Solving Sparse Semidefinite Programs Using the Dual Scaling Algorithm with an Iterative Solver*

Changhui Choi
Department of Management Sciences
The University of Iowa
Iowa City, Iowa 52242, U.S.A.

Yinyu Ye
Department of Management Sciences
The University of Iowa
Iowa City, Iowa 52242, U.S.A.

March 7, 2000

Abstract

Recently, the dual-scaling interior-point algorithm has been used to solve large-scale semidefinite programs arisen from discrete optimization, since it better exploits the sparsity structure of the problems than several other interior-point methods, while retain the same polynomial time complexity. However, solving a linear system of a fully dense Gram matrix in each iteration of the algorithm becomes the time-bottleneck of computational efficiency. To overcome this difficulty, we have tested using an iterative method, the conjugate gradient method with a simple preconditioner, to solve the linear system for a prescribed accuracy. In this report, we report computational results of solving semidefinite programs with dimension up to 20,000, which show that the iterative method could save computation time up-to 25 times of using the directed Cholesky factorization solver.

Key words. Semidefinite program, dual-scaling algorithm, conjugate gradient method, precondition.

*This work is partially supported by NSF grants DMI-9908077 and DMS-9703490.

1 Introduction

Consider the semidefinite programming problem

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

where C and A_i are given $n \times n$ symmetric matrices, $C \bullet X = \text{tr } C^T X = \sum_{jk} C_{jk} X_{jk}$, and $X \succeq 0$ means that X is positive semidefinite. Furthermore, we assume that A_i's are linearly independent, meaning that $\sum_{i=1}^m y_i A_i = 0$ implies $y_1 = \ldots = y_m = 0$;

The dual of (SDP) can be written as:

$$\text{(DSDP)} \qquad \begin{array}{ll} \text{Maximize} & b^T y \\ \text{Subject to} & \sum_{i=1}^m y_i A_i + S = C, \quad S \succeq 0, \end{array} \tag{2}$$

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

We have the following well-known duality theorem [13]:

Theorem 1 *(Strong Duality) Provided that (SDP) and (DSDP) are both feasible and there is a strictly interior point to either (SDP) or (DSDP), there is no duality gap.*

Thus, if both (SDP) and (DSDP) are well behaved or a primal and dual optimal solution pair (X^*) and (y^*, S^*) exists, then $C \bullet X^* = b^T y^*$.

There have been several theoretical achievements (see, e.g., Goemans and Williamson [5]) on using semidefinite programming to approximate combinatorial and graphical problems. The approach is to relax a combinatorial graph-optimization problem into a semidefinite program, then solve the relaxation program and construct a solution to the original combinatorial problem. This solution often possesses a guaranteed quality, although it is not 100% optimal. In these semidefinite relaxations, most often the constraint matrices are rank-one, i.e., $A_i = a_i a_i^T$, $a_i \in \mathcal{R}^n$; C represents the weighted graph-incidence or Laplacian matrix of the original graph; and S possesses the same sparse structure of C. These features result in considerable simplifications of the algorithm that we presented later.

There exist various interior-point algorithms that solve the semidefinite program in "polynomial" time, based on either the primal scaling (using X), the dual scaling (using S), or the primal-dual scaling (using both X and S), see, e.g., [1], [7], [10], [12], [11], [14], and [16]. (Other nonlinear programming based methods for semidefinite programming include Burer, Monteiro and Zhang[3], Helmberg and Kiwiel [6], Vavasis [17], Vanderbei and Benson [18], etc.) Recently, the dual-scaling algorithm (see Benson et al. [2]) has gained some attention, since it (using S) better

exploits the sparsity structure of the problems than the other methods. However, solving a linear system of a fully dense gram matrix in each iteration of the algorithm becomes the time-bottleneck of computational efficiency. To overcome this difficulty, we have tested using an iterative method, the conjugate gradient method with a simple preconditioner, to solve the linear system with a prescribed accuracy. In this report, we report computational results of solving semidefinite programs with dimension up to 20,000, which show that the iterative method could save computation time up-to 25 times of using the directed Cholesky factorization.

2 Dual-Scaling Algorithm

We briefly describe the dual-scaling interior-point algorithm (e.g., [2]). It is a modification of the dual-scaling linear programming algorithm, and uses the dual potential function

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

where $\bar{z} = C \bullet X$ for some X feasible for the primal. Starting from an interior feasible point (y^0, S^0) and a $\bar{z}^0$, the algorithm generates a sequence of $(y^k, S^k, \bar{z}^k)$ such that the dual potential function is reduced by a constant, which forces (y^k, S^k) converge to an optimal dual solution and generates X^k, as a by-product, converge to an optimal primal solution.

Define an operator $\mathcal{A}(X)$ as

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

The gradient of the dual potential function, for a fixed $\bar{z}$, becomes

$$\nabla \psi(y, \bar{z}) = -\frac{\rho}{\bar{z} - b^T y} b + \mathcal{A}(S^{-1}) \tag{3}$$

which is composed of two terms.

From a strictly feasible dual point (y^k, S^k), the kth iteration of the algorithm solves the following subproblem for the displacement vector δy^*:

$$\begin{array}{ll} \text{Minimize} & \nabla \psi^T(y^k, \bar{z}^k) \delta y \\ \text{Subject to} & \|(S^k)^{-.5} \left(\mathcal{A}^T(\delta y) \right) (S^k)^{-.5}\| \leq \alpha, \end{array} \tag{4}$$

where α is a positive constant less than 1 and

$$\mathcal{A}^T(\delta y) = \sum_{i=1}^{m} \delta y_i A_i;$$

then assigns

$$y^{k+1} = y^k + \delta y^*.$$

The optimal solution δy^* has a close form:

$$M^k \delta y^* + \beta \nabla \psi(y^k, \bar{z}^k) = M^k \delta y^* + \beta(-\frac{\rho}{\bar{z}^k - b^T y^k} b + \mathcal{A}((S^k)^{-1})) = 0 \tag{5}$$

for a step-size β, where

$$M^k = \begin{pmatrix} A_1(S^k)^{-1} \bullet (S^k)^{-1} A_1 & \cdots & A_1(S^k)^{-1} \bullet (S^k)^{-1} A_m \\ \vdots & \ddots & \vdots \\ A_m(S^k)^{-1} \bullet (S^k)^{-1} A_1 & \cdots & A_m(S^k)^{-1} \bullet (S^k)^{-1} A_m \end{pmatrix}$$

and

$$\mathcal{A}((S^k)^{-1}) = \begin{pmatrix} A_1 \bullet (S^k)^{-1} \\ \vdots \\ A_m \bullet (S^k)^{-1} \end{pmatrix}.$$

The matrix M^k is called Gram Matrix and it is positive definite when $S^k \succ 0$ and A_i's are linearly independent. Note that if A_is are rank-one, i.e., $A_i = a_i a_i^T$, $a_i \in \mathcal{R}^n$, then

$$M^k = \begin{pmatrix} (a_1^T(S^k)^{-1}a_1)^2 & \cdots & (a_1^T(S^k)^{-1}a_m)^2 \\ \vdots & \ddots & \vdots \\ (a_m^T(S^k)^{-1}a_1)^2 & \cdots & (a_m^T(S^k)^{-1}a_m)^2 \end{pmatrix} \quad \text{and} \quad \mathcal{A}((S^k)^{-1}) = \begin{pmatrix} a_1^T(S^k)^{-1}a_1 \\ \vdots \\ a_m^T(S^k)^{-1}a_m \end{pmatrix}.$$

Thus, we can construct M^k by m back-solvers once S^k is factorized; and the factorization of S^k is usually easy since S^k is intrinsically sparse for real-world applications.

To obtain δy^*, one usually computes δy_1 and δy_2 such that

$$\begin{aligned} M^k \delta y_1 &= b \\ M^k \delta y_2 &= -\mathcal{A}((S^k)^{-1}), \end{aligned} \tag{6}$$

and let

$$\beta = \frac{\alpha}{\sqrt{-\nabla \psi^T(y^k, \bar{z}^k)(\frac{\rho}{\bar{z}^k - b^T y^k}\delta y_1 + \delta y_2)}} = \frac{\alpha}{\sqrt{\nabla \psi^T(y^k, \bar{z}^k)(M^k)^{-1}\nabla \psi(y^k, \bar{z}^k)}}. \tag{7}$$

Then, assign

$$\delta y^* = \beta(\frac{\rho}{\bar{z}^k - b^T y^k}\delta y_1 + \delta y_2). \tag{8}$$

In Benson et al. [2], the Cholesky factorization of M^k has been used to compute δy_1 and δy_2. Since M^k is an $m \times m$ almost always fully dense matrix, the factorization of M^k is extremely expensive and constitutes most of the computation work. Next we show that this work could be reduced dramatically by iteratively solving the systems of linear equations using a preconditioned conjugate gradient method.

3 Conjugate Gradient Method

We quickly review the conjugate gradient method in solving a linear system $Ax = b$ for an $m \times m$ real positive definite matrix A and a real vector b. Starting from an initial point x^0, the method generates a sequence of vectors that converges to the solution that satisfies $Ax = b$. Since this method was first introduced by Hestenes and Stiefel in 1952 ([8]), it has been widely used in solving large-scale linear systems bearing numerous applications and modifications.

To solve the system, one uses the fact that convex quadratic function

$$\phi(x) = \frac{1}{2}x^T Ax - b^T x \tag{9}$$

has the minimum solution x^* that satisfies $Ax^* = b$. Then, the conjugate gradient method generates a series of nonzero vectors p^i's that satisfy the equalities

$$(p^i)^T Ap^j = 0, \qquad \text{for all } \ i \neq j. \tag{10}$$

This property is known as conjugacy. Starting from the initial $x^0 \in \mathcal{R}^m$, the method then computes a sequence x^k, using these conjugate vectors as bases, which converges to x^*. More precisely, the iterative updates are:

$$x^{k+1} = x^k + \alpha^k p^k$$

where

$$\alpha^k = -\frac{(r^k)^T p^k}{(p^k)^T Ap^k}$$

and

$$r^k = b - Ax^k.$$

It is well known that, for any x^0, the conjugate gradient sequence converges to x^* in at most m steps. In fact, if A has only r distinct eigenvalues, then method will terminate in at most r steps. The Chebyshev inequality also holds:

$$\|x^k - x^*\|_A \leq 2\left(\frac{\sqrt{c}-1}{\sqrt{c}+1}\right)^k \|x^0 - x^*\|_A \tag{11}$$

where $\|u\|_A = (u^T Au)^{\frac{1}{2}}$ and $c = \frac{\lambda_{max}}{\lambda_{min}}$, the ratio of the max-eigenvalue over the min-eigenvalue of A.

One variation of the original conjugate gradient method is to use preconditioning. Using a linear translation $\hat{x} = B^{0.5}x$ for a nonsingular matrix B, the system becomes

$$\Phi(\hat{x}) = \frac{1}{2}\hat{x}^T(B^{-0.5}AB^{-0.5})\hat{x} - (B^{-0.5}\hat{x})^T b. \tag{12}$$

Then, one can apply the method to minimize $\Phi(\hat{x})$. The goal is to choose a nonsingular matrix B such that the eigenvalues of $B^{-0.5}AB^{-0.5}$ are clustered closely to each other and; thereby, the number of needed conjugate steps can be reduced. On the other hand, the computation of $B^{-0.5}$ must be cost-effective. In our experiment, the diagonal of A is used as B. Other popular choices are incomplete Cholesky factorizations or band matrices of A.

Thus, the simple standard preconditioned conjugate gradient method, using the diagonal of A as the preconditioner, can be descried as follows.

Algorithm PCG Start by choosing any $x^0 \in \mathcal{R}^m$ and set $r^0 = b - Ax^0$, $q^0 = Diag(A)^{-1}r^0$, $p^0 = q^0$ and $k = 0$. For a given tolerance ϵ,

while($\|r^k\|/\|b\| > \epsilon$)

$\alpha^k = \frac{(r^k)^T q^k}{(p^k)^T A p^k}$;

$x^{k+1} = x^k + \alpha^k p^k$;

$r^{k+1} = r^k - \alpha^k A p^k$;

$q^{k+1} = Diag(A)^{-1} r^{k+1}$;

$\beta^k = \frac{(r^{k+1})^T q^{k+1}}{(r^k)^T q^k}$;

$p^{k+1} = q^{k+1} + \beta^k p^k$;

$k = k + 1$;

end;

Now we apply this iterative method to solving (6). To generate a pair of δy's, we run the PCG method two times—one against b and one against $-\mathcal{A}((S^k)^{-1})$. But in actual computation we utilize the fact that they use the same right-hand matrix M^k. Therefore we run the pair of conjugate gradient method simultaneously to save some computation work. We stop the conjugate steps when the bigger of the relative norms, is smaller than the given tolerance ϵ. In one conjugate step, we need two matrix-vector multiplications and several vector-vector products. And we do not need any memory space to allocate matrices, so that the method is space efficient.

After solving these two linear equations of (6) approximately we return to the same procedure of Benson et al. [2] and continue the major iteration. Note that, since (6) is not solved exactly, the by-product primal update $\bar{z}^k = C \bullet X^k$ (see [2]) will not be an exactly feasible primal objective value; but $b^T y^k$ remains a feasible dual objective value. Thus, the algorithm terminates at a slightly sup-optimal (sub-optimal) value for the primal (dual), depending on the tolerance ϵ for the conjugate gradient process. Interestingly, the tolerance does not need to be too small for solving all our test problems, still producing excellent approximate combinatorial solutions for the original graphical problems. In our experiment, ϵ is set to be 0.1 and $x^0 = 0$ is used as the initial point.

The conjugate direction method has been used by Kaliski and Ye [9] and Resende and Veiga [15] in LP dual interior-point algorithms, and by Fujisawa and Kojima[4] in an SDP primal-dual interior-point algorithm.

4 Computational result

The following tables illustrate the test results of solving various types of graphical problems. The major iteration of the dual-scaling algorithm is terminated when the (approximate) primal-dual relative gap is below 10^{-4}. Then, the same rounding methods of Benson et al. [2] are used to

construct combinatorial solutions.

In Table 1 we compares the conjugate gradient method with the Cholesky factorization for solving the Max-Cut of the G-set graphs, generated by the machine independent graph generator, **rudy**, of G. Rinaldi and also used in Benson et al. [2]. "Dim" represents the size of the problem (the number of vertices in the graph), "Spars" is the density of the graphs, "SFden" is the density of the Cholesky factor of S (after reordering). The next two sections contain the solution times, the cut values, and their comparisons between the CG (conjugate gradient) and the CH (Cholesky factorization) solvers. Column "CG/CH" shows the ratio of the CG solution time to the CH time, and column "(CG-CH)/CH" is the ratio of the difference of the two resulting cut values to the cut value resulted from the CH solver. There, we see that the CG solution time is uniformly less than the CH solution time, while the two max-cut values remains extremely close.

				Time			Cut-Value		
Name	Dim	Spars(%)	SFden(%)	CG	CH	CG/CH	CG	CH	(CG-CH)/CH
G11	800	0.63	2.6	16.61	28.38	0.59	542	532	0.01
G12	800	0.63	3.64	17.66	29.84	0.59	540	534	0.01
G13	800	0.63	4.37	18.19	28.74	0.63	564	554	0.01
G14	800	1.59	14.49	35.23	47.76	0.74	2922	2982	-0.02
G15	800	1.58	13.88	32.11	54.9	0.58	2938	2975	-0.01
G20	800	1.59	14.05	32.03	58.46	0.55	838	876	-0.04
G21	800	1.58	13.86	37.56	61.29	0.61	841	855	-0.01
G22	2000	1.05	47.88	4123.31	4325.62	0.95	12960	12989	-0.00
G23	2000	1.05	47.68	3233.52	3402.86	0.95	13006	13002	0.00
G24	2000	1.05	48.29	3250.7	3549.09	0.92	12933	12985	-0.00
G30	2000	1.05	48.53	3718.93	3929.19	0.94	3038	3080	-0.01
G31	2000	1.05	48.70	3835.70	3784.45	1.01	2851	2936	-0.02
G32	2000	0.25	1.70	142.61	616.96	0.23	1338	1302	0.02
G33	2000	0.25	1.85	132.48	352.90	0.37	1330	1286	0.03
G34	2000	0.25	2.28	156.66	383.67	0.40	1334	1292	0.03
G48	3000	0.17	1.73	343.64	1113.6	0.31	6000	6000	0
G49	3000	0.17	1.57	303.3	1197.64	0.25	6000	6000	0
G50	3000	0.17	1.28	264.59	1706.73	0.16	5880	5880	0
G55	5000	0.1	1.04	1474.77	17854.76	0.08	9960	9960	0
G56	5000	0.12	8.76	15618.6	31341.04	0.50	3634	3649	-0.00
G57	5000	0.1	0.82	1819.76	15401.37	0.12	3320	3208	0.03
G60	7000	0.08	8.46	58535.13	89700.25	0.65	13610	13658	-0.00
G61	7000	0.08	8.46	52719.63	90491.23	0.58	5252	5273	-0.00
G62	7000	0.07	0.66	5187.23	63139.88	0.08	4612	4476	0.03
G63	7000	0.05	0.41	21386.79	62248.04	0.34	8017	8059	-0.00
G64	7000	0.18	11.19	102163.92	150017.27	0.68	7624	7861	-0.03
G65	8000	0.06	5.21	58309.63	133412.95	0.43	13261	13286	-0.00
G70	10000	0.03	0.31	33116.22	193932.17	0.17	9456	9499	-0.00
G72	10000	0.05	0.69	12838.07	265875.75	0.05	6644	6370	0.04
G77	14000	0.04	0.52	32643.36	804486.15	0.04	9418	9048	0.04
G81	20000	0.03	0.38	131778.21		0.00	13448		

Table 1: Comparison of the CG and CH solvers for solving Max-Cut problems (Time is in seconds).

This table shows a significant speed improvement for the dual-scaling algorithm from the CH factorization to the CG solver. For G77 (this graph has 14000 vertices), the cut solution takes 9 days to obtain when the CH factorization is used; while it takes about 9 hours when the CG solver is used. At the same time, the solution quality of the CG solver is even better than that of the CH factorization.

The CG solver can be implemented without explicitly storing M^k at each iteration, which makes the solver use much less memory than the CH method does. For example, G81 of Table 1 cannot be solved by the CH method due to the shortage of memory

Figure 1 shows the number of CG steps in each major iteration of the algorithm for solving the G38 max-cut problem. The horizontal axis is the indices of the major algorithm iterations, and the vertical axis is the number of CG steps used in each major iteration. The figure depicts a typical behavior of the CG solver for solving these max-cut problems. At the beginning, the method needs very few CG steps. Then more and more steps are needed; after few iterations the number starts to decrease. At the end, 1 or 2 CG steps are sufficient. This trend alludes that one might develop a better preconditioner during the middle of the iterative process to further reduce the number of CG steps.

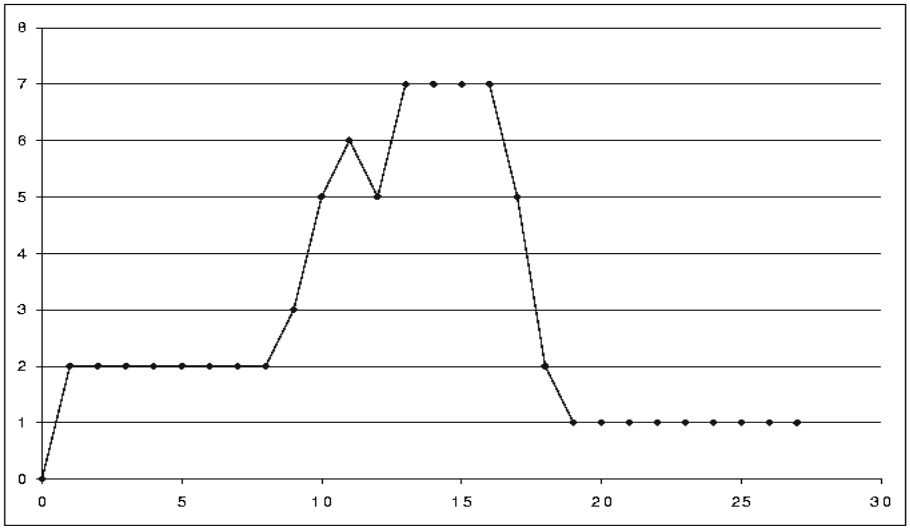

Figure 1: The number of CG steps for solving Max-Cut G38.

Another interesting trend is: the sparser of the graph, the faster of the CG method or the less of needed CG steps. Figure 2 depicts the relation between the density of the Cholesky factor of S—"SFden" (horizontal axis), and the solution time ratio—"CG/CH" (vertical axis) of Table 1.

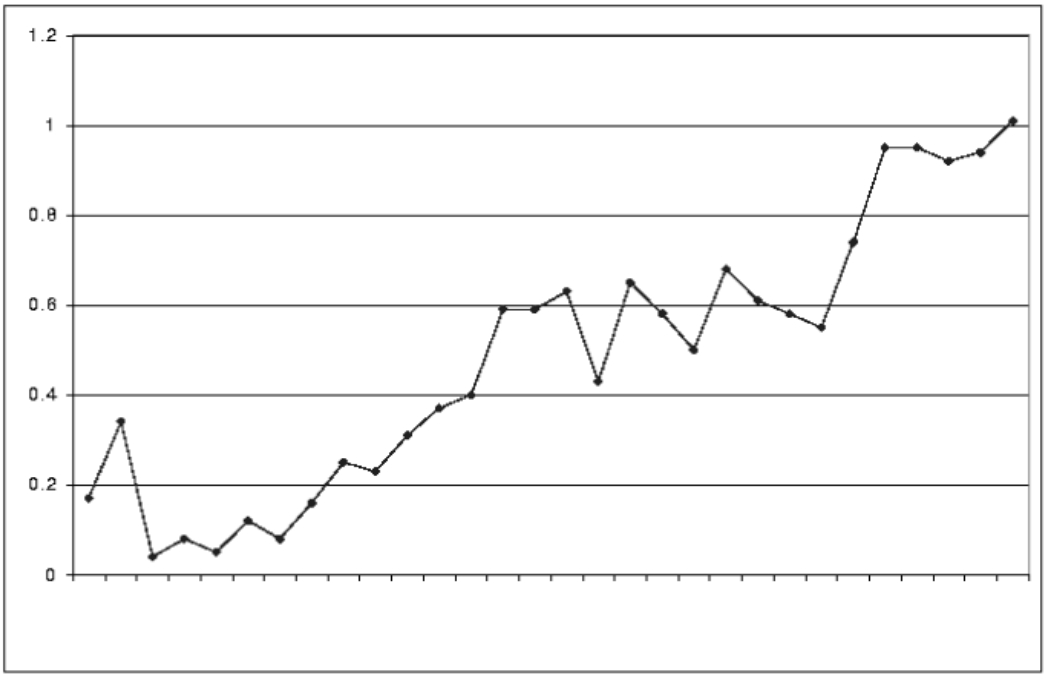

Figure 2: The solution time ratio and the density of S-factor.

Table 2 shows the test results for solving these graphs as the Max-Bisection—to find the max-cut while the two sides have the same number of vertices—using the CG solver. In addition to the G-set graphs, we also include in our test a set of circuit graphs from real-industrial applications. There, "# of Iterations" is the number of major iterations occurred in the dual-scaling algorithm, and "Max # of CG-steps" is the largest number of CG steps in each major iteration during the entire iterative process. We see that the time used in solving the Max-Bisection problem is about the same as that of solving the Max-Cut problem, and the number of CG steps needed in each major iteration remains low. Again, a substantial speed improvement has been achieved here. For example, using the CH solver the bisection of circuit "s15850" needs 88 hours, but now it needs only 4.5 hours.

Name	Dim	Spars(%)	SFden(%)	# of Iterations	Bisection-Value	Max # of CG-steps	Time
G48	3000	0.17	1.73	23	6000	2	511.54
G49	3000	0.17	1.57	23	5996	2	519.41
G50	3000	0.17	1.28	20	5878	2	462.34
G55	5000	0.1	1.04	22	9958	2	1793.39
G56	5000	0.12	8.76	46	3611	10	20793.52
G57	5000	0.1	0.82	32	3322	3	2090.77
G60	7000	0.08	8.46	39	13640	8	48949.86
G61	7000	0.08	8.46	34	5195	9	42467.23
G62	7000	0.07	0.66	34	4576	3	5445.99
G63	7000	0.05	0.41	82	7826	10	13166.68
G64	7000	0.18	11.19	76	7700	20	123409.74
G70	10000	0.03	0.31	85	1953	7	18254.4
G71	10000	0.02	0.03	65	1444	7	11459.39
G72	10000	0.05	0.69	35	6628	5	15383.86
G74	10000	0.06	2.07	58	6560	18	64514.15
G77	14000	0.04	0.52	40	9450	3	36446.69
G81	20000	0.03	0.38	31	13402	9	334824.22
balu	801	3.80	14.11	26	693	3	40.20
p1	833	1.48	6.25	26	852	2	31.47
bm1	882	1.32	5.48	25	848	2	33.88
t4	1515	9.12	12.23	27	1539	2	439.73
t3	1607	4.49	10.22	42	1494	4	588.14
t2	1663	6.08	11.76	31	1630	3	679.56
t6	1752	6.60	17.77	32	1526	3	1390.15
struct	1952	0.49	2.79	26	1753	1	185.17
t5	2595	6.49	10.21	38	2574	3	3039.91
19ks	2844	3.31	6.69	31	3128	3	2221.06
p2	3014	0.63	5.50	35	2851	3	2223.24
s9234	5866	0.11	0.58	64	5600	3	5181.22
biomed	6514	2.98	12.28	33	5355	2	46750.65
s13207	8772	0.08	0.40	52	8326	3	12077.33
s15850	10470	0.07	0.25	54	9940	2	15966.03

Table 2: Results of using the CG solver for solving Max-Bisection problems (Time is in seconds).

References

[1] 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.

[2] Steven J. Benson, Yinyu Ye, and Xiong Zhang, "Mixed linear and semidefinite programming for combinatorial and quadratic optimization," *Optimization Methods and Software* **11&12** (1999) 515-544.

[3] Samuel Burer, Renato D.C. Monteiro, Yin Zhang, "Interior-Point Algorithms for Semidefinite Programming Based on A Nonlinear Programming Formulation," Technical Report TR99-27,

Department of Computational and Applied Mathematics, Rice University, Houston, Texas 77005, December 1999.

[4] 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.

[5] M. X. Goemans and D. P. Williamson, " Improved approximation algorithms for Maximum Cut and Satisfiability problems using semidefinite programming," *Journal of ACM* 42 (1995) 1115–1145.

[6] C. Helmberg and K.C. Kiwiel, "A Spectral Bundle Method with Bounds," Preprint SC 99-37, Konrad-Zuse-Zentrum fuer Informationstechnik Berlin, 14195 Berlin, Germany, December 1999.

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

[8] Hestenes and Stiefel, "Methods of conjugate gradients for solving linear systems," *Nat. Bur. Standards J. Review*, 49:409-436, 1952.

[9] J. A. Kaliski and Y. Ye, "A short-cut potential reduction algorithm for linear programming," *Management Science*, 39:757–776, 1993.

[10] M. Kojima, S. Shindoh, and S. Hara, "Interior point methods for the monotone semidefinite linear complementarity problem in symmetric matrices." 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.*

[11] 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.

[12] R. D. C. Monteiro and Y. Zhang, "A unified analysis for a class of path-following primal-dual interior-point algorithms for semidefinite programming," manuscript, School of Ind. and Systems Engineering, Georgia Institute of Technology, Atlanta, 1996.

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

[14] 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, 14853-3801m 1995, to appear in *SIAM J. Optimization.*

[15] M. G. C. Resende and G. Veiga, "An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks," *SIAM Journal on Optimization*, 3:516–537, 1993.

[16] M. J. Todd, "On search directions in interior-point methods for semidefinite programming," Technical Report No. 1205, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853-3801, October 1997.

[17] S. Vavasis, "A note on efficient computation of the gradient in semidefinite programming," Working Paper, Department of Computer Science, Cornell University, September 1999.

[18] R.J. Vanderbei and H. Yurttan Benson, "On Formulating Semidefinite Programming Problems as Smooth Convex Nonlinear Optimization Problems," ORFE 99-01, Dept. of Operations Research and Financial Engineering, Princeton University, Princeton NJ, December 1999.