}^T(\Delta y)$$

and

$$\Delta X = \mathcal{E}^{-1}\left(\sigma\mu I - \mathcal{H}_P(XS) - \mathcal{F}(\Delta S)\right). \tag{1.18}$$

The matrix ΔS is symmetric since it is a linear combination of symmetric matrices. The primal matrix ΔX, however, involves a product of symmetric matrices and require symmetrization operators to ensure its symmetry.

The choice of P characterizes the dozens of primal dual methods that have been analyzed. The direction analyzed by Alizadeh, Haeberly, and Overton (AHO) [3] uses $P = I$. This simple choice of a symmetrization matrix is very effective, but it requires a considerable amount of computation. At several stages this direction requires the solution of Lyapunov equations of the form $WV + VW = 2R$.

Another step direction was analyzed independently by Helmberg, Rendl, Vanderbei, Wolkowitz; Kojima, Shida, Hara; and Monteiro (HKM) [18] [21] [24]. In this direction, $P = X^{\frac{1}{2}}$ or $P = S^{-\frac{1}{2}}$ gives (1.15) and (1.17) the form

$$\mathcal{A}\tilde{X} = 0$$

$$\Delta\tilde{X}S + X\Delta S = \sigma\mu I - (XS).$$

Then

$$\Delta\tilde{X} = \sigma\mu S^{-1} - X - X\Delta S S^{-1}$$

and $\Delta X = \frac{1}{2}(\Delta\tilde{X} + \Delta\tilde{X}^T)$. The simplicity and relatively low cost make this direction a very popular one.

A third primal dual method was developed by Nesterov and Todd [27]. It uses a scaling matrix P satisfying $(P^TP)^{-1} = X^{\frac{1}{2}}\left(X^{\frac{1}{2}}SX^{\frac{1}{2}}\right)^{-\frac{1}{2}}X^{\frac{1}{2}}$. The NT direction has the additional advantage that it can be generalized to convex optimization problems over the intersection of convex cones and affine sets.

Creating the Schur complement matrix M is usually the most time consuming part of semidefinite program. The best theoretical complexities found so far for primal dual directions found so far are by Monteiro and Zanjácomo [25]. They calculate the AHO direction using $5mn^3 + m^2n^2 + O(\max\{m, n\}^3)$ operations, the HKM direction in $2mn^3 + m^2n^2 + O(\max\{m, n\}^3)$ operations, and the NT direction [27] in $mn^3 + m^2n^2/2 + O(\max\{m, n\}^3)$ operations. These operation counts include additions and multiplications. The third order terms include the factorizations of X, S, and M.

A second class of algorithms for solving positive semidefinite programs are called primal algorithms. The primal-scaling algorithm (Nesterov and Nemirovskii [29], Alizadeh [1], Vandenberghe and Boyd [34], and Ye [37]) is the analogue of the primal potential reduction algorithm for linear programming. Instead of using the primal and dual matrices to determine the step direction, it uses only the primal matrix X.

A third type of algorithm for solving positive semidefinite programs is the dual-scaling algorithm (Vandenberghe and Boyd [34], Anstreicher and Fampa [4], and Ye [37]), which is the analogue of the dual-scaling algorithm for linear programming. The dual-scaling algorithm uses only S to generate the new iterate.

All of these algorithms possess $O(\sqrt{n}\log(1/\epsilon))$ iteration complexity to yield accuracy ϵ. Other scaling algorithms have been proposed in the past. For example an SDP equivalent of Dikin's affine-scaling algorithm could be very fast. However this algorithm may not converges in polynomial time. Muramatsu [26] showed an example in which this scaling algorithm will not converge to an optimal answer.

There are also quite few computational results and implementations of these interior algorithms, see K. M. Anstreicher and M. Fampa [4], F. Alizadeh, J. P. A. Haeberly, and M. L. Overton [3], K. Fujisawa, M. Kojima and K. Nakata [15], C. Helmberg, F. Rendl, R. J. Vanderbei, H. Wolkowicz [18], L. Vandenberghe and S. Boyd [34], H. Wolkowicz and Q. Zhao [35], etc. To the best of my knowledge, the largest problem that could be solved had $n = 628$. (After the initial version of this paper was submitted, two more implementations came out: K. Fujisawa, M. Fukuda, M. Kojima and K. Nakata [14] reported that they could solve a semidefinite program with $n = 1250$; C. Lin and R. Saigal [22] reported that they could solve the quadratic assignment semidefinite program with $n = 625$ with large m; both use powerful work-stations.)

The practical winner of solving large semidefinite programs is C. Helmberg and F. Rendl [17], an implementation of a non-interior-point algorithm called the spectral bundle method. The authors report solving a set of semidefinite programs with n up-to 3000. The spectral bundle method enables them to take advantages of the sparsity structure of these problems. The minor weakness of their method is that the method does not solve the primal problem and cannot guarantee or verify the optimality at its termination, and it is not a polynomial time algorithm.

Therefore, the open question is how to exploit the sparsity structure by polynomial interior-point algorithms so that they can also solve large-scale problems in practice. In this thesis I respond to this question and show that many large-scale semidefinite programs arising in combinatorial and quadratic optimization have features which make the dual scaling interior-point algorithm the most suitable choice for the following reasons.

1. The computational cost of each iteration in the dual algorithm is less than the cost of a primal-dual iteration. Although primal-dual algorithms possess superlinear convergence, their theoretical advantage may not be utilized since the approximation problems under consideration require less precision than some other applications. The dual scaling linear programming algorithm has been shown to perform equally well when only a lower precision answer is required, see, e.g., Vandenberghe and Boyd [34].

2. In most combinatorial applications, we need only a lower bound for the optimal objective value of (SDP). Solving (DSDP) alone would be sufficient to provide such a lower bound. Thus, we may not need to generate an X at all. Even if an optimal primal solution is necessary, the dual-scaling algorithm can generate an optimal X at the termination of the algorithm.

3. For large scale problems, S tends to be very sparse and structured since it is the linear combination of C and the A_i's. Dual algorithms can exploit this sparsity to save memory and computation time. The primal matrix, X, is usually much less sparse, so primal-dual algorithms cannot fully exploit this feature of large scale problems.

These problems include the semidefinite relaxations of graph partitioning problems and box constrained quadratic optimization problems. I will use these problems to illustrate the points above and report results of problems where n is up to 3000 and m is up to 7000.

CHAPTER II
DUAL SCALING ALGORITHM FOR POSITIVE SEMIDEFINITE PROGRAMMING

2.1 Dual Scaling Algorithm

The dual-scaling algorithm, which is a modification of the linear programming algorithm, reduces the Tanabe-Todd-Ye primal-dual potential function

$$\Psi(X, S) = \rho \ln(X \bullet S) - \ln\det X - \ln\det S.$$

The first term decreases the duality gap, while the second and third terms keep X and S in the cone of positive semidefinite matrices. When $\rho > n$, the minimum of the potential function occurs at an optimal solution.

Let $\bar{z} = C \bullet X$ for some feasible X and consider the dual potential function

$$\psi(y, \bar{z}) = \rho \ln(\bar{z} - b^T y) - \ln\det S.$$

Its gradient is

$$\nabla\psi(y, \bar{z}) = -\frac{\rho}{\bar{z} - b^T y} b + \mathcal{A}S^{-1}. \tag{2.19}$$

Each iteration of the algorithm will try to reduce the potential function as much as possible. To estimate the potential reduction from one iterate to the next, use a lemma from linear programming that can be found in [37] and is essentially due to Karmarkar [20].

Lemma 2.1. *Let $X \in \mathcal{S}^n$ and $\|X - I\|_\infty < 1$. Then*

$$\ln\det(X) \geq \operatorname{tr}(X - I) - \frac{\|X - I\|}{2(1 - \|X - I\|_\infty)}$$

where I denotes the identity matrix, $\|\cdot\|_\infty$ denotes the eigenvalue with the largest magnitude, and $\|\cdot\|$ denotes the Frobenius norm.

Proof. For $1 \le i \le n$, let $\lambda_i(X)$ be the eigenvalues of the matrix X and let $x_i = \lambda_i(X)$

$$\begin{aligned}
\ln x_i &= \ln(1 + x_j - 1) \\
&= (x_i - 1) - \tfrac{(x_i-1)^2}{2} + \tfrac{(x_i-1)^3}{3} - \tfrac{(x_i-1)^4}{4} + \dots \\
&\ge (x_i - 1) - \tfrac{(x_i-1)^2}{2}(1 + |x_i - 1| + |x_i - 1|^2 + \dots) \\
&= (x_i - 1) - \tfrac{(x_i-1)^2}{2(1-|x_i-1|)} \ge (x_i - 1) - \tfrac{(x_i-1)^2}{2(1-\|X-I\|_\infty)}
\end{aligned}$$

Summing the inequality over i, we have the result. □

Using the lemma above, the concavity of the first term in the potential function, and the fact that

$$(S^k)^{-.5} S (S^k)^{-.5} - I = (S^k)^{-.5} \left(\mathcal{A}^T(y - y^k)\right) (S^k)^{-.5}, \tag{2.20}$$

we want to apply the overestimater

$$\begin{aligned}
\psi(y, \bar{z}^k) - \psi(y^k, \bar{z}^k) &= \rho \ln(\bar{z}^k - b^T y) - \rho \ln(\bar{z}^k - b^T y^k) - \ln\det((S^k)^{-.5} S (S^k)^{-.5}) \\
&\le \rho \ln(\bar{z}^k - b^T y) - \rho \ln(\bar{z}^k - b^T y^k) + \operatorname{tr}((S^k)^{-.5} S (S^k)^{-.5} - I) \\
&\quad + \tfrac{\|(S^k)^{-.5}\left(\mathcal{A}^T(y-y^k)\right)(S^k)^{-.5}\|}{2(1-\|(S^k)^{-.5}\left(\mathcal{A}^T(y-y^k)\right)(S^k)^{-.5}\|_\infty)} \qquad (2.21) \\
&\le \nabla\psi(y^k, \bar{z}^k)^T (y - y^k) + \tfrac{\|(S^k)^{-.5}\left(\mathcal{A}^T(y-y^k)\right)(S^k)^{-.5}\|}{2(1-\|(S^k)^{-.5}\left(\mathcal{A}^T(y-y^k)\right)(S^k)^{-.5}\|_\infty)}
\end{aligned}$$

when $\|(S^k)^{-.5} \left(\mathcal{A}^T(y - y^k)\right) (S^k)^{-.5}\| \le 1$.

Lemma 2.2. *Thus, if $\alpha < 1$, then $S = C - \mathcal{A}^T y \succ 0$ for any y satisfying*

$$\|(S^k)^{-.5} \left(\mathcal{A}^T(y - y^k)\right) (S^k)^{-.5}\| \le \alpha.$$

Proof. The Frobenius norm of a matrix is the square root of the sum of the squares of its eigenvalues. Since the norm is less than 1, each eigenvalue has magnitude less than 1. Subtracting the identity matrix reduces each eigenvalue by 1, so using (2.20), $1 - \alpha \le \lambda_i \left((S^k)^{-.5} S (S^k)^{-.5}\right) \le 1 + \alpha$. If $\alpha < 1$, the eigenvalues are positive. Since S^k is positive definite, $S \succ 0$. □

Beginning with a strictly feasible dual point (y^k, S^k) and upper bound $\bar{z}^k$, each iteration of the potential reduction algorithm solves this following problem.

$$\begin{array}{ll} \text{Minimize} & \nabla\psi^T(y^k, \bar{z}^k)(y - y^k) \\ \text{Subject to} & \|(S^k)^{-.5}\left(\mathcal{A}^T(y - y^k)\right)(S^k)^{-.5}\| \le \alpha, \end{array} \tag{2.22}$$

where α is a positive constant less than 1. This constraint creates an ellipse that is contained in the feasible region.

Letting $\hat{A}_i = (S^k)^{-.5}A_i(S^k)^{-.5}$, the first order Karusch-Kuhn-Tucker conditions state that the minimum point, y^{k+1}, of this convex problem satisfies

$$\begin{pmatrix} \hat{A}_1 \bullet \hat{A}_1 & \cdots & \hat{A}_1 \bullet \hat{A}_m \\ \vdots & \ddots & \vdots \\ \hat{A}_m \bullet \hat{A}_1 & \cdots & \hat{A}_m \bullet \hat{A}_m \end{pmatrix} (y^{k+1} - y^k) + \beta\nabla\psi(y^k, \bar{z}^k) = 0 \tag{2.23}$$

for a positive value of β.

The matrix in (2.23) is a Gram matrix and is positive definite when $S \succ 0$ and the constraint matrices A_i are linearly independent. In this paper, it will be referred to as M.

Using the ellipsoidal constraint, the minimal solution, y^{k+1}, of (2.22) is given by

$$y^{k+1} - y^k = \frac{\alpha}{\sqrt{\nabla\psi^T(y^k, \bar{z}^k)M^{-1}\nabla\psi(y^k, \bar{z}^k)}} d(\bar{z}^k)_y \tag{2.24}$$

where

$$d(\bar{z}^k)_y = -M^{-1}\nabla\psi(y^k, \bar{z}^k). \tag{2.25}$$

Unlike linear programming, positive semidefinite programs require a significant amount of the time to compute the system of equations used to determine the step direction. Generally, $M_{ij} = A_iS^{-1} \bullet S^{-1}A_j$. When $A_i = a_ia_i^T$, the Gram matrix can be rewritten in the form

$$M = \begin{pmatrix} (a_1^T(S^k)^{-1}a_1)^2 & \cdots & (a_1^T(S^k)^{-1}a_m)^2 \\ \vdots & \ddots & \vdots \\ (a_m^T(S^k)^{-1}a_1)^2 & \cdots & (a_m^T(S^k)^{-1}a_m)^2 \end{pmatrix}. \tag{2.26}$$

This matrix can be computed more quickly without computing, or saving, $(S^k)^{-1}$. Instead, one of the methods below can be used to compute it.

Algorithm M: To compute M, factor S^k and do the following:

For $i = 1 : m$;

Solve $S^k w_i = a_i$.

For $j = i : m$; $M_{ij} = (a_j^T w_i)^2$; end;

end;

Since $A_i \bullet S^{-1} = a_i^T S^{-1} a_i$, the algorithm can also compute the vector $\mathcal{A}(S^{-1})$ for almost no additional effort.

The cost of Algorithm M is $\frac{1}{3}n^3 + 2n^2 m + nm^2 + O(nm)$ arithmetic operations. The first term comes from the factorization of S^k. The second term comes from the backward and forward substitutions used to solve the m systems of equations. Each solve uses $2n^2 + O(n)$ operations. Since there are $m(m+1)/2$ vector inner products, these multiplications comprise the third term of the complexity result.

When computer memory is not a limitation, the computation of M can be further reduced by eliminating the forward substitution in solving the systems of equations. Instead, if $S = LL^T$, where L is a lower triangular matrix, we could solve, and store, each vector $\tilde{w}_i$ satisfying $L^T \tilde{w}_i = a_i$. Then $M_{ij} = (\tilde{w}_i^T \tilde{w}_j)^2$. For large problems, however, the RAM used to store these vectors is more valuable than the computational effort it saves.

For arbitrary symmetric matrices A_i, the complexity of computing M rises an order of n and dominates the other steps of the algorithm.

To find a primal point, solve the least squares problem

$$\begin{array}{ll} \text{Minimize} & \|(S^k)^{.5} X (S^k)^{.5} - \frac{\bar{z}^k - b^T y^k}{\rho} I\| \\ \text{Subject to} & \mathcal{A}X = b. \end{array} \tag{2.27}$$

This problem looks for a feasible matrix $X(\bar{z}^k)$ near the central path. Larger values of ρ generally give a better objective value than lower values of ρ, but they also provide a solution that more frequently is not positive semidefinite. The answer is a by-product of computing (2.25), given explicitly by

$$X(\bar{z}^k) = \frac{\bar{z}^k - b^T y^k}{\rho}(S^k)^{-1}\left(\mathcal{A}^T d(\bar{z}^k)_y + S^k\right)(S^k)^{-1}. \tag{2.28}$$

Creating the primal matrix may be costly. However, the evaluation of the primal objective value $C \bullet X(\bar{z}^k)$ requires drastically less work.

$$\begin{aligned} C \bullet X(\bar{z}^k) &= b^T y^k + X(\bar{z}^k) \bullet S^k \\ &= b^T y^k + \operatorname{tr}\left(\tfrac{\bar{z}^k - b^T y^k}{\rho}(S^k)^{-1}\left(\mathcal{A}^T d(\bar{z}^k)_y + S^k\right)(S^k)^{-1}S^k\right) \\ &= b^T y^k + \tfrac{\bar{z}^k - b^T y^k}{\rho}\operatorname{tr}\left((S^k)^{-1}\mathcal{A}^T d(\bar{z}^k)_y + I\right) \\ &= b^T y^k + \tfrac{\bar{z}^k - b^T y^k}{\rho}\left(d(\bar{z}^k)_y^T \mathcal{A}(S^k)^{-1} + n\right) \end{aligned}$$

Since the vectors $\mathcal{A}(S^k)^{-1}$ and $d(\bar{z}^k)_y$ were found previously in the calculation of the dual step direction, the cost of computing a primal objective value is the cost of a dot product! The matrix $X(\bar{z})$ never gets computed, saving time and memory. Primal dual methods require far more work to compute the primal variables and objective value.

Defining

$$P(\bar{z}^k) = \frac{\rho}{\bar{z}^k - b^T y^k}(S^k)^{.5}X(\bar{z})(S^k)^{.5} - I, \tag{2.29}$$

we have the following lemma, adapted from Ye's [37] linear programming potential reduction algorithm.

Lemma 2.3. *Let* $\mu^k = \frac{\bar{z}^k - b^T y^k}{n}$, $\mu = \frac{X(\bar{z}^k) \bullet S^k}{n}$, $\rho \geq n + \sqrt{n}$, *and* $\alpha < 1$. *If*

$$\|P(\bar{z}^k)\| < min(\alpha\sqrt{\frac{n}{n+\alpha^2}}, 1-\alpha), \tag{2.30}$$

then the following three inequalities hold:

1. $X(\bar{z}^k) \succ 0$;

2. $\|(S^k)^{.5}X(\bar{z}^k)(S^k)^{.5} - \mu I\| \leq \alpha\mu$;

3. $\mu \le (1 - .5\alpha/\sqrt{n})\mu^k$.

Proof. The proofs are by contradiction. If the first inequality is false, then $(S^k)^{.5}X(\bar{z}^k)(S^k)^{.5}$ has at least one nonpositive eigenvalue, which by (2.29) implies that $P(\bar{z}^k)$ has an eigenvalue less than -1 and $\|P(\bar{z}^k)\| \ge 1$.

If the second inequality does not hold,

$$\begin{aligned}\|P(\bar{z}^k)\|^2 &= \|\tfrac{\rho}{n\mu^k}(S^k)^{.5}X(\bar{z}^k)(S^k)^{.5} - I\|^2\\ &= \|\tfrac{\rho}{n\mu^k}(S^k)^{.5}X(\bar{z}^k)(S^k)^{.5} - \tfrac{\rho\mu}{n\mu^k}I + \tfrac{\rho\mu}{n\mu^k}I - I\|^2\\ &= \|\tfrac{\rho}{n\mu^k}(S^k)^{.5}X(\bar{z}^k)(S^k)^{.5} - \tfrac{\rho\mu}{n\mu^k}I\|^2 + \|\tfrac{\rho\mu}{n\mu^k}I - I\|^2\\ &> \left(\tfrac{\rho\mu}{n\mu^k}\right)^2\alpha^2 + \left(\tfrac{\rho\mu}{n\mu^k} - 1\right)^2 n\\ &\ge \alpha^2\left(\tfrac{n}{n+\alpha^2}.\right)\end{aligned}$$

The last inequality is true because the quadratic term has a minimum at $\frac{\rho\mu}{n\mu^k} = \frac{n}{n+\alpha^2}$.

If the third inequality does not hold, then

$$\frac{\rho\mu}{n\mu^k} > \left(1 + \frac{1}{\sqrt{n}}\right)\left(1 - \frac{\alpha}{2\sqrt{n}}\right) \ge 1.$$

The fourth line in the inequality above leads to

$$\begin{aligned}\|P(\bar{z}^k)\|^2 &\ge \left(\tfrac{\rho\mu}{n\mu^k} - 1\right)^2 n\\ &\ge \left(\left(1 + \tfrac{1}{\sqrt{n}}\right)\left(1 - \tfrac{\alpha}{2\sqrt{n}}\right) - 1\right)^2 n\\ &= \left(1 - \tfrac{\alpha}{2} - \tfrac{\alpha}{2\sqrt{n}}\right)^2\\ &\ge (1-\alpha)^2.\end{aligned}$$

□

Focusing on the expression $P(\bar{z}^k)$, it can be rewritten as

$$\begin{aligned}P(\bar{z}^k) &= \tfrac{\rho}{\bar{z}^k - b^T y^k}(S^k)^{.5}\left(\tfrac{\bar{z}^k - b^T y^k}{\rho}(S^k)^{-1}\left(\mathcal{A}^T d(\bar{z}^k)_y + S^k\right)(S^k)^{-1}\right)(S^k)^{.5} - I\\ &= (S^k)^{-.5}\left(\mathcal{A}^T d(\bar{z}^k)_y\right)(S^k)^{-.5}\end{aligned}$$

which by (2.22) and (2.24), makes

$$\begin{aligned}\|P(\bar{z}^k)\| &= \tfrac{\sqrt{\nabla\psi^T(y^k,\bar{z}^k)M^{-1}\nabla\psi(y^k,\bar{z}^k)}}{\alpha}\|(S^k)^{-.5}\left(\mathcal{A}^T(y - y^k)\right)(S^k)^{-.5}\|\\ &= \sqrt{\nabla\psi^T(y^k,\bar{z}^k)M^{-1}\nabla\psi(y^k,\bar{z}^k)}\\ &= \sqrt{-\nabla\psi^T(y^k,\bar{z}^k)d(\bar{z}^k)_y}.\end{aligned} \tag{2.31}$$

Furthermore

$$\nabla\psi^T(y^k,\bar{z}^k)(y^{k+1}-y^k) = -\alpha\|P(\bar{z}^k)\|^2. \tag{2.32}$$

Update the dual variables according to

$$y^{k+1} = y^k + \frac{\alpha}{\|P(\bar{z}^{k+1})\|}d(\bar{z})_y \qquad \text{and} \qquad S^{k+1} = C - \mathcal{A}^T y^{k+1}, \tag{2.33}$$

Using (2.21) and (2.32), the reduction in the potential function satisfies the inequality

$$\psi(y^{k+1},\bar{z}^k) - \psi(y^k,\bar{z}^k) \leq -\alpha\|P(\bar{z}^k)\| + \frac{\alpha^2}{2(1-\alpha)}. \tag{2.34}$$

When $\alpha < 1$, Lemma 2.2 assures the new point will be feasible. Larger values of α increase the step size and can significantly accelerate the convergence of the algorithm. However, larger steps can also step outside the cone of positive definite matrices. Checking the feasibility of the new point requires a factorization of S.

One commonly used for technique for determining a stepsize [2], [10], [33] finds the exact distance to the boundary of the feasible set and then scales the exact stepsize back by a constant less than one. An exact line search involves computing the minimum eigenvalue of the matrix $(S^k)^{-.5}\Delta S(S^k)^{-.5}$. This calculation requires an additional matrix factorization, which is very time consuming for large scale problems.

A technique more suitable for large scale problems uses an inexact line search. The stepsize can be increased or decreased until an acceptable stepsize is found. Check the feasibility of a new point by checking if all the pivots in a Cholesky factorization are positive. Since this factorization is needed in the next iterate anyway, a large step that stays within the cone of feasible matrices may cost nothing.

This factorization of S can also be used to evaluate the dual potential function. If $S = LDL^T$, then $\det S = \Pi_{i=1}^n D_{ii}$. Hence, the value of α can be chosen to minimize the potential function is dual step direction.

The theoretical algorithm can be stated as follows.

DUAL ALGORITHM. Given an upper bound $\bar{z}^0$ and a dual point (y^0, S^0) such that $S^0 = C - \mathcal{A}^T y^0 \succ 0$, set $k = 0$, $\rho > n + \sqrt{n}$, $\alpha \in (0, 1)$, and do the following:

while $\bar{z}^k - b^T y^k \geq \epsilon$ **do**

begin

1. Compute $\mathcal{A}(S^k)^{-1}$ and the matrix M (2.26).

2. Solve (2.25) for the dual step direction $d(\bar{z}^k)_y$.

3. Calculate $\|P(\bar{z}^k)\|$ using (2.31).

4. **if** (2.30) is true, **then** $X^{k+1} = X(\bar{z}^k)$, $\bar{z}^{k+1} = C \bullet X^{k+1}$, and $(y^{k+1}, S^{k+1}) = (y^k, S^k)$.

   else $y^{k+1} = y^k + \frac{\alpha}{\|P(\bar{z}^k)\|} d(\bar{z}^{k+1})_y$, $S^{k+1} = C - \mathcal{A}^T y^{k+1}$, $X^{k+1} = X^k$, and $\bar{z}^{k+1} = \bar{z}^k$.

   endif

5. $k := k + 1$.

end

The following potential reduction theorem is based on the above lemma:

Theorem 2.4.

$$\Psi(X^{k+1}, S^{k+1}) \leq \Psi(X^k, S^k) - \delta$$

where $\delta > 1/50$ *for a suitable* α.

Proof. Let $\alpha = 0.3$.

$$\begin{aligned} \Psi(X^{k+1}, S^{k+1}) - \Psi(X^k, S^k) &= \Psi(X^{k+1}, S^{k+1}) - \Psi(X^{k+1}, S^k) \\ &\quad + \Psi(X^{k+1}, S^k) - \Psi(X^k, S^k) \end{aligned}$$

In each iteration, one of the differences is zero. If $\|P(\bar{z})\|$ is large, violating (2.30), the dual variables get updated and by (2.34), the potential function decreases by an amount satisfying:

$$\psi(y^{k+1},\bar{z}^k)-\psi(y^k,\bar{z}^k)\leq -(.3)(.3)\sqrt{\frac{1}{1+(.3)^2}}+\frac{(.3)(.3)}{2(1-0.3)}. \tag{2.35}$$

Hence,

$$\Psi(X^{k+1},S^{k+1})-\Psi(X^{k+1},S^k)=\psi(y^{k+1},\bar{z}^k)-\psi(y^k,\bar{z}^k)<-0.022.$$

On the other hand, if $\|P(\bar{z})\|$ satisfies (2.30) the primal matrix gets updated and

$$\begin{aligned}
&n\ln\left(X^{k+1}\bullet S^k\right)-\sum_{i=1}^n \ln\lambda_i\left(X^{k+1}S^k\right)\\
&=n\ln\left(X^{k+1}\bullet S^k/\mu\right)-\sum_{i=1}^n \ln\lambda_i\left(X^{k+1}S^k/\mu\right)\\
&=n\ln n-\sum_{i=1}^n \ln\lambda_i\left((S^k)^{.5}X^{k+1}(S^k)^{.5}/\mu\right)\\
&\leq n\ln n+\tfrac{\|(S^k)^{.5}X^{k+1}(S^k)^{.5}/\mu-I\|}{2\left(1-\|(S^k)^{.5}X^{k+1}(S^k)^{.5}/\mu-I\|_\infty\right)}\\
&\leq n\ln n+\tfrac{\alpha^2}{2(1-\alpha)}\\
&\leq n\ln\frac{\frac{\sum\lambda_i(X^{k+1}S^k)}{n}}{(\Pi\lambda_i(X^{k+1}S^k))^{\frac{1}{n}}}n+\frac{\alpha^2}{2(1-\alpha)}\\
&=n\ln\left(X^k\bullet S^k\right)-\sum_{i=1}^n \ln\lambda_i\left((S^k)^{.5}X^k(S^k)^{.5}\right)+\frac{\alpha^2}{2(1-\alpha)}
\end{aligned}$$

The fifth line is true since the arithmetic mean of positive numbers is greater that the geometric mean. By the third part of Lemma 2.3

$$\sqrt{n}\left(\ln\det((S^k)^{.5}X^{k+1}(S^k)^{.5})-\ln\det((S^k)^{.5}X^k(S^k)^{.5})\right)=\sqrt{n}\ln\frac{\mu}{\mu^k}\leq-\frac{\alpha}{2}$$

Adding the two inequalities gives

$$\Psi(X^{k+1},S^{k+1})\leq\Psi(X^k,S^k)-\frac{\alpha}{2}+\frac{\alpha^2}{2(1-\alpha)}$$

When $\alpha=0.3$,

$$\Psi(X^{k+1},S^{k+1})-\Psi(X^k,S^k)\leq-0.085$$

□

This theorem leads to

Corollary 2.5. *Let $\rho \geq n + \sqrt{n}$. Then, the algorithm terminates in at most $O((\rho - n)\log(1/\epsilon))$ iterations.*

Proof. In $O((\rho - n)\log(1/\epsilon))$ iterations,

$$\Psi(X^k, S^k) \leq (\rho - n)\log(\epsilon)).$$

Also,

$$(\rho - n)\ln(C \bullet X^k - b^T y^k) = (\rho - n)\ln(X^k \bullet S^k) \leq \Psi(X^k, S^k) - n\ln n \leq \Psi(X^k, S^k).$$

Combining the two inequalities,

$$C \bullet X^k - b^T y^k = X^k \bullet S^k < \epsilon.$$

□

Proposition 2.6. *An iteration of the dual algorithm uses fewer arithmetic computations that an iteration of a primal dual algorithm.*

Proof. The computation of S and $S + \mathcal{A}^T d(\bar{z}^k)_y$ uses m matrix additions and mn^2 operations apiece since they are symmetric and $A_i = a_i a_i^T$. Factoring it uses $\frac{n^3}{3}$ operations. Creating the Gram matrix uses $nm^2 + n^2 m + O(nm)$ operations, and solving the system of equations uses $\frac{1}{3}m^3$ operations. The vector inner products for $\bar{z}$ and $\|P(\bar{z}^k)\|$, and the calculation of y^{k+1} use only $O(m)$ operations for a total of $\frac{1}{3}n^3 + nm^2 + 2n^2 m + \frac{1}{3}m^3 + O(mn)$ operations. In practical implementations both S, and $S + \mathcal{A}^T(d(\bar{z}^k))$ get computed. In conjuction with Algorithm M, the arithmetic complexity is $\frac{1}{3}n^3 + nm^2 + 4n^2 m + \frac{1}{3}m^3 + O(mn)$ operations.

Primal dual iterations require additional computations. First, the various Schur complement matrices used to compute the step directions cost more to compute than the Gram matrix used in this dual algorithm. Constraint matrices of the form $A_i = a_i a_i^T$ reduce the complexity of primal-dual algorithms by a factor of n, but even the quickest direction to compute, the HKM direction, takes about twice as long as the dual direction. The methods proposed by Monteiro and Zanjácomo [25], however, cannot take advantage of sparsity. In fact, the low computational complexity gets achieved at the cost of storing additional representations of the data and variable matrices. These methods appear unsuitable for large scale problems where computer memory is a major restriction.

Implementations of primal dual steps that can exploit some sparsity in the data matrices were developed by Fujisawa, Kojima and Nakata [15], who developed a technique for computing primal-dual step directions that can exploit some of the sparsity of the data matrices. The worst case complexity of these methods is higher than the complexities of Monteiro and Zanjácomo and higher than the complexity of the dual scaling algorithm.

Also important are the third order steps of the algorithm which no longer get dominated by higher order routines. In particular, the primal step direction involves the product of three matrices (1.18), which can be very expensive. Additionally, primal-dual algorithms require a factorization of the primal matrix X^k. These operations cost nothing in the dual algorithm since it does not have to compute, factor, or store the primal matrix. It uses the primal matrix only for proving convergence and complexity. If needed, though, the dual algorithm can generate an X^k as a by-product from (2.28). $\square$

Proposition 2.7. *The memory required by the dual algorithm is less than the memory required for primal dual algorithms.*

Proof. Aside from storing the data, which is the same for all algorithms, the dual algorithm needs to store the upper (or lower) half of matrices M and S. The same space can be used to store their factorizations. These matrices use $n(n+1)/2+m(m+1)/2$ double precision variables. The matrix $\mathcal{A}^T d(\bar{z}^k)_y + S$ can be stored in the same space as S since it is needed only after the dual matrix is needed. Additional vectors of length m store y and $d(\bar{z}^k)_y$.

Primal dual algorithms require the space mentioned above for the dual algorithms, and they also require the storage of matrices X and the primal step direction ΔX which require a total of $n(n+1)$ double precision variables. $\square$

For large scale problems, the memory complexity is as important as the computational complexity. A symmetric 5000×5000 matrix, for example, stored in double precision uses $5000 \times 5000 \times 8 \div 2 = 100$ MB RAM. This amount exceeds the capacity of most computers and emphasizes the need to store the minimal amount of data necessary.

2.2 Exploiting Sparsity: The Maximum Cut Problem

The positive semidefinite relaxation of the maximum cut problem, in the standard form of a minimization problem, is

$$\begin{array}{ll} \text{Minimize} & C \bullet X \\ \text{Subject to} & \operatorname{diag}(X) = e, \\ & X \succeq 0. \end{array} \tag{2.36}$$

The dual program can be expressed as:

$$\begin{array}{ll} \text{Maximize} & e^T y \\ \text{Subject to} & \operatorname{Diag}(y) + S = C, \quad S \succeq 0, \end{array} \tag{2.37}$$

The operator $\operatorname{diag}(\cdot)$ takes the diagonal of a matrix and makes it a vector, while the operator $\operatorname{Diag}(\cdot)$ forms a diagonal matrix from a vector.

Many examples of the maximum cut problems have a very sparse matrix C. Each of the constraint matrices A_i has only one nonzero entry. Since S is a linear combination of sparse matrices, it possesses a sparse structure that can be exploited in several phases of the algorithm.

Applying the dual scaling algorithm to this relaxation,

$$\nabla\psi(y^k,\bar{z}^k) = -\frac{\rho}{\bar{z}^k - e^T y^k} e + \mathrm{diag}((S^k)^{-1})$$

and

$$M = (S^k)^{-1} \circ (S^k)^{-1}, \tag{2.38}$$

where $\circ$ represents the Hadamard product. That is, $M_{ij} = \left((S^k)^{-1}_{ij}\right)^2$. When the graph represented by C is connected, M is dense – even when C is sparse.

Although sparsity does not help the factorization of M, it can drastically reduce the time required to factor S and create M. Recall from Theorem 2.6 that the creation of M is the most expensive part of the computation. In the maximum cut problem, when $m = n$, the creation of M may cost up to nine times more that its factorization. When the factorization of S is sparse, the back and forward substitutions, repeated m times, are much quicker. Even the cost of the vector inner products can be reduced because each vector a_i has only one nonzero entry. The sparse structure of S remains constant for all iterations. Reordering S at the beginning of the algorithm to reduce fill-in during Cholesky factorization may drastically speed up its factorization and the many forward and back substitutions required to compute the Gram matrices at every iteration.

The step direction $d(\bar{z}^k)_y$ of (2.25) is comprised of two parts:

$$dy_1 = M^{-1}e \tag{2.39}$$

$$dy_2 = M^{-1}\mathrm{diag}((S^k)^{-1}) \tag{2.40}$$

so that

$$d(\bar{z}^k)_y = \frac{\rho}{\bar{z}^k - e^T y^k} dy_1 - dy_2 \tag{2.41}$$

Since the dual step depends upon the upper bound $\bar{z}$, splitting the step direction into these two parts allows the algorithm more flexibility and the use of a better upper bound.

The step size uses the value

$$\|P(\bar{z}^k)\| = \sqrt{-\nabla\psi^T(y^k, \bar{z}^k)^T d(\bar{z}^k)_y}. \tag{2.42}$$

If $\|P(\bar{z}^k)\|$ is sufficiently small, Lemma 2.3 guarantees a good upper bound. From (2.28), the primal matrix

$$X(\bar{z}^k) = \frac{\bar{z}^k - e^T y^k}{\rho}(S^k)^{-1}\left(\mathrm{Diag}(d(\bar{z}^k)_y) + S^k\right)(S^k)^{-1}.$$

Frequently, an improved primal objective value can be found for larger values of $\|P(\bar{z}^k)\|$. Since $S^k \succ 0$, $X(\bar{z}^k)$ is positive definite if and only if

$$\left(\mathrm{Diag}(d(\bar{z}^k)_y) + S^k\right) \succ 0. \tag{2.43}$$

To check if $\mathrm{Diag}(d(\bar{z}^k)_y) + S^k \succ 0$, attempt a Cholesky factorization and simply check if its pivots are all positive. Stop the factorization process when it encounters a negative or zero pivot, and conclude that the matrix is not positive definite. Note that $\mathrm{Diag}(d(\bar{z}^k)_y) + S^k$ has the same sparsity pattern as S^k, allowing it to share the same data structure and efficient ordering.

The upper bound can be calculated by

$$\bar{z}^{k+1} = C \bullet X(\bar{z}^k) = e^T y^k + \frac{\bar{z}^k - b^T y^k}{\rho}\left(\mathrm{diag}((S^k)^{-1})^T d(\bar{z}^k)_y + n\right). \tag{2.44}$$

if (2.43) is positive definite. Otherwise, $\bar{z}^{k+1} = \bar{z}^k$.

An improved upper bound can modify the dual step direction calculated in (2.41) which is why the step direction was divided into two parts. The dual variables will be updated by

$$y^{k+1} = y^k + \frac{\alpha}{\|P(\bar{z})\|} d(\bar{z}^{k+1})_y \quad \text{and} \quad S^{k+1} = C - \mathrm{Diag}(y^{k+1}).$$

If $\alpha < 1$, Lemma 2.2 guarantees that $C - \mathrm{Diag}(y^{k+1}) \succ 0$. Larger step sizes can further reduce the potential function and accelerate the convergence of the algorithm.

Factoring a new dual matrix S checks its feasibility allows for a quick evaluation of the dual potential function. Since the matrix S is sparse and well ordered, increasing the step size may cost very little. These factorizations cost far less than the creation and factorization of the dense M, so a reducuction in the number of iterations will save time.

We now state the algorithm.

DUAL ALGORITHM. Reorder C to reduce fill-in during Cholesky factorization. Set $\bar{z}^0 = C \bullet I$ and choose y^0 such that $S^0 = C - \mathrm{Diag}(y^0) \succ 0$. Set $k = 0$, $\alpha = .99$ and do the following:

while $\frac{\bar{z}^k - e^T y^k}{|e^T y^k| + 1} \geq \epsilon$ **do**

begin

1. Compute $\mathcal{A}(S^k)^{-1}$ the matrix M (2.38) using Algorithm M.

2. Solve (2.39) and (2.40) to compute the dual step direction $d(\bar{z}^k)_y$ (2.41).

3. Use (2.44) to compute a new upper bound $\bar{z}$.

4. **If** $\bar{z} < \bar{z}^k$ **and** $\left(\mathrm{Diag}(d(\bar{z}^k)_y) + S^k\right) \succ 0$,

   then let $\bar{z}^{k+1} = \bar{z}$ and recompute $d(\bar{z}^{k+1})_y$ using (2.41).

   else let $\bar{z}^{k+1} = \bar{z}^k$. **endif**

5. Compute $\|P(\bar{z}^{k+1})\|$ using (2.42).

6. Select $\beta \geq \alpha / \|P(\bar{z}^{k+1})\|$, so that $y^{k+1} = y^k + \beta d(\bar{z}^{k+1})_y$ and $S^{k+1} = C - \mathrm{Diag}(y^{k+1}) \succ 0$.

7. $k := k + 1$.

end

2.3 Dense Constraints: Unequal Cut Problem

The unequal cut problem has only one more constraint than the maximum cut problem, but the new constraint makes the dual slack matrix totally dense. The additional constraint arises from trying to separate the vertices of a graph into sets of a particular size. Its positive semidefinite relaxation is:

$$\begin{array}{llll} \text{Minimize} & C \bullet X & & \\ \text{Subject to} & \operatorname{diag}(X) & = e, & X \succeq 0 \\ & (ee^T) \bullet X & = \kappa^2, & \kappa \neq 0 \end{array} \tag{2.45}$$

and its dual program can be expressed as:

$$\begin{array}{lll} \text{Maximize} & e^T y + \kappa^2 \lambda & \\ \text{Subject to} & \operatorname{Diag}(y) + \lambda(ee^T) + S & = C, \quad S \succeq 0. \end{array} \tag{2.46}$$

The new constraint makes the matrix S totally dense. Additional consideration though, can preserve the advantages of sparsity demonstrated in the previous section.

For the purposes of analysis, use y to represent the dual variables corresponding to the sparse constraints, and let λ represent the dual variable corresponding to the dense constraint.

The dual algorithm will compute

$$\nabla_y \psi(y^k, \lambda^k, \bar{z}^k) = -\frac{\rho}{\bar{z}^k - e^T y^k - \kappa^2 \lambda^k} e + \operatorname{diag}((S^k)^{-1})$$
$$\nabla_\lambda \psi(y^k, \lambda^k, \bar{z}^k) = -\frac{\rho}{\bar{z}^k - e^T y^k - \kappa^2 \lambda^k} \kappa^2 + e^T (S^k)^{-1} e$$

and the matrix

$$M^k = \begin{pmatrix} (S^k)^{-1} \circ (S^k)^{-1} & (S^k)^{-1} e \circ (S^k)^{-1} e \\ e^T (S^k)^{-1} \circ e^T (S^k)^{-1} & (e^T (S^k)^{-1} e)^2 \end{pmatrix}. \tag{2.47}$$

A good ordering of the sparse matrix $C - \operatorname{Diag}(y)$ and the use of the Sherman-Morrison-Woodbury update formula provides a solution of equations in Algorithm M that retains the advantages of sparsity. In the case of this problem, if $\tilde{S} = C - \operatorname{Diag}(y)$, and

$$w_i = S^{-1}a_i = \left((\tilde{S})^{-1} - \frac{\lambda}{1+\lambda e^T\tilde{S}^{-1}e}\tilde{S}^{-1}ee^T\tilde{S}^{-1}\right)a_i.$$

The vector $e^T\tilde{S}^{-1}$ only needs to be computed once per iteration, so the update formula only has the additional cost of m vector products $e^T\tilde{S}^{-1}a_i$ and m vector additions $(\tilde{S})^{-1}a_i - \frac{\lambda e^T\tilde{S}^{-1}a_i}{1+\lambda e^T\tilde{S}^{-1}e}\tilde{S}^{-1}e$. Hence the cost is an additional $4nm$ operations, which may be insignificant compared to the savings offered by the sparsity. In general, if there are m_d dense constraints, the additional cost will be $4n * m * m_d + 2mm_d^2$.

The two parts of the step direction $d(\bar{z}^k)_y$ are:

$$\begin{bmatrix} dy_1 \\ d\lambda_1 \end{bmatrix} = (M^k)^{-1}\begin{bmatrix} e \\ \kappa^2 \end{bmatrix} \tag{2.48}$$

$$\begin{bmatrix} dy_2 \\ d\lambda_2 \end{bmatrix} = (M^k)^{-1}\begin{bmatrix} \operatorname{diag}((S^k)^{-1}) \\ e^T(S^k)^{-1}e \end{bmatrix} \tag{2.49}$$

so that

$$\begin{bmatrix} d(\bar{z}^k)_y \\ d(\bar{z}^k)_\lambda \end{bmatrix} = \frac{\rho}{\bar{z}^k - e^Ty^k - \kappa^2\lambda}\begin{bmatrix} dy_1 \\ d\lambda_1 \end{bmatrix} - \begin{bmatrix} dy_2 \\ d\lambda_2 \end{bmatrix}. \tag{2.50}$$

Again, the dual variables will be updated for some step-size $\beta \geq \frac{\alpha}{\|\overline{P}(\bar{z}^k)\|}$:

$$y^{k+1} = y^k + \beta d(\bar{z}^k)_y, \qquad \lambda^{k+1} = \lambda^k + \beta d(\bar{z}^k)_\lambda,$$

and

$$S^{k+1} = C - \operatorname{Diag}(y^{k+1}) - \lambda^{k+1}(ee^T).$$

A new primal point, $X(\bar{z}^k)$, can be composed using

$$X(\bar{z}^k) = \frac{\bar{z}^k - e^Ty^k - \kappa^2\lambda^k}{\rho}(S^k)^{-1}\left(\operatorname{Diag}(d(\bar{z}^k)_y) + d(\bar{z}^k)_\lambda\,(ee^t) + S^k\right)(S^k)^{-1},$$

and a new upper bound $C \bullet X(\bar{z}^k)$ is available using

$$\begin{aligned} C \bullet X(\bar{z}^k) &= e^Ty^k + \kappa^2\lambda^k \\ &+ \tfrac{\bar{z}^k - e^Ty^k - \kappa^2\lambda^k}{\rho}\left(\operatorname{diag}((S^k)^{-1})^T d(\bar{z}^k)_y + \left(e^T(S^k)^{-1}e\right)d(\bar{z}^k)_\lambda + n\right) \end{aligned} \tag{2.51}$$

As in the previous section, if

$$\left(\operatorname{Diag}(d(\bar{z}^k)_y) + d(\bar{z}^k)_\lambda\,(ee^t) + S^k\right) \succ 0, \tag{2.52}$$

then $X(\bar{z}^k) \succeq 0$, but unlike the previous section, the matrix is dense. Factoring the matrix will test the feasibility of $X(\bar{z}^k)$, but since it is dense, there is a quicker method.

Separate the dense constraint from the sparse ones and let

$$L\tilde{I}L^T = \left(\mathrm{Diag}(d(\bar{z}^k)_y - y^k) + C\right)$$

where $\tilde{I}$ is a diagonal matrix of positive or negatives ones. Then

$$\left(\mathrm{Diag}(d(\bar{z}^k)_y - y^k) + C\right) = L\left(\tilde{I} + \left(L^{-1}ee^TL^{-T}\right)\left(d(\bar{z}^k)_\lambda - \lambda\right)\right)L^T. \tag{2.53}$$

The middle term of the right hand side of (2.53) can be used to check the feasibility of $X(\bar{z}^k)$. If the number $d(\bar{z}^k)_\lambda - \lambda^k < 0$, then $\tilde{I}$ must equal the identity matrix for $X(\bar{z}^k) \succeq 0$. However, if $d(\bar{z}^k)_\lambda - \lambda^k > 0$, a maximum of one diagonal element of $\tilde{I}$ may be -1. In the event there is one negative diagonal element, $\tilde{I}_{ii}$, let $f = L^{-1}e$ and write the matrix as a sum of rank one matrices and the identity matrix:

$$\left(\tilde{I} + \left(L^{-1}ee^TL^{-T}\right)\left(d(\bar{z}^k)_\lambda - \lambda\right)\right) = I + (d(\bar{z}^k)_\lambda - \lambda^k)ff^T - 2e_ie_i^T$$

The following lemma shows how to inexpensively check whether $X(\bar{z}) \succeq 0$.

Lemma 2.8. *Let $W = I + \sum_{i=1}^{m_d} \gamma_i(w^i w^{i^T})$ where $w^i \in \Re^n$, $1 \le i \le m_d$, are linearly independent, I is the identity matrix, and $m_d < n$. The positive definiteness of W can be checked in $O(nm_d^2 + m_d^3)$ operations.*

Proof. The second part of W is generally a large dense matrix of low rank. A change of basis can transform it into a sparse matrix with zeros everywhere except for a small $m_d \times m_d$ sub matrix. The new matrix, $\tilde{W}$, has the same eigenvalues as W and can be used to quickly determine whether or not W is positive definite.

More specifically, construct an orthonormal basis in $\Re^n$ by first applying the Gram-Schmidt process to vectors w^i. For $i \le m_d$, let basis vector

$$\tilde{Q}_i = w^i - \sum_{k=1}^{i-1} \frac{\tilde{Q}_k^T w^i}{\tilde{Q}_k^T \tilde{Q}_k} \tilde{Q}_k.$$

For $i > m_d$, $\tilde{Q}_i$ can be any nonzero vector such $\tilde{Q}_i^T\tilde{Q}_j = 0$ $(i \neq j)$. Normalize these vectors by letting $Q_i = \frac{\tilde{Q}_i}{\|\tilde{Q}_i\|}$ and define the orthonormal matrix $Q = (Q_1 \ \cdots \ Q_n)$. Let

$$\begin{aligned}
\hat{W} &= Q^T W Q \\
&= Q^T I Q + Q^T \left(\sum_{k=1}^{m_d} \gamma_k (w^k {w^k}^T) \right) Q \\
&= I + \sum_{k=1}^{m_d} \gamma_k Q^T w^k {w^k}^T Q \\
&= I + \sum_{k=1}^{m_d} \gamma_k \begin{bmatrix} Q_1^T w^k \\ \vdots \\ Q_n^T w^k \end{bmatrix} \begin{bmatrix} Q_1^T w^k & \ldots & Q_n^T w^k \end{bmatrix}
\end{aligned}$$

so that $\hat{W}_{ij} = I_{ij} + \sum_{k=1}^{m_d} \gamma_k (Q_i^T w^k)(Q_j^T w^k)$. By this construction of Q, $Q_i^T w^k = 0$ for all $i > m_d$. Therefore, W is the sum of the identity matrix and a smaller, $m_d \times m_d$, matrix. Factoring the small matrix uses only $O(m_d^3)$ operations. Usually m_d is very small: in the unequal cut problem, m_d equals 1 or 2.

This process does not require computing the matrix Q. Instead, the dot products $({w^i}^T w^j)$ and the following equations can be used to calculate the matrix.

$$\begin{aligned}
Q_i^T w^j &= \left(\tilde{Q}_i^T w^j \right) / \|\tilde{Q}_i\| \\
&= \left({w^i}^T w^j - \sum_{k=1}^{i-1} \frac{(\tilde{Q}_k^T w^i)(\tilde{Q}_k^T w^j)}{\tilde{Q}_k^T \tilde{Q}_k} \right) / \|\tilde{Q}_k\|
\end{aligned}$$

The norm $\|\tilde{Q}_i\|$ is given by

$$\begin{aligned}
\tilde{Q}_i^T \tilde{Q}_i &= \tilde{Q}_i^T \left(w^i - \sum_{k=1}^{i-1} (\tilde{Q}_k^T w^i) \tilde{Q}_k \right) \\
&= \tilde{Q}_i^T w^i - \sum_{k=1}^{i-1} (\tilde{Q}_k^T w^i)(\tilde{Q}_i^T \tilde{Q}_k) \\
&= \tilde{Q}_i^T w^i
\end{aligned}$$

The following algorithm computes the matrix $\hat{W}$ by first computing the quantities $\tilde{Q}_i^T w^j$, and then the matrix $\hat{W}$. Since $Q_i^T w^j = 0$ for $i < j$, and $\tilde{W}$ is symmetric, the only memory it requires is the upper half of a matrix $\tilde{W} \in \mathcal{S}^{m_d}$, a vector $q \in \Re^{m_d}$, and a temporary scaler ζ.

Change of Basis Algorithm

begin

for $i = 1 : m_d$; (* Compute vectors $\tilde{Q}_i^T w_j$ *)

for $j = i : m_d$;

$\tilde{W}(i,j) = w^{i^T} w^j$;

for $k = 1 : i-1$; $\tilde{W}(i,j) = \tilde{W}(i,j) - \tilde{W}(k,i) * \tilde{W}(k,j) / \tilde{W}(k,k)$; end;

end;

$q(i) = \sqrt{\tilde{W}(i,i)}$; (* The norm of $\tilde{Q}$ *)

end;

for $i = 1 : m_d$; (* Compute $\sum_{k=1}^{m_d} \gamma_k (Q_i^T w^k)(Q_j^T w^k)$ *)

for $j = i : m_d$;

$\zeta = 0$;

for $k = j : m_d$; $\zeta = \zeta + \gamma_i * \tilde{W}(i,k) * \tilde{W}(j,k)$; end;

$\tilde{W}(i,j) = \zeta / (q(i) * q(j))$;

end;

$\tilde{W}(i,i) = \tilde{W}(i,i) + 1$; (* Add identity matrix *)

end;

If $\tilde{W} \succeq 0$, return yes, $W \succeq 0$.

Else return no, W is not positive semidefinite.

end.

The complexity of this algorithm comes from the $m \times (m-1) \div 2$ dot products, the nested loops, and the factorization of $\tilde{W}$. $\square$

For the unequal cut problem, there are further simplifications. If $(d(\bar{z}^k)_\lambda - \lambda^k) <$

0 and $\tilde{I} = I$, $X(\bar{z}^k) \succeq 0$ if $\|L^{-1}e\|^2(d(\bar{z}^k)_\lambda - \lambda^k) \geq -1$. On the other hand, if $(d(\bar{z}^k)_\lambda - \lambda^k) > 0$ and there exists one negative diagonal element, $\tilde{I}_{ii}$, $X(\bar{z}^k) \succeq 0$ if $\left(d(\bar{z}^k)_\lambda - \lambda^k\right)(2(L^{-1}e)_i^2 - \|L^{-1}e\|^2) > 1$.

2.4 Mixed Positive Semidefinite and Linear Programming: The Box Constrained Quadratic Problem

The positive semidefinite relaxation of the box constrained quadratic problem is

$$\begin{aligned} \text{Minimize} \quad & C \bullet X \\ \text{Subject to} \quad & \text{diag}(X) \leq e, \\ & X \succeq 0 \end{aligned} \tag{2.54}$$

By adding slack variables, the problem can be written in the form

$$\begin{aligned} \text{Minimize} \quad & C \bullet X \\ \text{Subject to} \quad & \text{diag}(X) + x = e, \\ & X \succeq 0, x \geq 0. \end{aligned} \tag{2.55}$$

In fact, this problem is just an example of a linear and semidefinite program in mixed form (MP):

$$\begin{aligned} \text{Minimize} \quad & C \bullet X + c^T x \\ \text{Subject to} \quad & A_i \bullet X + \hat{a}_i^T x = b_i, \quad i = 1, \ldots, m. \\ & X \succeq 0, \ x \geq 0. \end{aligned} \tag{2.56}$$

where A_i, $X \in \mathcal{S}^n$ and $\hat{a}_i$,$x \in \Re^p$. Algorithms for solving these problems can be derived by embedding this problem into another, larger, positive semidefinite program.

Transform vectors $c, \hat{a}_i$, and x to a diagonal matrices and define the symmetric $(n+p) \times (n+p)$ matrices

$$\overline{C} = \begin{pmatrix} C & 0 \\ 0 & \text{Diag}(c) \end{pmatrix}, \ \overline{A}_i = \begin{pmatrix} A_i & 0 \\ 0 & \text{Diag}(\hat{a}_i) \end{pmatrix}, \ \overline{X} = \begin{pmatrix} X & 0 \\ 0 & \text{Diag}(x) \end{pmatrix}. \tag{2.57}$$

to pose (MP) as a "pure" semidefinite program.

For any point $X \succeq 0$ and $x \geq 0$ such that $A_i \bullet X + \hat{a}_i^T x = b_i$, $\overline{X} \succeq 0$ and

$\overline{A}_i \bullet \overline{X} = b_i$. Using a constraint matrix A with rows composed of the vectors $\hat{a}_i$, the new program may contain even more sparsity than the problems discussed earlier. The dual of (MP) can be written as:

$$\begin{array}{lrl} \text{Maximize} & b^T y & \\ \text{Subject to} & \sum_{i=1}^{m} y_i A_i + S & = C, \quad S \succeq 0, \\ & A^T y + s & = c, \quad s \geq 0. \end{array} \tag{2.58}$$

The dual potential function becomes:

$$\psi(y, \bar{z}) = \rho \ln(\bar{z} - b^T y) - \ln \det S - \sum_{i=1}^{p} \ln s_i.$$

Recall the two main areas upon which sparsity can be exploited, are the factorization of S and the calculation of $\mathcal{A}S^{-1}$ and M. It is no additional work to invert the larger matrix since its additional size has nonzeros only along the diagonal. Defining $\overline{S}$ as in (2.57), it is routine algebra to show that

$$\overline{A}_i \bullet \overline{S}^{-1} = A_i \bullet S^{-1} + \hat{a}_i^T \mathrm{Diag}(s)^{-1} e.$$

Furthermore,

$$M_{ij} = \overline{A}_i \overline{S}^{-1} \bullet \overline{S}^{-1} \overline{A}_j = A_i S^{-1} \bullet S^{-1} A_j + \hat{a}_i^T \mathrm{Diag}(s)^{-2} \hat{a}_j$$

The second half of the Gram matrix can be computed by vector multiplication whose cost, compared with the matrix multiplications in the first term, is relatively small. In linear programming, creating large matrices like M usually requires much less time than its factorization.

Once the system of equations (2.25) has been calculated, most of the program is the same. The step direction is

$$d(\bar{z}^k)_y = M^{-1} \nabla \psi^T (y^k, \bar{z}^k)$$

and

$$\|\overline{P}(\bar{z}^k)\| = \sqrt{-\nabla \psi^T (y^k, \bar{z}^k) d(\bar{z}^k)_y},$$

just like before, remembering that $\rho > n + p + \sqrt{n+p}$.

A new primal matrix is given by

$$X(\bar{z}^k) = \frac{\bar{z}^k - b^T y^k}{\rho}(S^k)^{-1}\left(\mathcal{A}^T d(\bar{z}^k)_y + S^k\right)(S^k)^{-1}, \tag{2.59}$$

and a new primal vector is given by

$$x(\bar{z}^k) = \frac{\bar{z}^k - b^T y^k}{\rho}\mathrm{Diag}(s^k)^{-2}\left(A^T d(\bar{z}^k)_y + s^k\right).$$

Neither $X(\bar{z}^k)$ nor $x(\bar{z}^k)$ needs to be calculated explicitly. However, unlike the matrix variable, the vector $x(\bar{z}^k)$ can be computed with minimal additional work. Primal dual methods dominate the area of linear programming because the primal variables can be computed almost for free.

In the case of the box constrained QP problem, the dual is (2.55)

$$\begin{array}{lll} \text{Maximize} & e^T y & \\ \text{Subject to} & \mathrm{Diag}(y) + S & = C, \quad S \succeq 0, \\ & y + s & = 0, \quad s \geq 0. \end{array} \tag{2.60}$$

The inequality constraints on this problem are looser than the maximum cut problem and the slack variables, $x(\bar{z}^k)$, create only a minimal amount of additional work. For this problem,

$$\nabla\psi_y(y^k, \bar{z}^k) = -\frac{\rho}{\bar{z}^k - b^T y^k}b + \mathrm{diag}((S^k)^{-1}) + \mathrm{Diag}(s^k)^{-1}e, \tag{2.61}$$

and the matrix

$$M^k = (S^k)^{-1} \circ (S^k)^{-1} + \mathrm{Diag}(s^k)^{-2}. \tag{2.62}$$

Adding linear variables adds very little cost to the algorithm. This is apparent from the sparsity inherent in matrix $\hat{S}$ where the linear variable resides. The gradient $\nabla\psi^T(y^k, \bar{z}^k)^T$ is easily modified and only the diagonal of the matrix M^k differs from the maximum cut problem. In linear programming, the factorization of the large matrix like M dominates the computation time. Creating the matrix uses very little work. Since our matrix already needs to be factored, the only additional cost is supplementing the matrix M.

2.5 Empty Interior: The Graph Bisection Problem

The graph bisection problem, or equal cut problem, asks for a partition that places an equal number of vertices into the two sets. Its semidefinite relaxation in standard form is:

$$\begin{array}{lrl} \text{Minimize} & C \bullet X & \\ \text{Subject to} & \operatorname{diag}(X) & = e \\ & (ee^T) \bullet X & = 0 \\ & X \succeq 0 & \end{array}$$

Since X and ee^T are both positive semidefinite matrices, $X \bullet (ee^T) = 0$ only if X has at least one zero eigenvalue. Since these matrices do not lie in the interior of the cone of positive semidefinite matrices, some additional considerations must be taken. To create an interior, add an artificial variable x and a large positive penalty parameter Λ, added to the objective function. The new problem is

$$\begin{array}{lrl} \text{Minimize} & C \bullet X + \Lambda x & \\ \text{Subject to} & \operatorname{diag}(X) & = e, \\ & (ee^T) \bullet X - x & = 0. \\ & X \succeq 0, x \geq 0 & \end{array} \tag{2.63}$$

This second representation allows $(ee^T) \bullet X > 0$, yet the he penalty parameter Λ forces it to be small. The previous section shows that the addition of variable x will not significantly slow the algorithm. Solving the graph bisection problem is almost identical to the unequal cut problem with $\kappa = 0$.

Another example of a problem with no interior is the $s - t$ cut problem. It requires that two vertices s and t are separated. Defining the vector e_{st} to have a 1 in indices s and t, and 0's elsewhere, its semidefinite relaxation requires $(e_{st}e_{st}^T \bullet X) = 0$. Alternatively, if both vertices are required to be placed in the same set, $(e_{st}e_{st}^T) \bullet X = 4$. The primal feasible set of this problem also does not have an interior. It can all be given an interior using the technique above. Starting from the original formulation

$$\begin{aligned} \text{Minimize} \quad & C \bullet X \\ \text{Subject to} \quad & \operatorname{diag}(X) &&= e, \\ & (e_{st} e_{st}^T) \bullet X &&= 4, \\ & X \succeq 0. \end{aligned} \tag{2.64}$$

¸ add the variable x and a penalty parameter Λ. The problem

$$\begin{aligned} \text{Minimize} \quad & C \bullet X + \Lambda x \\ \text{Subject to} \quad & \operatorname{diag}(X) &&= e, \\ & (ee^T) \bullet X + x &&= 4. \\ & X \succeq 0, x \geq 0 \end{aligned} \tag{2.65}$$

has an interior and can be solved using the dual scaling algorithm.

CHAPTER III
RANK REDUCTION

Combinatorial problems such as (1.5), (1.8), and (1.9) require a rank one matrix of the form $X = vv^T$. The solution to the positive semidefinite relaxation may have higher rank. In fact, the approximate solution returned by the positive semidefinite program will have full rank. After spending considerable effort solving the positive definite program, finding a rank one solution to the combinatorial problem should be fast and effective.

3.1 Maximum Cut and Box Constrained Quadratic Problem

Goemans and Williamson [16] used the following randomized algorithm to extract a vector v, and analyzed it with respect to the maximum cut problem.

Randomized Algorithm

1. Find a matrix V such that $X^* = V^T V$.

2. Randomly generate a vector $u \in \Re^n$ from the unit sphere.

3. Let $\hat{v} = V^T u$ and $v_i = sign(\hat{v}_i)$.

Because of the random element of this algorithm, it can be repeated arbitrarily often to generate different vectors $\hat{v}$.

To stay consistent with the standard form of the positive semidefinite program, assume the combinatorial problems are formulated to minimize the objective value. To measure the quality of the cut given by v, let $\overline{q}$ and $\underline{q}$ denote the the upper and lower bounds of the positive semidefinite relaxation. Define $q(v) = v^T Q v$ and an ϵ

optimal approximation by

$$\epsilon(v) = \frac{\overline{q} - q(v)}{\overline{q} - \underline{q}}.$$

Any feasible vector v provides at least a 1 optimal approximation. The best solution, however, may not be 0 optimal since the bound generated by the positive semidefinite relaxation may not be tight.

Goemans and Williamson [16] analyzed the expectation of the solution when all edges have positive weight. They proved that $\mathrm{E}[\epsilon(v)] < 0.122$ where $\mathrm{E}[\cdot]$ is the expectation operator and when u is randomly selected from the unit sphere. Nesterov [28] proved that, for arbitrarily weighted edges,

$$\mathrm{E}[\epsilon(v)] \leq 3/7.$$

This procedure is fast and simple. In practice it gets repeated numerous times and the law of large numbers almost guarantees an answer at least as good as the expectation.

Ye [36] extended the same quality bound to the box constrained QP problem by letting

$$v_i = \sqrt{X_{ii}} sign(\hat{v}_i), \ i = 1, \ldots, n.$$

The same technique and $\frac{3}{7}$ optimality bound can also be applied to the s-t cut problem. Recall that this problem requires vertices s and t to be placed in different sets. Ye [36] also showed that the vertices will be placed into different sets with a chance of 100%.

The combinatorial problems under consideration have an objective function $q(v) = C \bullet (vv^T)$. The expectation of $q(v)$ depends on $\mathrm{E}[v_i v_j]$. which depends upon the probability $Pr[sign(V_i^T u) = sign(V_j^T u)]$.

Each column V_i of V lies one the unit sphere since $V_i^T V_i = X_{ii} = 1$. Let θ_{ij} be the angle between the unit vectors V_i and V_j. Then $\hat{v}_i$ is the dot product between V_i and u. The chance that v_i and v_j will have the same sign is $\frac{2\theta_{ij}}{2\pi}$. Figure 3 shows vectors V_i, V_j, and u on the unit sphere and the angles between them. Using the figure and some trigonometry,

$$Pr(v_i = v_j) = 1 - \frac{1}{\pi}\arccos(V_i^T V_j)$$

and

$$\begin{aligned} \mathrm{E}[q(v)] &= Pr[v_i = v_j] + (1 - Pr[v_i = v_j])(-1) \\ &= 1 - \tfrac{2}{\pi}\arccos(V_i^T V_j) \\ &= \tfrac{2}{\pi}\arcsin(V_i^T V_j) \end{aligned} \quad .$$

3.2 Graph Bisection and Unequal Cut Problems

Applying the same algorithm to the positive semidefinite relaxation of the equal cut problem may not generate a feasible answer. To enforce linear constraints of the form $|h^T v| = \kappa$, let $\sigma(i)$ be the permutation function that sorts the vector in ascending order. That is, $\hat{v}_{\sigma(1)}$ is the smallest component in the vector and $\hat{v}_{\sigma(n)}$ is the largest. For the equal cut problem, define an approximate solution

$$v_{\sigma(i)} = \begin{cases} -1 & \text{if } i \leq \frac{n}{2} \\ 1 & \text{if } i > \frac{n}{2} \end{cases}$$

This procedure is an extension of the previous one. Although there are no quality bounds available, a partial result follows.

Procedure E: Let X^* be the solution to the semidefinite relaxation and let $X^* = V^T V$. Using a vector u from the unit ball, let $\widehat{v} = V^T u$. For $k \in [-1, 1]$, let

$$v_i = \begin{cases} -1 & \text{if } \widehat{v}_i \leq k \\ 1 & \text{if } \widehat{v}_i > k \end{cases}$$

Lemma 3.1. *Let $k \geq 0$. Using Procedure E,*

$$E\left[v_i v_j | k\right] = \begin{cases} \frac{2}{\pi}\arcsin\left(V_i^T V_j\right) & \textit{if} \quad \arccos\left(V_i^T V_j\right) \leq 2\arccos\left(k\right) \\ \frac{4}{\pi}\arcsin\left(k\right) - 1 & \textit{otherwise} \end{cases}$$

Proof. The event that $k \leq V_i^T u$ is equivalent to $\arccos\left(V_i^T u\right) \leq \arccos\left(k\right)$ since $\arccos\left(\cdot\right)$ is a monotone decreasing function. As illustrated in Figure 3, let $\Theta =$

$\arccos(k)$, $\theta_{ij} = \arccos\left(V_i^T V_j\right)$, and let $-\Theta \leq \theta \leq 2\pi - \Theta$ satisfy $\cos\theta = V_i^T u$. Two values for θ satisfy this constraint, allowing every vector on the unit sphere a representation. In Figure 3 ,there are seven vectors emanating from the origin. One represents the vector V_i, one represents V_j, and another represents a random vector u. The remaining four vectors identify the regions $[-\Theta, \Theta]$ and $[\theta_{ij} - \Theta, \theta_{ij} + \Theta]$. Whether the vector u falls in or out of these regions determines the sign of $\hat{v}_i$ and $\hat{v}_j$. One of two cases may occur. On the left, θ_{ij} is small enough for both $V_i^T u$ and $V_j^T u$

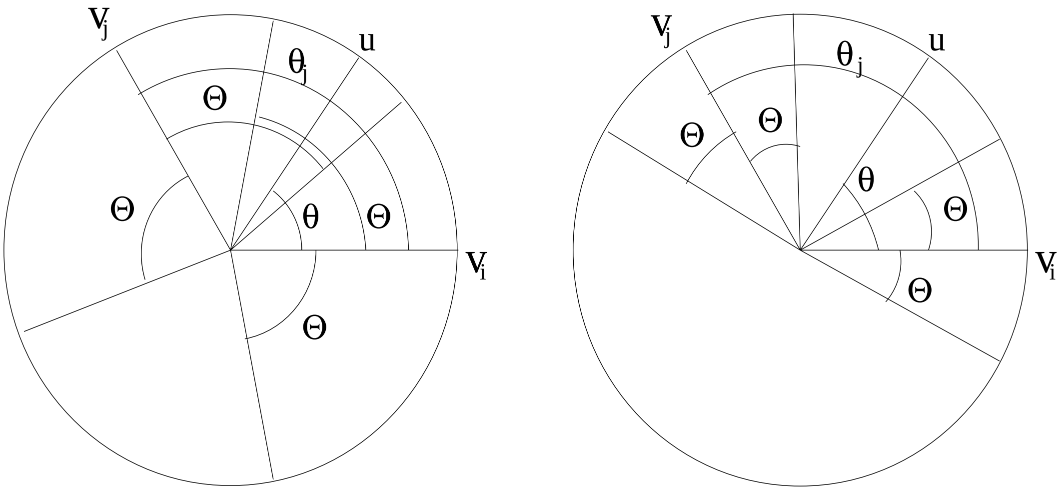

Figure 3: Random vectors from unit sphere

to be greater than k. In that case, the following table shows what value v_i and v_j would be assigned for each angle θ.

Since vectors are randomly chosen from the unit ball, angle θ is a random variable uniformly distributed over the interval $[-\Theta, 2\pi - \Theta]$. The chance that v_i

Case 1				Case 2		
Angle θ	v_i	v_j		Angle θ	v_i	v_j
$-\Theta \leq \theta < \theta_{ij} - \Theta$	1	-1		$-\Theta \leq \theta < \Theta$	1	-1
$\theta_{ij} - \Theta \leq \theta < \Theta$	1	1		$\Theta \leq \theta < \theta_j - \Theta$	-1	-1
$\Theta \leq \theta < \theta_{ij} + \Theta$	-1	1		$\theta_{ij} - \Theta \leq \theta < \theta_{ij} + \Theta$	-1	1
$\theta_{ij} + \Theta \leq \theta < 2\pi - \Theta$	-1	-1		$\theta_j + \Theta \leq \theta < 2\pi - \Theta$	-1	-1

Table 1: Assigning partitions

and v_j have the same sign is $\frac{2\theta_j}{2\pi}$ and given the first condition is true,

$$\begin{aligned} E\left[v_i v_j\right] &= 1\left(1 - \tfrac{1}{\pi}\arccos\left(v_i v_j\right)\right) + (-1)\tfrac{1}{\pi}\arccos\left(v_i v_j\right) \\ &= 1 - \tfrac{2}{\pi}\arccos\left(v_i v_j\right) \\ &= \tfrac{2}{\pi}\arcsin\left(V_i^T V_j\right). \end{aligned}$$

Since $k \geq 0$, the right side of Figure 3 shows that it it may not be possible for both values to be greater than k. In that event, the chance that v_i and v_j will have the same sign is $\frac{4\Theta}{2\pi}$ and

$$\begin{aligned} E\left[v_i v_j\right] &= 1\left(1 - \tfrac{2}{\pi}\arccos\left(k\right)\right) + (-1)\tfrac{2}{\pi}\arccos\left(k\right) \\ &= 1 - \tfrac{4}{\pi}\arccos\left(k\right) \\ &= \tfrac{4}{\pi}\arcsin\left(k\right) - 1 \\ &\leq \tfrac{2}{\pi}\arcsin\left(V_i^T V_j\right). \end{aligned}$$

. $\square$

Lemma 3.2. *Let $k < 0$. Using Procedure E,*

$$E\left[v_i v_j | k\right] = \begin{cases} \frac{2}{\pi}\arcsin\left(V_i^T V_j\right) & \textit{if} \quad \arccos\left(V_i^T V_j\right) \leq 2\arccos\left(-k\right) \\ \frac{4}{\pi}\arcsin\left(-k\right) - 1 & \textit{otherwise} \end{cases}$$

Proof. For any vector u, use $\widetilde{u} = -u$. This changes the signs of $\widehat{v}$ and k and the previous lemma can be applied. $\square$

When $k = 0$, the first case always applies and the expectation has a simple form. The maximum cut problem and box constrained quadratic problem always use $k = 0$.

For the equal problems, k is determined by $\widehat{v}_{median}$, which has an expectation of zero. When k is near zero, the first case usually applies. However vectors pointing in nearly opposite directions will be placed together more often than before. Experiments confirm that k is generally near 0 and the first case will usually apply. However, the instances when it does not, will decrease the expected objective value by small amount. Changing the value of k does not affect values that are very likely to be given the same sign. However if $\arccos\left(V_i^T V_j\right)$ is large, a positive k increases the chance they will be placed in the same set.

The quality of the expected answer depends on the quantity $E\left[v_i v_j\right]$. Note that

$$E\left[v_i v_j\right] = E_k\left[E\left[v_i v_j | k\right]\right]$$

A theoretical bound to the problem would follow from a distribution on the median, or at least an approximation of it.

For the unequal cut problem in which $|e^T v| = \kappa$, denote the median component $\hat{v}_{median}$. If $\hat{v}_{median} \leq 0$,

$$v_{\sigma(i)} = \begin{cases} -1 & \text{if } i \leq \frac{n-\kappa}{2} \\ 1 & \text{if } i > \frac{n-\kappa}{2} \end{cases}$$

Otherwise, when $\hat{v}_{median} > 0$,

$$v_{\sigma(i)} = \begin{cases} -1 & \text{if } i \leq \frac{n+\kappa}{2} \\ 1 & \text{if } i > \frac{n+\kappa}{2} \end{cases}$$

3.3 Maximum Cover Problems

Recall that the maximum cover problems require vectors that satisfy the constraints that $v \in \{-1, 1\}^{n+1}$ and $|v_0 + v_i + v_j| = 1$ for $(i, j) \in \mathrm{E}$. This is the same as saying these all three variables may not have the same sign. One feasible answer assigns v_0 and $v_i = -1, i = 1, \ldots, n$. This answer actually minimizes the objective function. To maximize it, generate a $\hat{v} \in \Re^{n+1}$ and initially let $v = sign(\hat{v})$. Without loss of generality, assume that $v_0 = 1$. A constraint is violated when $(i, j) \in \mathrm{E}$ and $v_i = v_j = 1$. To restore feasibility, select one the the two vertices and switching its sign. Applying this technique will not create any new constraint violations. Ye [8] has also shown that that these constraint will be satisfied naturally more than 91% of the time.

The choice of vertices v_i or v_j to switch may be arbitrary. Alternatively, the order statistics used for the equal and unequal cut problem suggest a better choice. If $|v_i - v_0| > |v_j - v_0|$, change the sign of v_i. Otherwise, change the sign of v_j.

3.4 Primal Solution as a Correlation Matrix

Consider when the random unit vector $u = V_i$. In this case, $\hat{v} = V^T V_i = X_i$. Each row and column of the primal solution matrix can be used to determine an approximate solution v to the combinatorial problem. For example, the primal solution to the maximum cut problem in Figure 1 is

$$X^* = \begin{pmatrix} 1.00 & 0.46 & -0.81 & -0.46 & -0.46 & 0.46 & 0.46 & 0.46 \\ 0.46 & 1.00 & -0.90 & -1.00 & -0.99 & 0.99 & 0.99 & 0.99 \\ -0.81 & -0.90 & 1.00 & 0.89 & 0.89 & -0.89 & -0.89 & -0.89 \\ -0.46 & -1.00 & 0.89 & 1.00 & 0.99 & -0.99 & -0.99 & -0.99 \\ -0.46 & -0.99 & 0.89 & 0.99 & 1.00 & -0.99 & -1.00 & -0.99 \\ 0.46 & 0.99 & -0.89 & -0.99 & -0.99 & 1.00 & 0.99 & 0.99 \\ 0.46 & 0.99 & -0.89 & -0.99 & -1.00 & 0.99 & 1.00 & 0.99 \\ 0.46 & 0.99 & -0.89 & -0.99 & -0.99 & 0.99 & 0.99 & 1.00 \end{pmatrix}$$

The primal solution X^* of the positive semidefinite program can also be interpreted as a correlation matrix of the variables in v. The matrix is symmetric with diagonal elements equal to one and off diagonal elements in the interval $[-1, 1]$. More parallels between X and a covariance matrix are discussed by Bertsimas and Ye [8].

Each column of X^* creates a partition. Column i assigns the variables with a positive correlation to v_i in one set and the variables with a negative correlation to v_i in the other set. In the case of this example, vertices C, D, and E get put in one set while vertices A, B, F, G, and H get put in the other set. This partition is, in fact, the optimal one. In other problems, the columns of X^* may offer different answers and the answers may not be optimal.

3.5 Structure Provided by the Dual Scaling Algorithm

Recall that the dual scaling algorithm can provide a primal matrix X when it is needed. By (2.28),

$$X = \frac{\bar{z}^k - b^T y^k}{\rho} (S^k)^{-1} \left(\mathcal{A}^T d(\bar{z}^k)_y + S^k \right) (S^k)^{-1}$$

Let $LL^T = \left(\mathcal{A}^T d(\bar{z}^k)_y + S^k \right)$ and choose

$$V = \sqrt{\frac{\bar{z}^k - b^T y^k}{\rho}} (S^k)^{-1} L \tag{3.66}$$

This choice utilizes sparsity even though X is dense. After generating a random vector u, multiply L and u and then solve the sparse linear system to solve for $\hat{v}$.

For the heuristic procedure,

$$X_i = \frac{\bar{z}^k - b^T y^k}{\rho} \left((S^k)^{-1} \left(\sum_{i=1}^{m} d(\bar{z}^k)_{y_i} a_i a_i^T \right) (S^k)^{-1} + (S^k)^{-1} \right) e_i \tag{3.67}$$

This choice of $\hat{v}$ saves time in cases like the equal cut problem when considerable sparsity exists, but a few constraints make the dual matrix dense. In these problems the matrix L is totally dense. The use of equation (3.67) requires solving a system of equations against two right hand sides, but the rank one structure of the constraints makes the matrix multiplications in the middle term very fast and preserves the advantages of sparsity.

3.6 Generating Random Vectors

The random vectors u in the randomized algorithm come from a uniform distribution over the unit sphere. There are many ways to theoretically generate these vectors. One well known method selects each coordinate from a Gaussian distribution and then normalizing the vector to length 1. The random number generator of most C compilers, however, uses a uniform distribution. Gaussian distributions can be approximated by a function of a uniformly distributed random variable, but I propose a better method.

Theorem 3.3. *Let $\theta_1, \dots, \theta_n$ be independent, identically distributed random variables from a uniform distribution over the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. Let $u_i = \tan(\theta_i)$, $i = 1, 2, \dots, n$, $u = (u_1, u_2, \dots, u_n)$, and normalize the vector $u := u/\|u\|$. Then, the random vector u has a uniform distribution over the unit sphere.*

Proof. Consider the hemisphere $\mathcal{H} = \{u \in \Re^{n+1} | \sum_0^n u_i^2 = r^2, u_0 > 0, r > 1\}$. The intersection of the hemisphere with the hyperplane $u_0 = 1, (0 < k < r)$ defines a sphere $\mathcal{S} = \{u \in \Re^n | \sum_{i=1}^n u_i^2 = r^2 - 1\}$, that is embedded in $\mathcal{H}$. If vectors on $\mathcal{H}$ have a uniform distribution, the vectors on any subset, like $\mathcal{S}$, also have a uniform distribution.

Define the hyperplane $\mathcal{P}_i = \{u \in \Re^{n+1} | u_j = k_j, j \neq 0, i\}$ where $i \neq 0$ and k_j's are constants. This hyperplane is parallel to the the u_0 and $u_{i \neq 0}$ axes and perpendicular to the other axes. The intersection of $\mathcal{P}_i$ and $\mathcal{H}$ is either empty, a single point, or a half circle (Figure 4). On the hyperplane, define the angle θ_i to be the angle between the vector on the circle and the u_0 axis. Given the uniform distribution of points on the hemisphere, the angle θ_i is a uniformly distributed random variable over the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. The coordinates u_0 and u_i are related by $u_i/u_0 = \tan(\theta_i)$ for any radius r. This relation is independent of the other coordinates.

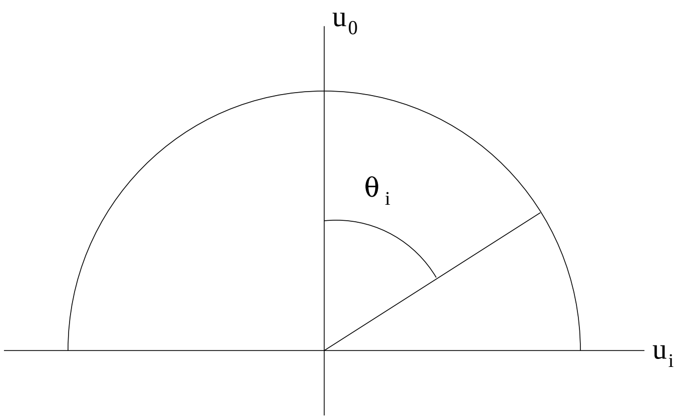

Figure 4: Random vector on hemisphere $\mathcal{H}$.

To generate a point on the unit sphere, select n independent random variables taken from a uniform distribution over the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. These values determine the ratios u_i/u_0. Fixing $u_0 = 1$ maps the points onto the circle $\mathcal{S}$. After normalizing it, the vector $(u_0, \ldots, u_n)$ is equally likely to be any vector on the hemisphere $\mathcal{H}$. The vector $(u_1, \ldots, u_n)$ can be renormalized to lie in the unit sphere. $\square$

CHAPTER IV
COMPUTATIONAL RESULTS

4.1 Implementation

I implemented the dual scaling algorithm in ANSI C and tested the computer code on large scale combinatorial optimization problems. This work was done with help from Xiong Zhang, who developed efficient routines for reordering matrices, factoring them, and solving systems of equations. His routines use sparse data structures and utilize the cache memory of computers to make the algorithm as fast as possible.

The code is designed to solve problems in with both linear and positive definite constraints in the form of (MP).

$$\begin{aligned} &\text{Minimize} && C \bullet X + c^T x \\ &\text{Subject to} && (a_i a_i^T) \bullet X + \hat{a}_i^T x = b_i, \quad i = 1, \dots, m. \\ & && X \succeq 0, \ x \geq 0. \end{aligned} \tag{4.68}$$

The user must supply a data file containing the objective matrix C, objective vector c, constraint vectors a_i and $\hat{a}_i$, and the vector b. The problem (MP) must have constraint matrices have a rank one structure $A_i = a_i a_i^T$ and an interior. The program solves the positive semidefinite program using the dual scaling algorithm.

This program is one of the first codes written in C to solve positive semidefinite programs. There are several codes available in Matlab, but they are generally unsuitable for large scale problems. This software is available for free in the public domain.

There are several options the user may select. The program reads a specification file that states which options should be used. For example, the user may choose to minimize, or maximize the primal objective function. If the user chooses to maximize it, the program formulates the problem as a minimization problem by multiplying the objective function by negative one.

The program offers the choice between a constant step size parameter $\alpha < 1$, or a dynamic step size. The dynamic step size will uses a backtracking method to choose the value of α that approximately minimizes the dual potential function. In problems where dense constraints get separate treatment from sparse one, the dual matrix is not completely factored. Evaluating the potential function is more difficult, so simple heuristics determine the dynamic step size. If one step size works for consecutive iterates, the program tries increasing it. On the other hand, if the selected value yields an infeasible point, the program reduces the step size and tries again. In the event that the step does not improve the objective value, the longer step size does not get used.

The program allows the user to specify a threshold to determine when to apply a separate treatment for dense constraints. By default, constraints with more than half nonzeros get classified as dense and receive the treatment discussed in Section 2.3.

The user can also specify the important parameter ρ. The user can provide constant greater than one called rho_{factor}. Then $\rho = rho_{factor} \times (n + p)$.

For the combinatorial problems discussed in this paper, the user may also request a rank one answer for the original combinatorial optimization problem. The program uses the methods discussed in Chapter 3. The randomized procedure gets repeated n times and the best answer returns to the user.

The current version also requires constraints that bound the diagonal of X. The program uses these constraints to make the initial dual matrix diagonally dominant so that Gerschgorin's Theorem [5]. ensures it is positive definite.

Finally, the user can customize the program to their own machine by specifying the cache memory on that machine.

4.2 Benchmark Problems: G-set

Most of the previous numerical tests solving positive definite problems [12], [22], [35], [38], [39], were conducted on smaller problem data sets where the dimension n was only a few hundred or less.

A study of several larger problems was conducted by Fujisawa, Fukuda, Kojima, and Nakata [14]. Using primal dual positive semidefinite algorithms, they tested the positive semidefinite relaxation of the maximum cut problem with dimension up to 1250.

Helmberg and Rendl [17] used a spectral bundle method to solve larger maximum cut problems. Their test problems were generated from a machine independent graph generator, **rudy**, created by G. Rinaldi. This set of problems becomes a standard test set for graph optimization. These maximum cut problems range in size from 800 to 3000. Many of these problems, like G1-G10, G22-G31, and G43-G47 have a randomly created structure. Problems G11-G13, G32-G34, and G48-G50 come from a 2 dimensional toroidal grid, while G14-G21, G35-G42, and G51-G54 represent planar type graphs. Graphs G11 - G13 and G32 - G34 have negatively weighted edges while the others have positively weighted edges.

Our tests [6], [7] were performed on a PC with 400 MHz, 256 MB RAM, and 256 K cache memory. We tested the program on maximum cut, s-t cut, equal cut, unequal cut, maximum cover, and box QP problems.

In all the problems except the maximum cover problems the parameters of the dual scaling algorithm were chosen as follows. It used values of α equal to $0.99, 1.5, 3,$

and 6. Initially, we set $\alpha = 3$. When the value of α was successful for three consecutive iterates, we tried the next larger value. If we stepped out of the feasible region, we tried the next smaller value of α.

In addition, we tried a dynamic selection of ρ. Initially, it was set to $5(n+p)$. After a couple of iterates, ρ was dynamically selected using the following criteria:

$$\rho = 1.4 \times (n+p) \times \sqrt{(rgap^{k-1}/rgap^{k})}$$

where $rgap^{k-1}$ and $rgap^{k}$ are the relative duality gaps at the previous and current iteration.

The initial point was determined by

$$y_i^0 = C_{ii} - \sum_{j \neq i} |C_{ij}| - 1, \quad i = 1, ..., n,$$

which by Gerschgorin's Theorem, guarantees $S^0 \succ 0$ [5]. We stopped the iteration process when the relative duality gap

$$rgap^k = \frac{\bar{z}^k - b^T y^k}{1 + |b^T y^k|} \leq 10^{-6}$$

Tables 2 and 3 show the performance of the computer code for solving the positive semidefinite relaxation of maximum cut problems with stopping tolerances of $rgap \leq 10^{-4}$ and $rgap \leq 10^{-6}$. The column headings $PObj$, $DObj$, and $Rgap$, are the primal objective value, dual objective value, and relative duality gap at termination. The dimension and sparsity of the objective matrix is also shown. The tables list the number of iterations and the time, in seconds, needed to solve the problems.

Sparse problem with dimension $n = 800$ were solved using as few as 64 seconds. More dense problems of the same size used about 12 minutes. Problems with $n = 2000$ were solved in fewer than 9,400 seconds. The problems with $n = 3000$ were extremely sparse, and could be solved in less than 2900 seconds.

Fujisawa, Fukuda, Kojima, and Nakata [14]. performed their calculations on a DEC AlphaServer 8400 with a processing speed of 437 MHz and 8GB memory.

On sparse maximum cut problems with dimension of 1000, they used about 63,130 seconds; problems of dimension 1250 used 111,615 seconds.

Another positive semidefinite programming solver, CSDP [9] was released by Borchers. His solver also uses the HKM primal dual algorithm and tries to take advantage of sparsity. I tested it on the positive semidefinite relaxation of several G-set maximum cut problems with dimension of 800 and found that it required over ten minutes per iteration. In fairness, both CSDP, and the implementation of Fujisawa, et. al., work with arbitrary symmetric constraint matrices instead of the rank one matrices used in our implementation. Nonetheless, the dual scaling algorithm appears significantly faster than primal dual methods.

The computations of Helmberg and Rendl were performed on a UltraSPARC station with 64 MB memory. Their spectral bundle method was performed very well on the large problems. On problems with a randomly created structure, the bundle method appears faster. In these problems, the Cholesky factor of the slack matrix is relatively dense, despite the sparsity of the objective matrix. For the toroidal and planar graphs, the dual matrix has a much better structure. In these problems, a minimum degree ordering kept the Cholesky factor sparse and the back and forward substitutions quick. Figure 8 shows how sparsely structured is the Cholesky factor of problem G14. In problems with a more structured objective matrix, the dual SDP algorithm outperformed the bundle method. One advantage of the spectral bundle method is that it uses considerably less memory since it does not create or store a matrix as large as M. It is not a polynomial time algorithm, however. It also cannot solve the primal problem which is needed guarantee or verify optimality at termination. Furthermore, because no primal solution is available, it also cannot generate rank one answers to the initial combinatorial problems.

Our implementation of the dual scaling algorithm appears to be the first to converge to an optimal point in polynomial time, verify the optimality by solving the primal and dual problems, and use the characteristics inherent in many large scale problems to its advantage. Its success with relatively dense examples shows that the algorithm is generally very efficient, while the improved performance on more sparse examples shows how it exploits the structure of most large scale problems.

4.3 Large Scale Combinatorial Optimization Problems

Many large scale problems contain a sparse structure. An efficient algorithm for solving these problems must take advantage of the sparsity. The dual scaling algorithm does take advantage of this structure. Table 4 shows the time required by the three most expensive steps of the dual scaling algorithm. For six different maximum cut problems, the table shows the seconds required for one factorization of matrix S, one computation of the matrix M, and one factorization of the matrix M.

Since the sparsity of the Cholesky factor of S affects the time to factor S and create M, the sparsity statistic in the second column gives the percentage of nonzero entries in the factor.

Table 4 shows how the dual algorithm exploits sparsity. When factor of the dual matrix, S_{chol}, is sparse, the factorization of S and the creation of M is very fast and the factorization of matrix M dominates the computation time. Since M is generally dense, regardless of the sparsity of S, its computation time is constant for problems of equal size. For dense problems, the creation of M dominates the computation time: maximum cut problem it use up to $3n^3$ floating point operations, while the Cholesky factorization uses a sixth of that amount.

Table 5 demonstrates the savings achieved by treating dense constraints in a separate manner from the sparse constraints. Recall that the maximum cut problem

						$\text{rgap}=10^{-4}$		$\text{rgap}=10^{-6}$	
Name	Dim	Spars	Pobj	Dobj	Rgap	Iter	Time	Iter	Time
G1	800	6.12%	-4.833276e+04	-4.833279e+04	5.452359e-07	20	616.08	24	741.15
G2	800	6.12%	-4.835767e+04	-4.835772e+04	9.502824e-07	19	592.69	23	719.25
G3	800	6.12%	-4.833732e+04	-4.833733e+04	3.401351e-07	19	589.56	24	746.54
G4	800	6.12%	-4.844576e+04	-4.844581e+04	9.824817e-07	19	595.43	23	723.28
G5	800	6.12%	-4.839953e+04	-4.839955e+04	3.552550e-07	19	594.71	24	752.24
G6	800	6.12%	-1.062463e+04	-1.062464e+04	8.541375e-07	21	646.12	25	769.41
G7	800	6.12%	-9.957042e+03	-9.957051e+03	8.420776e-07	21	654.40	25	779.28
G8	800	6.12%	-1.002773e+04	-1.002773e+04	9.678249e-07	21	655.63	24	780.46
G9	800	6.12%	-1.011492e+04	-1.011493e+04	7.775437e-07	21	654.09	25	778.95
G10	800	6.12%	-9.940246e+03	-9.940253e+03	7.465415e-07	20	620.63	24	742.82
G11	800	0.63%	-2.516658e+03	-2.516659e+03	3.719002e-07	18	64.40	23	82.05
G12	800	0.63%	-2.495497e+03	-2.495498e+03	5.198500e-07	19	71.03	24	89.50
G13	800	0.63%	-2.588544e+03	-2.588546e+03	9.570427e-07	20	76.70	24	91.88
G14	800	1.59%	-1.276625e+04	-1.276627e+04	9.986387e-07	29	166.43	33	189.11
G15	800	1.58%	-1.268623e+04	-1.268623e+04	3.450954e-07	33	188.68	39	222.87
G16	800	1.59%	-1.270007e+04	-1.270007e+04	4.674935e-07	27	154.31	31	177.01
G17	800	1.58%	-1.268530e+04	-1.268531e+04	4.548353e-07	26	149.60	30	172.44
G18	800	1.59%	-4.664038e+03	-4.664040e+03	4.708559e-07	47	276.82	51	299.92
G19	800	1.58%	-4.328040e+03	-4.328042e+03	3.345754e-07	33	193.66	38	221.98
G20	800	1.59%	-4.445567e+03	-4.445570e+03	7.261350e-07	33	193.80	37	216.54
G21	800	1.58%	-4.417133e+03	-4.417136e+03	8.094406e-07	38	223.30	42	246.10
G22	2000	1.05%	-5.654376e+04	-5.654378e+04	3.957778e-07	23	8215.71	28	9997.62
G23	2000	1.05%	-5.658202e+04	-5.658204e+04	3.910436e-07	23	8146.30	28	9920.25
G24	2000	1.05%	-5.656340e+04	-5.656342e+04	4.981024e-07	23	8323.22	27	9759.06
G25	2000	1.05%	-5.657696e+04	-5.657698e+04	3.876964e-07	23	8306.01	28	10106.27
G26	2000	1.05%	-5.653145e+04	-5.653148e+04	5.027876e-07	23	8323.72	27	9756.52
G27	2000	1.05%	-1.656663e+04	-1.656664e+04	6.729729e-07	25	8851.74	29	10268.68

Table 2: Benchmark problems: solving the positive semidefinite relaxation of the maximum cut G-set problems- Part 1.

Name	Dim	Spars	Pobj	Dobj	Rgap	rgap= 10^{-4} Iter	Time	rgap= 10^{-6} Iter	Time
G28	2000	1.05%	-1.640315e+04	-1.640316e+04	7.335225e-07	25	8862.04	29	10278.55
G29	2000	1.05%	-1.683555e+04	-1.683556e+04	6.531697e-07	25	9034.53	29	10483.11
G30	2000	1.05%	-1.686152e+04	-1.686153e+04	6.413532e-07	26	9386.62	30	10843.05
G31	2000	1.05%	-1.646672e+04	-1.646673e+04	6.898466e-07	25	9047.75	29	10493.99
G32	2000	0.25%	-6.270553e+03	-6.270559e+03	9.633737e-07	23	1070.39	27	1255.70
G33	2000	0.25%	-6.177246e+03	-6.177250e+03	6.333926e-07	25	1175.24	29	1362.61
G34	2000	0.25%	-6.186747e+03	-6.186750e+03	4.510949e-07	24	1182.73	28	1381.23
G35	2000	0.64%	-3.205895e+04	-3.205896e+04	3.752328e-07	46	5167.17	51	5716.93
G36	2000	0.64%	-3.202383e+04	-3.202386e+04	9.011525e-07	38	4381.39	42	4841.52
G37	2000	0.64%	-3.207448e+04	-3.207449e+04	3.820329e-07	42	4836.22	47	5400.13
G38	2000	0.64%	-3.205987e+04	-3.205990e+04	8.761401e-07	47	5392.57	52	5952.48
G39	2000	0.64%	-1.151058e+04	-1.151059e+04	7.376637e-07	59	6615.01	63	7056.50
G40	2000	0.64%	-1.145915e+04	-1.145916e+04	3.728182e-07	52	6123.85	57	6703.57
G41	2000	0.64%	-1.146087e+04	-1.146087e+04	4.158517e-07	58	6629.09	63	7194.15
G42	2000	0.64%	-1.178500e+04	-1.178501e+04	7.308258e-07	45	5049.62	49	5495.55
G43	1000	2.10%	-2.812887e+04	-2.812889e+04	7.416858e-07	18	767.17	22	939.50
G44	1000	2.10%	-2.811152e+04	-2.811154e+04	7.511560e-07	18	770.22	22	939.07
G45	1000	2.10%	-2.809911e+04	-2.809913e+04	4.530243e-07	21	900.51	25	1075.49
G46	1000	2.10%	-2.811972e+04	-2.811973e+04	3.978937e-07	18	782.46	23	999.66
G47	1000	2.10%	-2.814662e+04	-2.814664e+04	7.609519e-07	18	751.57	22	920.39
G48	3000	0.17%	-2.399999e+04	-2.400000e+04	3.699426e-07	14	2878.85	19	3861.30
G49	3000	0.17%	-2.399999e+04	-2.400000e+04	3.718535e-07	14	2873.69	19	3826.62
G50	3000	0.17%	-2.395268e+04	-2.395269e+04	3.543263e-07	14	2389.73	19	3192.86
G51	1000	1.28%	-1.602501e+04	-1.602502e+04	7.759416e-07	29	341.45	33	388.06
G52	1000	1.28%	-1.603854e+04	-1.603856e+04	7.427919e-07	36	441.94	41	489.28
G53	1000	1.28%	-1.603886e+04	-1.603887e+04	8.122974e-07	26	306.85	30	353.86
G54	1000	1.28%	-1.602476e+04	-1.602478e+04	7.421491e-07	29	340.43	33	387.51

Table 3: Benchmark problems: solving the positive semidefinite relaxation of the maximum cut G-set problems- Part 2.

Graph	Dim n	Sparsity of S_{chol}	Factor S (sec.)	Compute M (sec.)	Factor M (sec.)
G1	800	73.6%	1.412	12.856	1.983
G11	800	4.2%	0.010	1.272	1.863
G14	800	14.3%	0.140	3.105	1.863
G22	2000	47.8%	11.076	129.917	32.046
G32	2000	1.6%	0.030	9.864	30.744
G35	2000	11.8%	1.352	41.113	30.764

Table 4: Time required for steps of the maximum cut problem.

has no dense constraints. Column 3 of Table 5 shows the seconds used in each iterate of the maximum cut problem. The equal cut problem adds one dense constraint to the maximum cut problem. The fourth column shows the seconds per iterate used for the equal cut problem when the dense constraint is treated separately. The fifth column, under the heading "Equal Cut 2", shows the seconds per iterate solving the equal cut problem when the dense constraint is treated the same as the other constraints. The data shows that the special treatment for dense constraints improves maintains the advantages offered by sparsity.

Graph	dim n	Sparsity of S_{chol}	Maximum Cut (sec./iterate)	Equal Cut 1. (sec./iterate)	Equal Cut 2 (sec./iterate)
G1	800	73.6%	34.10	34.21	45.79
G11	800	4.2%	3.52	3.76	45.71
G14	800	14.3%	6.34	6.36	45.61

Table 5: Savings achieved by separating dense constraints from sparse ones.

Tables 4 and 5 also emphasize the importance of a good ordering of the matrix in the beginning of the algorithm. The reordering of matrices has been studied for years [11], but to illustrate its importance, we include a few figures. The following figures show the structure of S and its Cholesky factor in the 800×800 example G14.

The objective matrix G14 has about 1.58% nonzero entries. Figure 5 shows the sparsity structure of the matrix before the minimum ordering and Figure 7 presents the structure after the minimum ordering. The dual solution S has the same sparse structures. Figure 6 and Figure 8 show the sparsity patterns of the Cholesky factors. The Cholesky factor of the unordered matrix is very dense. Comparing the running times of G14 with G1, which has a much denser factorization, shows that the reordering matrix would reduce the running time by approximately a factor of four.

I also implemented the rank reduction techniques in Chapter 3. These techniques find approximate solutions to the combinatorial problems. The program also found vector solutions for these problems using the randomized techniques and the heuristic that uses the rows of the primal solution matrix. Tables 7,8,9,10,and 11 show the results for maximum cut, equal cut, unequal cut, s-t cut, and box quadratic problems. The rank reduction procedures repeated n times using random vectors, and once for each of the n columns of the primal matrix, used the time of approximately one or two iterates of the dual scaling algorithm.

The solutions generated for the combinatorial problems were of a very high quality. In cases where the edge weights are positive, the solutions had an objective value of 91 - 100% of the bound generated by the dual problem. In graphs with negatively weighted edges, like G11 - G13 and G32 - G34, the bound generated by the semidefinite program is not as tight as when only positive weights are present. In these cases, the the best solution was better than 83% of the bound.

The data also shows how the dual scaling algorithm may provide a sparse structure to the primal matrix X. In problems with all sparse constraints, the time required by the rank reduction was proportional to the sparsity of the the factor of S. Also, the randomized technique was fast than heuristic that uses the rows of X. In the equal cut problem, where a dense constraint is present, triangular matrix L in (3.66) is dense. In these problems, it is faster to use the rows of X because that technique can take advantage of sparsity. Table 6 demonstrates these results.

Graph	n	Sparsity of S_{chol}	Maximum Cut			Equal Cut		
			SDPtime	Htime	Rtime	SDPtime	Htime	Rtime
G1	800	73.6%	863	53	50	757	51	57
G11	800	4.2%	60	2.5	3.0	107	6	30
G14	800	14.3%	195	7	11	364	11	33
G22	2000	47.8%	8994	590	521	10515	618	704
G32	2000	1.6%	966	22	30	2194	50	419
G35	2000	11.8%	4304	147	136	6095	176	483

Table 6: Selected rank reduction times (seconds)

Name	Dim	Spar	SDPobj	Hobj	Robj	Ratio	Iter	SDPtime	Htime	Rtime	Ttime
G1	800	6.12%	12083.2	11421	11410	0.945	20	684	53	50	789
G2	800	6.12%	12089.4	11398	11396	0.943	19	656	53	50	762
G3	800	6.12%	12084.3	11425	11427	0.946	19	663	53	51	770
G11	800	0.63%	629.2	532	530	0.845	18	60	3	3	66
G12	800	0.63%	624.0	528	528	0.846	19	68	3	3	74
G13	800	0.63%	647.1	550	554	0.856	20	74	3	4	81
G14	800	1.59%	3191.6	2978	2978	0.933	29	195	7	11	214
G15	800	1.58%	3171.6	2967	2975	0.938	33	218	7	10	236
G16	800	1.59%	3175.0	2965	2971	0.936	27	175	7	11	194
G22	2000	1.05%	14136.0	12993	12975	0.919	23	8994	590	522	10121
G23	2000	1.05%	14145.5	12973	12964	0.917	23	8975	587	524	10102
G24	2000	1.05%	14140.9	12960	12955	0.917	23	9146	597	529	10288
G32	2000	0.25%	1567.6	1312	1304	0.837	23	966	22	30	1019
G33	2000	0.25%	1544.3	1274	1284	0.831	25	1066	23	31	1121
G34	2000	0.25%	1546.7	1286	1292	0.835	24	1095	28	36	1160
G35	2000	0.64%	8014.7	7447	7438	0.929	36	4304	147	136	4602
G36	2000	0.64%	8006.0	7427	7431	0.928	38	4453	158	146	4759
G37	2000	0.64%	8018.6	7455	7472	0.932	40	4705	151	140	5014
G43	1000	2.10%	7032.2	6534	6524	0.929	18	835	71	66	974
G44	1000	2.10%	7027.9	6498	6486	0.925	18	853	71	67	993
G45	1000	2.10%	7024.8	6484	6496	0.925	21	984	71	67	1125
G48	3000	0.17%	6000.0	6000	6000	1.000	14	2514	83	87	2667
G49	3000	0.17%	6000.0	6000	6000	1.000	14	2297	77	82	2462
G50	3000	0.17%	5988.2	5880	5880	0.982	14	2191	68	73	2355
G51	1000	1.28%	4006.3	3731	3734	0.932	29	386	16	20	422
G52	1000	1.28%	4009.6	3740	3742	0.933	37	494	16	20	531
G53	1000	1.28%	4009.7	3749	3743	0.935	26	337	16	20	373

Table 7: Solving the maximum cut G-set problems.

Name	Dim	Spar	SDPobj	Hobj	Robj	Ratio	Iter	SDPtime	Htime	Rtime	Ttime
G1	800	6.12%	12083.0	11425	11425	0.946	21	757	57	57	874
G2	800	6.12%	12089.3	11422	11423	0.945	21	772	58	57	890
G3	800	6.12%	12083.8	11426	11409	0.946	22	818	57	57	935
G11	800	0.63%	629.2	530	528	0.842	21	107	6	30	146
G12	800	0.63%	623.8	526	532	0.853	23	129	7	30	168
G13	800	0.63%	647.1	552	552	0.853	23	139	7	30	179
G14	800	1.59%	3189.9	2986	2977	0.936	33	317	11	33	364
G15	800	1.58%	3171.1	2973	2972	0.938	36	352	11	33	398
G16	800	1.59%	3174.6	2965	2967	0.935	29	262	11	33	310
G22	2000	1.05%	14135.8	12990	12959	0.919	25	10515	618	704	11879
G23	2000	1.05%	14145.4	12968	12995	0.919	24	10005	620	702	11369
G24	2000	1.05%	14140.4	12970	12946	0.917	25	10806	626	707	12180
G32	2000	0.25%	1567.6	1310	1310	0.836	27	1683	51	420	2194
G33	2000	0.25%	1544.3	1286	1288	0.834	28	1778	52	419	2290
G34	2000	0.25%	1546.7	1284	1288	0.833	28	1958	57	422	2477
G35	2000	0.64%	8014.1	7459	7440	0.931	35	5395	176	483	6095
G36	2000	0.64%	8005.0	7419	7423	0.927	40	6679	185	487	7391
G37	2000	0.64%	8017.4	7443	7435	0.928	39	5892	183	489	6604
G43	1000	2.10%	7032.1	6533	6480	0.929	20	1008	77	91	1181
G44	1000	2.10%	7027.7	6501	6483	0.925	23	1184	77	91	1357
G45	1000	2.10%	7024.6	6479	6456	0.922	20	1014	78	91	1188
G48	3000	0.17%	6000.0	6000	6000	1.000	15	3997	151	1375	5696
G49	3000	0.17%	6000.0	6000	6000	1.000	15	3531	144	1373	5245
G50	3000	0.17%	5988.2	5880	5880	0.982	15	2478	132	1367	4131
G51	1000	1.28%	4005.6	3732	3737	0.933	29	549	22	64	640
G52	1000	1.28%	4009.1	3741	3732	0.933	31	582	23	64	673
G53	1000	1.28%	4009.2	3745	3737	0.934	29	529	22	63	619

Table 8: Solving the equal cut G-set problems.

Name	Dim	Spar	SDPobj	Hobj	Robj	Ratio	It	SDPtime	Htime	Rtime	Ttime
G1	800	6.12%	11928.0	11188	11204	0.939	25	856	57	59	975
G2	800	6.12%	11928.5	11183	11196	0.939	25	861	57	59	981
G3	800	6.12%	11932.3	11176	11187	0.938	26	897	57	59	1016
G11	800	0.63%	628.6	502	504	0.802	31	104	6	30	143
G12	800	0.63%	622.8	502	500	0.806	27	97	7	30	136
G13	800	0.63%	646.3	522	522	0.808	31	116	7	30	156
G14	800	1.59%	3188.0	2935	2927	0.921	39	261	11	34	308
G15	800	1.58%	3167.3	2916	2923	0.923	42	277	11	34	324
G16	800	1.59%	3170.7	2911	2927	0.923	41	269	11	34	316
G22	2000	1.05%	13971.9	12710	12721	0.911	33	12954	618	743	14356
G23	2000	1.05%	13978.9	12703	12696	0.909	28	10890	614	739	12283
G24	2000	1.05%	13979.5	12673	12698	0.908	35	13851	624	745	15260
G32	2000	0.25%	1565.7	1224	1230	0.786	46	1940	49	419	2409
G33	2000	0.25%	1542.2	1194	1214	0.787	49	2100	51	420	2573
G34	2000	0.25%	1544.6	1192	1222	0.791	41	1867	55	421	2345
G35	2000	0.64%	8001.2	7331	7313	0.916	51	5662	185	487	6362
G36	2000	0.64%	7990.3	7297	7308	0.915	71	8351	184	492	9064
G37	2000	0.64%	8006.9	7310	7345	0.917	62	6974	182	494	7601
G43	1000	2.10%	6949.4	6390	6379	0.920	24	1114	77	94	1290
G44	1000	2.10%	6946.9	6360	6377	0.918	27	1261	77	95	1438
G45	1000	2.10%	6942.9	6349	6350	0.915	26	1218	78	94	1396
G48	3000	0.17%	5906.3	5252	5252	0.889	21	4643	147	1374	6332
G49	3000	0.17%	5906.3	5252	5252	0.889	21	3424	142	1381	5113
G50	3000	0.17%	5894.6	5252	5252	0.891	21	3316	128	1361	4959
G51	1000	1.28%	3999.2	3663	3686	0.922	35	463	22	64	554
G52	1000	1.28%	4003.3	3683	3685	0.921	32	429	22	64	520
G53	1000	1.28%	4002.1	3664	3688	0.922	34	441	22	64	532

Table 9: Solving the unequal cut G-set problems ($\kappa = n/8$).

The code was also tested on problems with an even higher dimension. Table 12 shows the times needed to solve the positive semidefinite relaxation of problems with 5000 and 7000 vertices. The stopping tolerance for these problems was 10^{-3}.

Name	Dim	Spar	SDPobj	Hobj	Robj	Ratio	Iter	SDPtime	Htime	Rtime	Ttime
G1	800	6.12%	12082.1	11316	11419	0.945	21	719	53	47	822
G2	800	6.12%	12087.7	11325	11419	0.945	21	725	53	47	828
G3	800	6.12%	12082.1	11340	11452	0.948	21	740	54	48	845
G11	800	0.63%	629.2	498	530	0.842	23	75	3	3	81
G12	800	0.63%	623.9	506	524	0.840	24	83	3	3	89
G13	800	0.63%	647.1	512	554	0.856	20	73	3	3	80
G14	800	1.59%	3191.1	2925	2982	0.935	25	164	7	10	182
G15	800	1.58%	3171.3	2895	2979	0.939	39	255	7	10	272
G16	800	1.59%	3174.8	2919	2969	0.935	31	207	7	10	225
G22	2000	1.05%	14134.7	12817	12995	0.919	25	11762	618	741	13243
G23	2000	1.05%	14141.3	12822	12964	0.917	25	12022	623	742	13534
G24	2000	1.05%	14138.9	12860	12983	0.918	23	9972	612	731	11492
G32	2000	0.25%	1567.6	1268	1302	0.831	24	1474	24	39	1547
G33	2000	0.25%	1544.3	1226	1282	0.830	25	1565	26	41	1643
G34	2000	0.25%	1546.6	1232	1284	0.830	24	1576	42	47	1665
G35	2000	0.64%	8014.4	7323	7441	0.929	36	5441	209	183	5834
G36	2000	0.64%	8005.7	7371	7430	0.928	49	7788	222	195	8207
G37	2000	0.64%	8018.2	7351	7457	0.930	42	6463	214	188	6868
G43	1000	2.10%	7030.2	6421	6502	0.925	21	976	71	66	1117
G44	1000	2.10%	7026.9	6427	6501	0.925	21	988	71	67	1128
G45	1000	2.10%	7024.2	6423	6483	0.923	21	997	71	67	1139
G48	3000	0.17%	6000.0	6000	6000	1.000	17	3208	78	84	3388
G49	3000	0.17%	6000.0	6000	6000	1.000	22	4225	73	79	4394
G50	3000	0.17%	5987.6	5860	5866	0.980	20	3158	65	73	3309
G51	1000	1.28%	4006.0	3700	3746	0.935	29	378	16	20	414
G52	1000	1.28%	4009.5	3669	3734	0.931	33	437	16	20	473
G53	1000	1.28%	4009.3	3703	3743	0.934	26	336	16	20	372

Table 10: Solving the $s-t$-cut G-set problems ($s=[n/3],\ t=2s$).

Name	Dim	Spar	SDPobj	Hobj	Robj	Ratio	Iter	SDPtime	Htime	Rtime	Ttime
G1	800	6.12%	48332.8	45681.9	45693.9	0.945	28	954	53	50	1059
G2	800	6.12%	48357.7	45589.9	45601.9	0.943	30	1035	54	50	1141
G3	800	6.12%	48337.3	45697.9	45697.9	0.945	29	1013	53	51	1119
G11	800	0.63%	2539.3	2156.4	2151.0	0.849	36	120	2	3	126
G12	800	0.63%	2513.6	2162.0	2146.4	0.860	35	124	3	3	131
G13	800	0.63%	2606.2	2261.1	2268.7	0.871	35	130	3	4	137
G14	800	1.59%	12766.3	11911.5	11931.5	0.935	30	202	7	11	222
G15	800	1.58%	12686.2	11867.6	11939.6	0.941	29	192	7	11	210
G16	800	1.59%	12700.1	11859.5	11911.5	0.938	31	202	7	11	220
G22	2000	1.05%	56543.8	51969.7	51849.7	0.919	33	12845	592	526	13978
G23	2000	1.05%	56582.1	51889.6	51961.6	0.918	33	12821	587	520	13944
G24	2000	1.05%	56563.4	51837.7	51921.7	0.918	40	15821	596	529	16963
G32	2000	0.25%	6316.9	5346.0	5381.0	0.852	40	1688	22	30	1740
G33	2000	0.25%	6232.0	5265.7	5240.8	0.845	43	1832	23	31	1887
G34	2000	0.25%	6241.1	5277.0	5280.3	0.846	49	2237	29	36	2302
G35	2000	0.64%	32059.0	29787.0	29807.0	0.930	35	3886	148	136	4172
G36	2000	0.64%	3202.0	29706.6	29650.6	0.928	39	4560	156	145	4863
G37	2000	0.64%	32074.5	29822.5	29810.5	0.930	44	5012	151	140	5306
G43	1000	2.10%	28128.9	26134.8	26070.8	0.929	35	1623	71	66	1762
G44	1000	2.10%	28111.6	25990.7	25942.7	0.925	54	2535	71	67	2676
G45	1000	2.10%	28099.1	25934.9	25982.9	0.925	33	1543	71	66	1683
G48	3000	0.17%	24000.0	23999.0	23999.0	1.000	18	3331	83	88	3511
G49	3000	0.17%	24000.0	23999.1	23999.1	1.000	18	3095	78	82	3273
G50	3000	0.17%	23952.7	23518.0	23518.9	0.982	17	2892	67	74	3037
G51	1000	1.28%	16025.0	14923.0	14947.4	0.933	43	563	16	20	599
G52	1000	1.28%	16038.6	14959.4	14975.4	0.934	35	467	16	20	504
G53	1000	1.28%	16038.9	14995.3	14995.3	0.935	34	441	16	20	477

Table 11: Solving the box-qp G-set problems.

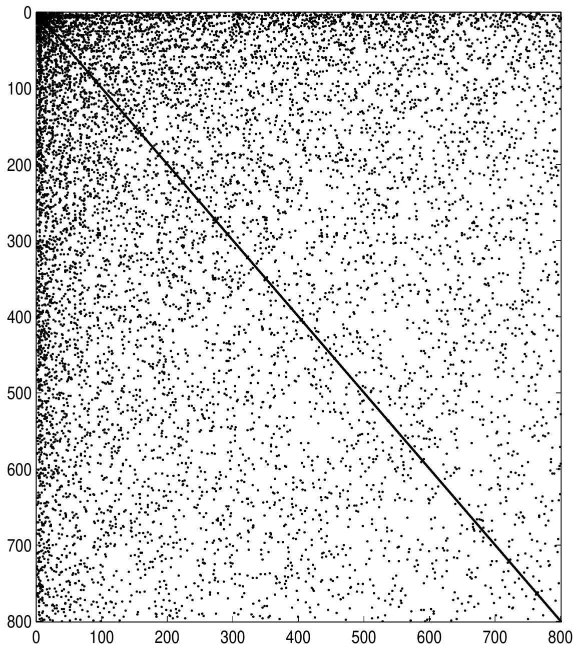

Figure 5: Sparsity pattern in G14

Name	Dim	Sparsity	Iter	SDP time(sec.)
G55	5000	0.12%	46	87,664
G56	5000	0.06%	38	27,904
G60	7000	0.08%	51	266,280
G62	7000	0.04%	28	90,238

Table 12: Maximum cut problems with $n \geq 5000$

I also solved several sets of maximum cover problems. The results are summarized in Table 13. Despite the sparsity of the dual matrix, these problems behave much more like dense problems because the dimension m greatly exceeds the dimension n.

Name	n	m	Spar	SDPobj	Iter	SDPtime
G11	800	2401	0.63%	7.39e-03	86	6480
G12	800	2401	0.63%	2.83e-03	85	6160
G13	800	2401	0.63%	3.19e+00	87	6515
G51	1000	6910	1.28%	3.02e+02	104	265,511

Table 13: Solving maximum cover problems

4.4 Precision

In most of the combinatorial experiments, the positive semidefinite program terminated once it found an approximate solution with a relative duality gap of less than 10^{-4}. Although this precision is less than the precision sought in other applications, it seems more than sufficient for combinatorial problems.

Many combinatorial applications require only a bound for the problem. Tables 7 - 11 show that the best solution to the combinatorial problem is only $85\% - 95\%$ of the bound generated by the positive semidefinite relaxation. The bounds offer only a few significant digits of accuracy, so higher precision answers mean very little.

Generating rank one answers does not require a high precision answer either. Recall that the original problem (1.1) asks only for binary decisions: does vertex i belong to one set or the other? In most of the rank reduction techniques discussed earlier, the sign of a scaler $\hat{v}_i$ determines the set in which a vertex belongs. To confirm that low precision is sufficient, I tested combinatorial problems by applying the rank reduction procedures each time the dual scaling algorithm provided an new primal matrix. Table 14 shows the results for solving G14 as a maximum cut problem. For each primal matrix the algorithm generated, the table shows the iterate of the dual scaling algorithm that generated it, the primal objective value, dual objective value, and relative duality gap at that iterate, and the best rank one answers found to the maximum cut problem. Each column of X was tried, and the best solution appears under "HObj". In addition, I used n applications of the randomized algorithms and listed the best under "RObj".

Iterate	PObj	DObj	Rgap	HObj	RObj
1	2330.305	5260.521	5.6e-01	2660	2670
3	2540.012	4659.185	4.5e-01	2683	2701
14	2943.495	3220.023	8.6e-02	2952	2953
19	3071.274	3194.135	3.8e-02	2973	2981
22	3171.967	3191.787	6.2e-03	2979	2988
23	3181.604	3191.650	3.1e-03	2974	2988
25	3189.057	3191.583	7.9e-04	2978	2976
26	3190.311	3191.576	4.0e-04	2978	2993
28	3191.251	3191.568	9.9e-05	2978	2976
29	3191.409	3191.568	5.0e-05	2978	2976
32	3191.547	3191.567	6.2e-06	2978	2980
33	3191.557	3191.567	3.1e-06	2978	2978
35	3191.564	3191.567	7.8e-07	2978	2987
36	3191.566	3191.567	3.9e-07	2978	2978
38	3191.566	3191.567	9.7e-08	2978	2979
39	3191.567	3191.567	4.9e-08	2978	2986
42	3191.567	3191.567	6.1e-09	2978	2985

Table 14: Rank one solutions of the G14 maximum cut problem.

This example was very typical of the other examples. Early iterates improved the quality of the cuts significantly. However, the quality of the answers improved very little after achieving a relative gap of 10^{-3}. In fact, the best cut I found was calculated in iterate 26 using the randomized algorithm. The best cut produced by the columns of X was found in iterate 22, with a relative duality gap of $6.2e-03$. Hence, stopping the scaling algorithm upon reaching a relative duality gap of 10^{-4} seems sufficient.

At this level of precion, the superlinear convergence that characterizes some primal-dual algorithms just begins to exhibit itself. Hence, the iteration count for the dual algorithm may be competitive with primal dual algorithms. I have shown that iterations of the dual scaling algorithm cost less than primal-dual iterations and exploit structure and sparsity in the data. The dual scaling algorithm is a theoretically sound and practically efficient method for solving large scale combinatorial optimization problems.

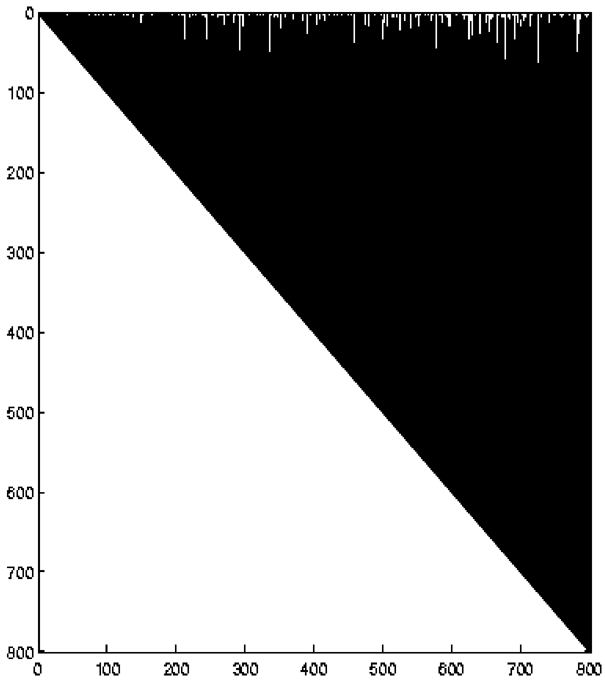

Figure 6: Sparsity pattern of Cholesky factor in G14

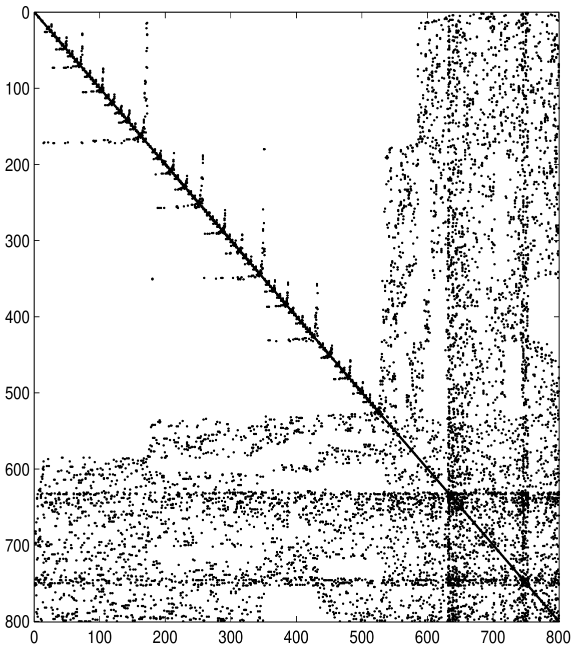

Figure 7: Sparsity pattern of ordered G14

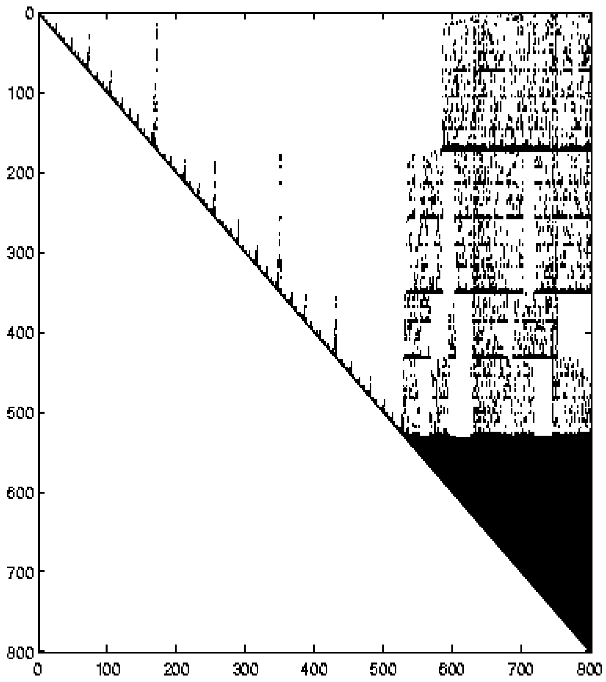

Figure 8: Sparsity pattern of Cholesky factor of ordered G14

REFERENCES

[1] F. Alizadeh. *Combinatorial optimization with interior point methods and semidefinite matrices.* PhD thesis, University of Minnesota, Minneapolis, MN, 1991.

[2] F. Alizadeh, J. P. A. Haeberly, M. V. Nayakkankuppam, M. L. Overton, and S. Schmieta. Sdppack user's guide – version 0.9 beta. Technical Report TR1997-737, Courant Institute of Mathematical Sciences, New York University, June 1997.

[3] F. Alizadeh, J. P. A. Haeberly, and M. L. Overton. Primal-dual interior point methods for semidefinite programming. Technical Report 659, Computer Science Department, Courant Institute of Mathematical Sciences, New York University, 1994.

[4] K. M. Anstreicher and M. Fampa. A long-step path following algorithm for semidefinite programming problems. Working Paper, Department of Management Science, The University of Iowa, Iowa City, IA, 1996.

[5] K. E. Atkinson. *An Introduction to Numerical Analysis.* John Wiley & Sons, New York, second edition, 1989.

[6] S. J. Benson, Y. Ye, and X. Zhang. Solving large scale sparse semidefinite programs for combinatorial optimization. Working Paper, University of Iowa, Iowa City, IA, September 1997.

[7] S. J. Benson, Y. Ye, and X. Zhang. Mixed linear and semidefinite programming for combinatorial optimization. Working Paper, University of Iowa, Iowa City, IA, February 1998.

[8] D. Bertsimas and Y. Ye. Semidefinite relaxations, multivariate normal distributions, and order statistics. Working Paper, Sloan School, MIT, December 1997.

[9] B. Borchers. CSDP, a C library for semidefinite programming. Department of Mathematics, New Mexico Tech, Socorro, NM, 1998.

[10] N. Brixius, F. A. Potra, and R. Sheng. Sdpha: A matlab implementation of homogeneous interior point algorithms for semidefinite programming. Department of Computer Science, University of Iowa, Iowa City, IA, 1997.

[11] I. S. Duff. Sparse numerical linear algebra: Direct methods and preconditioning. In *The State of the Art in Numerical Analysis.* Clarendon Press, Oxford, 1997.

[12] J. Falkner, F. Rendl, and H. Wolkowicz. A computational study of graph partitioning. *Mathematical Programming*, 66(2):211–240, 1994.

[13] T. Fujie and M. Kojima. Semidefinite programming relaxation for nonconvex quadratic programs. Research Report B-298, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo 152, May 1995. to appear in *Journal of Global Optimization.*

[14] K. Fujisawa, M. Fukuda, M. Kojima, and K. Nakata. Numerical evaluation of SDPA (semidefinite programming algorithm). Research Report B-330, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro-ku, Tokyo 152, September 1997.

[15] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior point methods for semidefinite programming. Research Report B-324, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro-ku, Tokyo 152, September 1997.

[16] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145, 1995.

[17] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. ZIB Preprint SC 97-37, Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, Takustrasse 7, D-14195 Berlin, Germany, August 1997.

[18] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. *SIAM Journal of Optimization*, 6:342–361, 1996.

[19] S. E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. Technical Report DIKU-TR-97/9, Department of Computer Science, University of Copenhagen, July 1 1997.

[20] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[21] M. Kojima, S. Shindoh, and S. Hara. Interior point methods for the monotone semidefinite linear complementarity problem in symmetric matrices. Technical Report Research Reports on Information Sciences, Ser. B: Operations Research B-282, Dept. of Information Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152, Japan, April 1994. to appear in *SIAM Journal of Optimization.*

[22] C. Lin and R. Saigal. On solving large scale semidefinite programming problems - a case study of quadratic assignment problem. Technical report, Dept. of Ind. and Oper. Eng., University of Michigan, Ann Arbor, MI, 1997.

[23] L. Lovász and A. Shrijver. Cones of matrices and setfunctions, and $0-1$ optimization. *SIAM Journal on Optimization*, 1:166–190, 1990.

[24] R. D. C. Monteiro. Primal-dual path following algorithms for semidefinite programming. manuscript, School of ISyE, Georgia Institute of Technology, Atlanta, GA 30332, USA, To appear in *SIAM J. on Optimization*.

[25] R. D. C. Monteiro and P. R. Zanjácomo. Implementation of primal-dual methods for semidefinite programming based on monteiro and tsuchiya directions and their variants. Technical report, School of Ind. and Systems Engineering, Georgia Institute of Technology, Atlanta, 1997.

[26] M. Muramatsu. Affine scaling algorithm fails for semidefinite programming. Technical report, Department of Mechanical Engineering, Sophia University, Tokyo, Japan, 1996.

[27] Y. E. Nesterov and M. Todd. Primal-dual interior point methods for self scaled cones. Technical Report 1125, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York, 14853-3801, 1995.

[28] Yu. E. Nesterov. Quality of semidefinite relaxation for nonconvex quadratic optimization. Technical report, CORE Discussion Paper #9719, Belgium, March 1997.

[29] Yu. E. Nesterov and A. S. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming : Theory and Algorithms*. SIAM Publications, SIAM, Philadelphia, 1993.

[30] F. Rendl S. Polijak and H. Wolkowicz. A recipe for semidefinite relaxation for 0-1 quadratic programming. *Journal of Global Optimization*, 7:51–73, 1995.

[31] H. D. Sherali and W. P. Adams. Computational advances using the reformulation - linearization technique (rlt) to solve discrete and continuous nonconvex problems. *Optima*, 49:1–6, 1996.

[32] N. Z. Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25:1–11, 1987. Originally published in Tekhnicheskaya Kibernetika, No. 1, 1987, pp. 128 - 139.

[33] K. C. Toh, M. J. Todd, and R. H. Tutuncu. SDPT3 – a MATLAB software package for semidefinite programming. Technical Report TR1177, Cornell University, December 1996.

[34] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1): 49–95, 1996.

[35] H. Wolkowicz and Q. Zhao. Semidefinite programming for the graph partitioning problem, manuscript. *Discrete Applied Math.*, 1996. to appear subject to revision.

[36] Y. Ye. Approximating quadratic programming with quadratic constraints. Working Paper, Department of Management Science, The University of Iowa, Iowa City, 1997.

[37] Y. Ye. *Interior Point Algorithms : Theory and Analysis.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1997.

[38] Q. Zhao. *Semidefinite Programming for Assignment and Partitioning Problems.* PhD thesis, University of Waterloo, 1996.

[39] Q. Zhao, S. Karisch, F. Rendl, and H. Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problems. *J. Combinatorial Optimization*, 2(1):95–27, 1998.