DSDP3: Dual Scaling Algorithm for General Positive Semidefinite Programming *

Steven J. Benson
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL U.S.A.

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

February 12, 2001

Abstract

We implement a dual scaling algorithm for positive semidefinite programming to handle a broader class of problems than could be solved with previous implementations of the algorithm. With appropriate representations of constraint matrices, we can solve general semidefinite programs and still exploit the structure of large-scale combinatorial optimization problems. Computational results show that our preliminary implementation is competitive with primal-dual solvers on many problems requiring moderate precision in the solution and is superior to primal-dual solvers for several types of problems.

Key words. Semidefinite programming, dual potential reduction algorithm.

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

1 Introduction

In the past several years, positive semidefinite programming (SDP) has been one of the most active fields of numerical optimization. At least two factors explain this high level of interest. First, applications of SDP have been found in areas as diverse as structural design, control theory, and combinatorial optimization. Second, although interior point methods adopted from linear programming have proven reliable on small and medium-sized problems, the computational and storage demands of these methods have exhausted the capacity of most computers and limited the size of problems that can be solved.

The positive semidefinite program in standard form is

$$\text{(SDP)} \qquad \begin{array}{ll} \inf & C \bullet X \\ \text{Subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m, \\ & X \in K, \end{array} \tag{1}$$

where $K = K_1 \oplus K_2 \oplus \cdots \oplus K_r$ and K_l is the cone of $n_l \times n_l$ symmetric positive semidefinite matrices, and $C, A_i \in \Re^{n \times n}$ are given symmetric matrices. The operation $C \bullet X = \text{tr } C^T X = \sum_{jk} C_{jk} X_{jk}$ and $X(\succeq) \succ 0$ means that X is (semi) positive definite. Furthermore, we assume the matrices A_i are linearly independent, meaning that $\sum_{i=1}^m y_i A_i = 0$ implies $y_1 = \ldots = y_m = 0$. Matrices X that satisfy the constraints are called feasible, while the others are called infeasible.

The dual of (SDP) can be written as

$$\text{(DSP)} \qquad \begin{array}{ll} \sup & b^T y \\ \text{Subject to} & \displaystyle\sum_{i=1}^m y_i A_i + S = C, \quad S \in K, \end{array} \tag{2}$$

where $b, y \in \Re^m$ and y_i is the i^{th} element of the vector y.

Following conventional notations, let

$$\mathcal{A}X = [\ A_1 \bullet X \quad \ldots \quad A_m \bullet X\ ]^T \quad \text{and} \quad \mathcal{A}^T y = \sum_{i=1}^m A_i y_i,$$

We have the following well known duality theorem [15]:

Theorem 1 *(Strong Duality) Provided there exist feasible points for both (SDP) and (DSP) and an interior feasible point for at least one of (SDP) and (DSP), the optimal values of these two problems are equal.*

Thus, under this relatively mild condition the primal and dual optimal solution pair (X^*) and (y^*, S^*) exist, and $C \bullet X^* = b^T y^*$.

Examples of SDP problems arise in control theory, truss topology design, and combinatorial optimization. In many of these problems, the constraints are dense and have full rank. Many combinatorial problems, however, have constraints that have rank one and a very sparse structure. The SDP relaxation of the maximum-cut problem, analyzed by Goemans and Williamson [11], has n constraints, and each constraint has exactly one nonzero element. Sparsity and low rank constraints are also present in the SDP relaxations of graph bisection, Lovasz theta number, graph coloring, and satisfiability problems. Real applications of these problems can be particularly large.

Various approaches have been tried to solve these problems. These approaches include primal-dual interior point methods (see Todd [18] for a survey of these methods) and a dual scaling interior point method of Benson, Ye, and Zhang [3]. Other types of methods that have been developed and applied to combinatorial problems, such as the maximum-cut problem, include the partial Lagrangian approach of Helmberg and Rendl [12], which uses a spectral bundle method to solve the nondifferentiable convex program, and transformation to a constrained nonlinear program which was first proposed by Homer and Peinado [13] and further developed by Burer and Monteiro [7] and Burer, Monteiro, and Zhang [8][9].

Details concerning the convergence of the feasible start dual scaling algorithm and its advantages over primal-dual methods can be found in [3] and [21]. The advantages of the algorithm are as follows

1. The cost of each iteration is relatively low. For many problems, the computation of the primal matrix X requires considerable computational effort, which is unnecessary in this algorithm.
2. The memory requirements of the dual algorithm are significantly lower than the requirement for primal-dual methods, enabling it to solve larger problems. Other than the data, a good implementation of the dual algorithm requires only two additional matrices: one for the dual matrix and one for the reduced linear system.
3. The dual scaling algorithm can exploit sparsity in the data to save computation time when the Cholesky factorization of the dual matrix is sparse.

The next section summarizes the dual scaling algorithm, which is a modification of the linear programming algorithm. Convergence of the algorithm has been discussed previously[3]. The purpose of this paper is to discuss our implementation of the algorithm and its success in solving a broad collection of problems.

2 Dual Scaling Algorithm

Given a dual point (y, S) such that $\mathcal{A}^T y + S - C = R$ and $S \succ 0$, and a barrier parameter $\hat{\mu} > 0$, each iteration of the dual scaling algorithm takes a step in the Newton direction to maximize the function

$$\phi(y) = b^T y + \hat{\mu} \ln \det S \tag{3}$$

subject to $\mathcal{A}^T y + S = C$. The dual step direction Δy solves the linear system

$$\begin{pmatrix} A_1 S^{-1} \bullet S^{-1} A_1 & \cdots & A_1 S^{-1} \bullet S^{-1} A_m \\ \vdots & \ddots & \vdots \\ A_m S^{-1} \bullet S^{-1} A_1 & \cdots & A_m S^{-1} \bullet S^{-1} A_m \end{pmatrix} \Delta y = \frac{1}{\hat{\mu}} b - \mathcal{A}(S^{-1}) - \mathcal{A}(S^{-1} R S^{-1}) \tag{4}$$

and $\Delta S = -\mathcal{A}^T \Delta y - R$. For notational convenience, we label the left-hand matrix of (4) M. For any feasible X, this linear system (4) can also be derived by taking the Schur complement of the equations

$$\mathcal{A}(\Delta X) = 0 \qquad \mathcal{A}^T(\Delta y) + \Delta S = -R \qquad \hat{\mu} S^{-1} \Delta S S^{-1} - \Delta X = X - \hat{\mu} \hat{S}^{-1}, \tag{5}$$

which are the Newton equations for the nonlinear system

$$\mathcal{A}X = b \qquad \mathcal{A}^T y + S = C \qquad \hat{\mu} S^{-1} = X. \tag{6}$$

A third derivation of the method minimizes the dual potential function

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

over a trust region [3]. In this dual potential function $\bar{z} = C \bullet X$ for a feasible matrix X, $\rho > n + \sqrt{n}$, and $\hat{\mu} = \frac{\bar{z} - b^T y}{\rho}$. The relationships between these derivations can be found in [21].

The step direction comprises of three parts:

$$dy_1 = M^{-1} b, \tag{7}$$

$$dy_2 = M^{-1} \mathcal{A}(S^{-1}), \tag{8}$$

$$dy_3 = M^{-1} \mathcal{A}(S^{-1} R S^{-1}), \tag{9}$$

which represent the affine scaling, centering, and feasibility directions, respectively. Together,

$$\Delta y = \frac{1}{\hat{\mu}} dy_1 - dy_2 + dy_3. \tag{10}$$

The algorithm then selects a step size β_{k+1} such that $y^{k+1} = y^k + \beta_{k+1} \Delta y$ and $S^{k+1} = S^k + \beta_{k+1} \Delta S \succ 0$.

Using the dual step direction and (5), we give a primal matrix X satisfying the linear constraints:

$$X = \hat{\mu} S^{-1} + \hat{\mu} S^{-1} \Delta S S^{-1}. \tag{11}$$

This X, which also minimizes $\|S^{.5} X S^{.5} - \hat{\mu} I\|$ subject to $\mathcal{A}X = b$, is positive semidefinite if and only if

$$S + \mathcal{A}^T(\Delta y) \succeq 0. \tag{12}$$

Matrix (12) has the same sparsity pattern as S and can be stored in the same data structure as S, eliminating the need for an additional matrix structure.

Since X is not needed to compute the next step direction, it does not have to be computed during the algorithm. Creating the primal matrix may be costly, but when (y, S) is a feasible point, the upper bound given by evaluating the primal objective value $C \bullet X(\bar{z}^k)$ requires much less work.

$$\begin{aligned} C \bullet X &= b^T y + X \bullet S \\ &= b^T y + \bar{\mu} \left(\Delta y^T \mathcal{A}(S^{-1}) + n \right). \end{aligned} \tag{13}$$

Since the vectors $\mathcal{A}(S^k)^{-1}$ and Δy were found previously, the cost of computing a primal objective value is the cost of a dot product of two vectors.

With a feasible dual starting point and appropriate choices for $\hat{\mu}$ and β, convergence results in [3] show that either the new dual point (y, S) or the new primal point X is feasible and reduces the Tanabe-Todd-Ye primal-dual potential function

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

enough to achieve linear convergence.

It has been previously documented [3] that the dual scaling algorithm can save time and memory compared to primal-dual method by utilizing sparsity in the problem and not explicitly computing the matrix X. The next two sections discuss effective methods to compute the linear system (4), select the parameter $\hat{\mu}$, and select a step size.

2.1 Constraints As a Sum of Rank-One Matrices

Many problems in positive semidefinite programming, especially combinatorial problems, have constraints with a very low rank. Efficient SDP solvers can offer significant savings in both memory and computational power by utilizing this structure. Since matrices can be written as a sum of rank-one matrices, let A_i be written as

$$A_i = \sum_{k=1}^{\mathcal{R}_i} \alpha_{ik} a_{ik} a_{ik}^T, \tag{14}$$

where $\mathcal{R}_i$ is the number of scalars α_{ik} and vectors $a_{ik} \in \Re^n$. used to representation of the matrix. The elements in the matrix M can be computing by using

$$M_{ij} = \sum_{k=1}^{\mathcal{R}_i} \sum_{l=1}^{\mathcal{R}_j} \alpha_{ik} \alpha_{jl} \left(a_{ik} S^{-1} a_{jl}^T \right)^2. \tag{15}$$

The matrix computation requires relatively little work space to store intermediate computations. With the following algorithm.

Compute M: To compute the lower triangular part of M, initially set M equal to zero, factor $S = LL^T$, and do the following:

For $i = 1 : m$;

For $k = 1 : \mathcal{R}_i$;

Solve $Sw_i = a_{ik}$; For $j = i : m$; $M_{ij} = M_{ij} + \sum_{l=1}^{\mathcal{R}_j} \alpha_{ik}\alpha_{jl}(a_{jl}^T w)^2$; end.

end.

end.

If each constraint has full rank, this technique costs $O(n^3m^2 + n^3m)$ arithmetic operations, which is higher than other implementations. Computing each element of M with $M_{ij} = A_i \bullet (S^{-1}A_jS^{-1})$ costs only $O(n^2m^2 + n^3m)$ operations. Many problems, especially combinatorial problems [6], have constraints with very low rank. If each constraint has rank one, M can be computed using $O(mn^2 + m^2n)$ operations excluding the cost of factoring S. When the constraints have a dense rank one structure, the complexity of this technique is an order of magnitude less than techniques that do not explicitly account for the rank-one structure.

Any combination of rank-one matrices that sum to A_i can be used to compute the linear system in this way, but the eigenvalues and eigenvectors provide a particularly efficient representation. There are efficient routines for computing eigenvalues and eigenvectors, and the orthogonality of these vectors reduces the risk on numerical instabilities that may be associated with the process. Furthermore, the eigenvectors corresponding to the nonzero eigenvalues comprise the smallest set of vectors that can be used to represent a matrix in this manner.

In the dual scaling algorithm, this technique can also be used in other computations as well. Each iteration requires the computation of $\mathcal{A}S^{-1}$ and $\mathcal{A}(S^{-1}RS^{-1})$, which can be computed by

$$A_i \bullet S^{-1} = \sum_{k=1}^{\mathcal{R}_i} \alpha_{ik} \left(a_{ik}^T S^{-1} a_{ik} \right)$$

and

$$A_i \bullet S^{-1}RS^{-1} = \sum_{k=1}^{\mathcal{R}_i} \alpha_{ik} \left(a_{ik}^T S^{-1} RS^{-1} a_{ik} \right).$$

These two vectors can be computed in the same routine for the matrix M and require very little additional effort.

For some problems, especially combinatorial problems arising from networks, the dual matrix S has a sparse Cholesky factorization. In these problems, it is significantly less expensive to use the Cholesky factorization to solve the linear system $Sw = a_{jl}$ than it would be to multiply a_{jl} by S^{-1}, which is almost always dense. Furthermore, use of backward and forward substitutions eliminates the need to calculate the inverse of the matrix.

2.2 Selection of $\hat{\mu}$

The linear algebra used to compute and solve the linear system (4) affects the cost of each iteration. However, the choice of parameters in the algorithm, especially $\hat{\mu}$, has an enormous impact upon the convergence of the algorithm.

The central path is characterized by a parameter $\mu \in (0, \infty)$, which connects the analytic center of the feasible region to the analytic center of the solution set [21]. For each $\mu > 0$, there is a feasible point along the central path that maximizes (3). At this point, the first order conditions state that

$$b^T(\frac{1}{\mu}dy_1 - dy_2) = 0. \tag{16}$$

The dual algorithm tries to follow the central path to a solution, where $\mu = 0$, by reducing this parameter at each iteration.

Although the initial dual point rarely lies on the central path, the early iterations of DSDP calculate an appropriate value for μ that satisfies (16). Since M is positive definite, $b^T dy_1 > 0$. The solutions to (SDP) and (DSP) lie along the edge of the feasible set, and the barrier function forces the steps away from the boundary, so usually $b^T dy_2 > 0$ and there exists a solution to (16). If not, our implementation selects $\mu = 1$. Primal-dual methods calculate μ by using the current primal and dual points, but the dual algorithm must use another technique, like the one described here, until a positive semidefinite matrix X can be found.

To find a feasible matrix X, DSDP uses several values of $\hat{\mu}$. These values are generally $0.1\mu, 0.4\mu, 0.7\mu$, and 0.9μ. Multiple values of $\hat{\mu}$ are used to find a feasible X because they provide a good heuristic for selecting a value $\hat{\mu}$ for the dual step. To compute the dual step direction, DSDP selects the smallest value of $\hat{\mu}$ that provides a positive semidefinite X. If no positive semidefinite X is found at this iteration, the algorithm computes a μ that satisfies (16) and sets $\hat{\mu} = 0.7\mu$.

2.3 Trust Region

The step size can be chosen by using either a line search or a trust region. Most solvers use a line search to determine the longest step that keeps the dual matrix positive definite. Since the step direction is a Newton direction, one simple and effective line search procedure has been to initially set $\beta_{k+1} = 1.0$ and test whether or not $S^k + \beta_{k+1}\Delta S$ is positive definite. If not, the backtracking line search reduces β_{k+1} until the dual matrix is positive definite. Alternative line searches that compute the exact step size to the boundary of the cone can be used, but these methods require expensive eigenvalue computations whose cost cannot be amortized over other parts of the algorithm.

In choosing a step size, we can also employ a trust region. As mentioned earlier, the dual step direction also minimizes

$$\begin{array}{ll} \text{Minimize} & \nabla\psi^T(y^k, \bar{z}^k)(y - y^k) \\ \text{Subject to} & \|(S^k)^{-.5}\left(\mathcal{A}^T(y - y^k)\right)(S^k)^{-.5}\| \leq \alpha, \end{array} \tag{17}$$

such that $\mathcal{A}^T y + S = C$, $\bar{z} = C \bullet X$ for a feasible matrix X, $\rho > n + \sqrt{n}$, and $\hat{\mu} = \frac{\bar{z} - b^T y}{\rho}$. For feasible points, the solution to this problem will also be feasible when $\alpha < 1.0$.

Using this trust region formulation, the step length β can be written [3] as

$$\beta = \frac{\alpha}{\sqrt{(\frac{1}{\hat{\mu}}b - A(S^{-1}))^T \Delta y}}.$$

Our implementation usually uses a trust region radius of $\alpha = 3$, although a slightly larger radius is chosen when $\sqrt{(b/\hat{\mu} - A(S^{-1}))^T \Delta y} > 15$. If this step length does not provide a positive definite matrix S^{k+1}, β is reduced until the dual matrix is positive definite. Experience has shown that limiting the size of the trust region slows the convergence of the algorithm on some problems but improves the overall robustness of the algorithm.

Much of the theory of the dual scaling algorithm is based on a feasible starting point. For that reason, we encourage very large steps to achieve feasibility as quickly as possible. When the iterates are infeasible, DSDP3 selects a maximum step length using $\alpha = n$. Never, however, is $\beta > 1$. A Cholesky factorization can test whether the matrix is positive definite. Slightly more sophisticated line searches were implemented in previous versions [2], but this simple backtracking technique is efficient because the next iterate will need a Cholesky factorization to compute (4).

To avoid computing ΔS explicitly, let $R^{k+1} = (1 - \beta_{k+1})R^k$, and compute the next point $S^{k+1} = C - \mathcal{A}^T y^{k+1} + R^{k+1}$. This technique avoids the use of an additional matrix to store ΔS. Aside from the data, only three matrices are needed in the algorithm: one to store M, one to store S and its Cholesky factor, and one to store R. To take advantage of the sparsity inherent in many large combinatorial problems, we use a sparse data structure to represent the dual matrix, and a diagonal matrix to represent the dual infeasibility. In fact, R is usually set to be a multiple of the identity matrix.

3 DSDP 3

The path-following algorithm outlined in the preceding section can be stated as follows.

DSDP3 ALGORITHM. Given $y^0 \in \Re^m$, let R^0 be a multiple of the identity matrix such that $S^0 = C - \mathcal{A}^T y^0 + R^0 \succ 0$, set $\bar{z} = \infty$, $\mu^* = \infty$, $k = 0$, and do the following:

while $\bar{z} - b^T y^k > \epsilon_1$ and $\|R\| > \epsilon_2$ **do**

1. Compute $\mathcal{A}(S^k)^{-1}$, $\mathcal{A}((S^k)^{-1}R(S^k)^{-1})$, and the matrix M^k of (4).
2. Solve (7), (8), and (9) for the step directions dy_1, dy_2, and dy_3, respectively.
3. If $\mu^* = \infty$, compute μ^k using (16).
4. Select one or more values of $\hat{\mu}$. For each $\hat{\mu}$ compute (12). If (12) is positive definite and $\hat{\mu} < \mu^*$, let $\mu^* = \hat{\mu}$ and compute an upper bound $\bar{z}$.
5. Select one value for $\hat{\mu}$, calculate Δy, and compute a step size β^{k+1} such that $y^{k+1} = y^k + \beta \Delta y$, $R^{k+1} = (1 - \beta_{k+1})R^k$ and $S^{k+1} = C - \mathcal{A}^T y^{k+1} + R^{k+1} \succ 0$
6. Set $\mu^{k+1} = \beta^{k+1}\hat{\mu} + (1 - \beta^{k+1})\mu^k$.
7. $k := k + 1$.

end

DSDP3 implements this algorithm in the C language for good performance and memory management. Like CSDP [4], DSDP3 can solve a positive semidefinite program with a set of subroutines, and like SDPT3 [19] and SeDuMi [17], DSDP3 can be used from within the Matlab environment. The Matlab interface uses arguments that are very similar to the ones used in SDPT3. The DSDP3 package an also contains a stand-alone version that reads files in SDPA [10] format.

The DSDP3 solver uses several techniques to improve the efficiency and robustness of the algorithm. Like other SDP solvers, DSDP3 takes advantage of the block structure that is present in some problems. When multiple distinct blocks are present in the problem and identified by the solver, DSDP3 will express each block as a matrix and write the each block within the constraint matrices as the sum of rank-one matrices. These blocks are used separately to compute the linear system (7).

To reduce the possibility of numerical error, DSDP3 also scales the primal objective matrix such that each element in the objective matrix has a maximum magnitude of 1.0. The starting point is generally set to zero, but for some combinatorial problems such as the maximum cut problem, we use feasible initial vectors y^0 that makes the dual matrix S^0 diagonally dominant.

If $C - \mathcal{A}^T y^0$ is not positive definite, DSDP3 adds a multiple of the identity matrix to the dual matrix to make it positive definite. It adds R^0 equal to the identity matrix initially and continues to double it until $S = C - \mathcal{A}^T y^0 + R^0 \succ 0$ and then multiplies this value of R^0 by ten.

The dual infeasibility decreases from one iterate to the next. On some problems, however, the step sizes become very short. To increase robustness of the software, DSDP doubles R when the step sizes are particularly short.

DSDP3 includes both a Cholesky factorization and a diagonally preconditioned conjugate residual method to solve (7). Either method can be chosen, but DSDP3 selects a hybrid strategy by default.

As shown by Toh and Kojima [20], the number of iterates required by the conjugate residual method is small for early iterates of interior point methods. As the matrix becomes more poorly conditioned in later iterates, the conjugate residual method requires more iterates. By default, DSDP3 switches from the iterative method to a direct method when the number of conjugate residual iterates on one linear system exceeds $m/5$. Within each iterative linear solve, DSDP uses a relative stopping tolerance of 10^{-8}.

4 Computational Results

The speed and robustness of DSDP3.1 has been tested using the SDPLIB [5] test suite, the Seventh DIMACS Implementation Challenge [16] problems, and several independent test problems.

Table 1 shows the dual solution value and the number of iterates required to achieve it for each of the SDPLIB problems. In each of these problems, we tried to compute a solution with six digits of accuracy using the default options and an initial dual vector equal to zero. DSDP3 solved each of the 94 problems with a level of accuracy in the objective value and dual solution similar to those of primal dual solvers. In most cases, DSDP3 found six digits of accuracy. In some cases, the number of iterations required to compute only three digits of accuracy was significantly less than the number shown in Table 1. Problems `thetaG11` and `thetaG51` could not be solved using DSDP3 and problems `hinf12` – `hinf15` could not be solved with high precision, but primal dual solvers also have difficulties with these problems. On several other problems, DSDP3 required a starting point different from the default. The constraints that bound the diagonal of the primal matrix in the combinatorial problem `qpG51` have dual variables that we used to create a feasible initial point with diagonally dominant dual matrix, and the vector with each component equal to 1.0 was used to create a feasible point in `arch8`. For problems `control6`, `control8`, `control10`, and `control11`, each element of the dual vector was set to the arbitrary value of -1. These initial points are not particularly special; experimental results showed, however, that DSDP3 converged to an optimal solution for a wide range of other initial points. Application developers are typically aware of the structure of a problem and can provide good initial solutions.

DSDP3 proved to be most successful on the `mcp`, `gpp`, `theta`, `max`, `equal`, `qp` and `qap` problems, which arise from the relaxation of combinatorial optimization problems. DSDP3 solved each of these problems to the desired accuracy. Furthermore, the iteration counts for these problems are lower than the iteration count for the other problems.

DSDP3 also correctly identified infeasibility in the four problems with that characteristic. When the primal problem is infeasible, DSDP3 returns with a high dual value and a code indicating primal infeasibility. DSDP3 does not return a feasible primal solution with a large objective value as some primal-dual solvers do, but it does correctly detect the dual infeasibility.

The success of DSDP3 on the SDPLIB library indicates the robustness of the dual scaling algorithm and the DSDP3 software package. We note, however, that the six digits of accuracy that we report hold only for the dual solution and objective value. The primal solution matrix returned by DSDP3 may have significantly less precision. Because the dual matrix S get closes to the solution, it contains eigenvalues that are very close to zero. The primal solution X is computed by using the product of S^{-1}, S^{-1}, and a third matrix (11), so numerical difficulties often impede the precision of this matrix. We find that best primal solutions that DSDP3 can return are those that are found after solving the problem with only 2 - 3 digits of accuracy.

Table 1: Performance on the SDPLIB Problems

Problem	its.	dobj		Problem	its.	dobj
arch0	66	0.566517		infd2	15	unbounded
arch2	56	0.674882		infp1	56	infeasible
arch4	69	1.072066		infp2	28	infeasible
arch8	69	7.056982		maxG11	36	629.1648
control1	26	17.78463		maxG32	45	1567.640
control2	31	8.300001		maxG51	63	4006.256
control3	38	13.63327		maxG55	62	12869.87
control4	34	19.79423		maxG60	46	15222.27
control5	51	16.88360		mcp100	24	226.1574
control6	73	37.30444		mcp124-1	26	141.9905
control7	69	20.62508		mcp124-2	25	269.8802
control8	69	20.28637		mcp124-3	27	467.7501
control9	67	14.67543		mcp124-4	24	864.4119
control10	89	38.53307		mcp250-1	28	317.2644
control11	59	31.95874		mcp250-2	26	531.9301
equalG11	73	629.1553		mcp250-3	27	981.1726
equalG51	83	4005.601		mcp250-4	26	1681.960
gpp100	39	-44.94354		mcp500-1	44	598.1486
gpp124-1	37	-7.343028		mcp500-2	34	1070.057
gpp124-2	37	-46.86226		mcp500-3	30	1847.970
gpp124-3	36	-153.0139		mcp500-4	31	3566.738
gpp124-4	36	-418.9874		qap5	36	-436.0000
gpp250-1	43	-15.44480		qap6	99	-381.2145
gpp250-2	42	-81.86865		qap7	35	-424.8196
gpp250-3	40	-303.5388		qap8	59	-756.9552
gpp250-4	42	-747.3275		qap9	45	-1409.941
gpp500-1	56	-25.30782		qap10	61	-1092.607
gpp500-2	54	-156.0574		qpG11	26	2448.659
gpp500-3	63	-513.0172		qpG51	59	1181.800
gpp500-4	53	-1567.008		ss30	111	20.23962
hinf1	29	2.032600		theta1	30	23.00000
hinf2	48	10.96706		theta2	30	32.87917
hinf3	31	56.94079		theta3	27	42.16698
hinf4	31	274.7639		theta4	36	50.32123
hinf5	29	362.2140		theta5	33	57.23231
hinf6	28	448.9279		theta6	41	63.47710
hinf7	45	390.8124		thetaG11	100	f
hinf8	31	116.1460		thetaG51	100	f
hinf9	50	236.2493		truss1	23	-8.999994
hinf10	39	108.7119		truss2	28	-123.3804
hinf11	40	65.86209		truss3	28	-9.109992
hinf12	100	0.000001		truss4	30	-9.009993
hinf13	39	44.34282		truss5	67	-132.6356
hinf14	44	13.00275		truss6	63	-901.0010
hinf15	100	23.95561		truss7	34	-900.0013
infd1	15	unbounded		truss8	99	-133.1145

Table 2: Performance on the DIMACS Implementation Challenge Problems

Problem	CSDP	SDPA	SDPT3	SeDuMi	DSDP3
toruspm3-8	99	435	276	955	42
toruspm3-15	15857	m	m	m	16450
torusg3-8	114	106	267	1311	43
torusg3-15	16006	m	m	m	17897
truss5	4	4	22	4	17
truss8	23	23	178	27	222
hinf12	f	1	7	1	1
hinf13	f	1	5	1	1
copo14	54	234	368	30	1132
copo23	3607	38894		5651	f
ham-7-5-6	89	485	423	365	115
ham-9-8	328	f	1876	1482	999
bm1	f	6532	5289	30661	1282

Tables 2 compares the performance of the DSDP3 with the performance of the four primal dual solvers entered in the Seventh DIMACS Implementation Challenge. The thirteen problems shown on this table contain positive semidefinite matrix variables but no second order cone variables. Large problems that could not be solved by most of the interior point solvers were not included in table. These results were compiled by Hans Mittelmann[14] on a Pentium II (512 MB RAM, 450 MHz) processor running Linux-2.4.0t8 and MATLAB6.0b. Each test used the default starting point and parameters. The table show the number of seconds required to solve the problem. An "i" indicates the problem was declared to be infeasible, an "m" indicates that the solver required an excessive amount of memory, and an "f" indicates that the solver fails to produce a solution with one degree of accuracy.

DSDP3 performed very well on the maximum cut problems `toruspm3-8`, `toruspm3-15`, `torusg3-8`, `torusg3-8`, minimum bisection problem `bm1`, and Lovasz theta number problems `ham-7-5-6` and `ham-9-8`. In these combinatorial problems, DSDP3 matched or outperformed the primal dual solvers. In `toruspm3-8`, for instance, DSDP3 was more that twice as fast as the fastest primal dual solver and more that twenty times faster than another primal dual solver.

The dual algorithm is well suited for these problems because the factorization of the dual matrix is usually sparse and the number of constraints is relatively small compared with the dimension of the variable matrices. Primal dual algorithms typically spend a lot of time on these problems performing $O(n^3)$ operations such as matrix-matrix multiplications and matrix factorizations required to compute the primal matrix X at each iteration. DSDP3 saves a significant amount of time because it does not require the primal matrix X for convergence. The lower memory requirements of the dual algorithm allow it to solve larger problems. In some of the larger maximum cut problems, the primal dual methods failed because of excessive memory requirements.

Primal dual solvers performed significantly better on `copo` problems and large truss design problems. In these cases, the number of constraints in these problems significantly exceeds the dimension of the variable matrices and the constraint matrices have high ranks. On these problems the cost of one dual iteration is at least as large as one primal dual iteration. Since the number of dual iterations is generally higher, it cannot compete well with primal dual methods with stronger convergence properties. Nonetheless, DSDP matched the performance of the best primal dual solvers in the two small matrix inequality problems `hinf12` and `hinf13`.

5 Concluding Remarks

Our preliminary implementation of the dual scaling algorithm for positive semidefinite programming efficiently solves the standard test problems. The algorithm can be applied to all semidefinite programs, but its features are particularly well suited for applications from combinatorial optimization where the constraint matrices have low ranks. The DSDP3 implementation of the algorithm emphasizes these advantages with its choice of data structures for the variable and data matrices. On classes of problems such as maximum cut, theta, and minimum bisection problems, it returns solutions in significantly less time than do the primal dual solvers. Furthermore, because the computer memory requirements of dual scaling algorithm are less than those of primal dual algorithms, it can solve larger problems than some primal dual solvers. The dual scaling algorithm typically requires more iterations than do primal dual algorithms, and the implementation choices magnify the disadvantages of the dual scaling algorithm. Nonetheless, when only moderate precision in the solution is required, the DSDP3 package solves many other problems in a time that is similar to the times needed by primal dual solvers. The DSDP3 package is available to the public and contains several interfaces that make it easy to use for application developers [1].

References

[1] S. J. Benson and Y. Ye. DSDP3.2 User Guide. http://www-unix.mcs.anl.gov/∼ benson, November 2000.

[2] S. J. Benson, Y. Ye, and X. Zhang. Mixed linear and semidefinite programming for combinatorial optimization. *Optimization Methods and Software*, 10:515–544, 1999.

[3] S. J. Benson, Y. Ye, and X. Zhang. Solving large scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal of Optimization*, 10:443–461, 2000.

[4] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Software and Methods*, 11:613–623, 1999.

[5] B. Borchers. SDPLIB 1.0 : A collection of semidefinite programming test problems. Technical report, Faculty of Mathematics, Institute of Mining and Technology, New Mexico Tech, Socorro, NM, USA, July 1998.

[6] B. Borchers. Presentation at the Seventh DIMACS Implementation Challenge Piscataway, NJ, November 2, 2000.

[7] S. Burer and R. D. C. Monteiro. An efficient algorithm for solving the MAXCUT SDP relaxation. Manuscript, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA USA, December 1998.

[8] S. Burer, R. D. C. Monteiro, and Y. Zhang. Solving semidefinite programs via nonlinear programming. part I: Transformations and derivatives. Technical report, School of Industrial and Systems Engineering, Georgia Tech, Atlanta, GA, September 1999.

[9] S. Burer, R. D. C. Monteiro, and Y. Zhang. Solving semidefinite programs via nonlinear programming. part II: Interior point methods for a subclass of SDPs. Technical report, School of ISyE, Georgia Tech, Atlanta, GA, October 1999.

[10] K. Fujisawa, M. Fukuda, M. Kojima, and K. Nakata. Numerical Evaluation of SDPA (SemiDefinite Programming Algorithm). In *High Performance Optimization.* Kluwer Academic Press, 1999, pp. 267–301.

[11] 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.

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

[13] S. Homer and M. Peinado. On the performance of polynomial-time CLIQUE algorithms on very large graphs. Technical report, Boston University, Boston, MA, 1994.

[14] Hans D. Mittelmann. Independent Benchmark Results http://plato.la.asu.edu/, November 2000.

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

[16] Gabor Pataki and Stefan H. Schmieta. The Seventh DIMACS Implementation Challenge: 1999-2000. http://dimacs.rutgers.edu/Challenges/Seventh/, 1999.

[17] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones". *Optimization Software and Methods*, 11:625-653, 1999.

[18] M. J. Todd. On search directions in interior-point methods for semidefinite programming. Technical Report 1205, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, October 1997.

[19] K. C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 – A MATLAB software package for semidefinite programming, version 1.3 *Optimization Software and Methods*, 11:545-581, 1999.

[20] K. C. Toh and M. Kojima. Solving some large scale semidefinite programs via the conjugate residual method. Working paper, August 2000.

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