DSDP5: Software for Semidefinite Programming

Preprint ANL/MCS-P1289-0905

STEVEN J. BENSON
Mathematics and Computer Science Division, Argonne National Laboratory
and
YINYU YE
Department of Management Science and Engineering, Stanford University

DSDP implements the dual-scaling algorithm for semidefinite programming. The source code for this interior-point algorithm, written entirely in ANSI C, is freely available. The solver can be used as a subroutine library, as a function within the Matlab environment, or as an executable that reads and writes to data files. Initiated in 1997, DSDP has developed into an efficient and robust general-purpose solver for semidefinite programming. Its features include a convergence proof with polynomially bounded worst-case complexity, primal and dual feasible solutions when they exist, certificates of infeasibility when solutions do not exist, initial points that can be feasible or infeasible, relatively low memory requirements for an interior-point method, sparse and low-rank data structures, extensibility that allows applications to customize the solver and improve its performance, a subroutine library that enables it to be linked to larger applications, scalable performance for large problems on parallel architectures, and a well-documented interface and examples of its use. The package has been used in many applications and tested for efficiency, robustness, and ease of use.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization; D.2.13 [**Software Engineering**]: Reusable Software - Reusable libraries

General Terms: semidefinite programming, linear matrix inequalities, conic programming, interior-point methods

Additional Key Words and Phrases: dual-scaling algorithm

1. SEMIDEFINITE PROGRAMMING

Over the last fifteen years, considerable attention has been paid to optimization problems in which the variable is not a vector but a symmetric matrix that is required to be positive semidefinite. Semidefinite programming (SDP) is concerned with choosing a symmetric positive semidefinite matrix to optimize a linear function subject to linear constraints. Part of the interest arises from the tight bounds that these problems generate for global optimization [Henrion and Lasserre 2003], and hard combinatorial optimization [Goemans and Williamson 1995; Anjos 2005; Yildirim and Fan 2005]. Other uses of semidefinite programming arise in quantum chemistry [Zhao et al. 2004], free material optimization [Zowe et al. 1997], stability of differential equations [Löfberg 2001], sum of squares optimization [Prajna et al. 2004], and graph realization and distance geometry [Biswas and Ye 2004; So and Ye 2005].

Part of the interest in semidefinite programming also derives from the great advances in our ability to solve such problems efficiently in theory and in practice. Interior-point methods such as those implemented in DSDP, CSDP [Borchers 1999a],

SeDuMi [Sturm 1999], SDPA [Yamashita et al. 2003], and SDPT3 [Toh et al. 1999] enjoy strong theoretical convergence properties. The generalized penalty method in Pennon [Kocvara and Stingl 2003], the low-rank factorization method in SDPLR [Burer and Monteiro 2003], and a spectral bundle method [Helmberg and Rendl 2000] have also proven effective for these kinds of problems. Surveys by Todd [2001] and Wolkowicz et al. [2000] present examples of SDP and the algorithms most frequently used for solving them.

The initial version of DSDP [Benson et al. 2000] was released to solve the semidefinite relaxations of the maximum-cut and equal-cut problems from combinatorial optimization. A second release [Benson et al. 1999] targeted combinatorial problems whose constraint matrices each had a rank of one. Later releases of DSDP could solve broader classes, and results from them have been presented at conferences [Center for Discrete Mathematics and Theoretical Computer Science 2000]. After years of development, the software has matured and this manuscript presents the implementation and performance of DSDP for general semidefinite programming.

2. NOTATION

The DSDP package implements a dual-scaling algorithm to find solutions (X_j, y_i, S_j) to linear and semidefinite optimization problems of the form

$$(P) \quad \inf \sum_{j=1}^{p} \langle C_j, X_j \rangle \text{ subject to } \sum_{j=1}^{p} \langle A_{i,j}, X_j \rangle = b_i, \ i = 1, \ldots, m, \quad X_j \in K_j,$$

$$(D) \quad \sup \sum_{i=1}^{m} b_i \ y_i \text{ subject to } \sum_{i=1}^{m} A_{i,j} y_i + S_j = C_j, \ j = 1, \ldots, p, \ S_j \in K_j.$$

In this formulation, b_i and y_i are real scalars.

For semidefinite programming, the data $A_{i,j}$ and C_j are symmetric matrices of dimension n_j ($\mathbb{S}^{n_j}$), and the cone K_j is the set of symmetric positive semidefinite matrices of the same dimension. The inner product $\langle C, X \rangle := C \bullet X := \sum_{k,l} C_{k,l} X_{k,l}$, and the symbol $\succ$ ($\succeq$) means the matrix is positive (semi)definite. In linear programming, A_i and C are vectors of real scalars, K is the nonnegative orthant, and the inner product $\langle C, X \rangle$ is the usual vector inner product.

More generally, users specify $C_j, A_{i,j}$ from an inner-product space V_j that intersects a cone K_j. Using the notation summarized in Table I, let the symbol $\mathcal{A}$ denote the linear map $\mathcal{A} : V \to \mathbb{R}^m$ defined by $(\mathcal{A}X)_i = \langle A_i, X \rangle$; its adjoint $\mathcal{A}^* : \mathbb{R}^m \to V$ is defined by $\mathcal{A}^* y = \sum_{i=1}^{m} y_i A_i$. Equivalent expressions for (P) and (D) can be written

$$\begin{array}{lll} (P) & \inf \langle C, X \rangle \text{ subject to } \mathcal{A}X = b, & X \in K, \\ (D) & \sup \ b^T y \quad \text{subject to } \mathcal{A}^* y + S = C, & S \in K. \end{array}$$

Formulation (P) will be referred to as the *primal* problem, and formulation (D) will be referred to as the *dual* problem. Variables that satisfy the linear equations are called feasible, whereas the others are called infeasible. The interior of the cone will be denoted by $\hat{K}$, and the interior feasible sets of (P) and (D) will be denoted by $\mathcal{F}^0(P)$ and $\mathcal{F}^0(D)$, respectively.

Table I. Basic terms and notation for linear (LP), semidefinite (SDP), and conic programming.

Term	LP	SDP	Conic	Notation
Dimension	n	n	$\sum n_j$	n
Data Space ($\ni C, A_i$)	$\mathbb{R}^n$	$\mathbb{S}^n$	$V_1 \oplus \ldots \oplus V_p$	V
Cone	$x, s \geq 0$	$X, S \succeq 0$	$X, S \in K_1 \oplus \ldots \oplus K_p$	$X, S \in K$
Interior of Cone	$x, s > 0$	$X, S \succ 0$	$X, S \in \hat{K}_1 \oplus \ldots \oplus \hat{K}_p$	$X, S \in \hat{K}$
Inner Product	$c^T x$	$C \bullet X$	$\sum \langle C_j, X_j \rangle$	$\langle C, X \rangle$
Norm	$\|x\|_2$	$\|X\|_F$	$\left(\sum \|X_j\|^2\right)^{1/2}$	$\|X\|$
Product	$[x_1 s_1 \ldots x_n s_n]^T$	XS	$X_1 S_1 \oplus \ldots \oplus X_p S_p$	XS
Identity Element	$[1 \ldots 1]^T$	I	$I_1 \oplus \ldots \oplus I_p$	I
Inverse	$[1/s_1 \ldots 1/s_n]^T$	S^{-1}	$S_1^{-1} \oplus \ldots \oplus S_p^{-1}$	S^{-1}
Dual Barrier	$\sum \ln s_j$	$\ln \det S$	$\sum \ln \det S_j$	$\ln \det S$

3. DUAL-SCALING ALGORITHM

This section summarizes the dual-scaling algorithm for solving (P) and (D). For simplicity, parts of this discussion assume that the cone is a single semidefinite block, but an extension of the algorithm to multiple blocks and other cones is relatively simple. This discussion also assumes that the A_is are linearly independent, there exists $X \in \mathcal{F}^0(P)$, and a starting point $(y, S) \in \mathcal{F}^0(D)$ is known. The next section discusses how DSDP generalizes the algorithm to relax these assumptions.

It is well known that under these assumptions, both (P) and (D) have optimal solutions X^* and (y^*, S^*), which are characterized by the equivalent conditions that the duality gap $\langle X^*, S^* \rangle$ is zero and the product $X^* S^*$ is zero. Moreover, for every $\nu > 0$, there exists a unique primal-dual feasible solution (X_ν, y_ν, S_ν) satisfies the perturbed optimality equation $X_\nu S_\nu = \nu I$. The set of all solutions $\mathcal{C} \equiv \{(X_\nu, y_\nu, S_\nu) : \nu > 0\}$ is known as the central path, and $\mathcal{C}$ serves as the basis for path-following algorithms that solve (P) and (D). These algorithms construct a sequence $\{(X, y, S)\} \subset \mathcal{F}^0(P) \times \mathcal{F}^0(D)$ in a neighborhood of the central path such that the duality gap $\langle X, S \rangle$ goes to zero. A scaled measure of the duality gap that proves useful in the presentation and analysis of path-following algorithms is $\mu(X, S) = \langle X, S \rangle / n$ for all $(X, S) \in K \times K$. Note that for all $(X, S) \in \hat{K} \times \hat{K}$, we have $\mu(X, S) > 0$ unless $XS = 0$. Moreover, $\mu(X_\nu, S_\nu) = \nu$ for all points (X_ν, y_ν, S_ν) on the central path.

The dual-scaling algorithm applies Newton's method to $\mathcal{A}X = b$, $\mathcal{A}^* y + S = C$, and $X = \nu S^{-1}$ to generate

$$\mathcal{A}(X + \Delta X) = b, \tag{1}$$

$$\mathcal{A}^*(\Delta y) + \Delta S = 0, \tag{2}$$

$$\nu S^{-1} \Delta S S^{-1} + \Delta X = \nu S^{-1} - X. \tag{3}$$

Equations (1)-(3) will be referred to as the Newton equations; their Schur complement is

$$\nu \begin{pmatrix} \langle A_1, S^{-1} A_1 S^{-1} \rangle & \cdots & \langle A_1, S^{-1} A_m S^{-1} \rangle \\ \vdots & \ddots & \vdots \\ \langle A_m, S^{-1} A_1 S^{-1} \rangle & \cdots & \langle A_m, S^{-1} A_m S^{-1} \rangle \end{pmatrix} \Delta y = b - \nu \mathcal{A} S^{-1}. \tag{4}$$

The left-hand side of this linear system is positive definite when $S \in \hat{K}$. In this

manuscript, it will sometimes be referred to as M. DSDP computes $\Delta' y := M^{-1}b$ and $\Delta'' y := M^{-1}\mathcal{A}S^{-1}$. For any ν,

$$\Delta_\nu y := \frac{1}{\nu}\Delta' y - \Delta'' y$$

solves (4). We use the subscript to emphasize that ν can be chosen after computing $\Delta' y$ and $\Delta'' y$ and that the value chosen for the primal step may be different from the value chosen for the dual step.

Using $\Delta_\nu y$ and (3), we get

$$X(\nu) := \nu\left(S^{-1} + S^{-1}(\mathcal{A}^*\Delta_\nu y)S^{-1}\right), \tag{5}$$

which satisfies $\mathcal{A}X(\nu) = b$. Because $X(\nu) \in \hat{K}$ if and only if

$$C - \mathcal{A}^*(y - \Delta_\nu y) \in \hat{K}, \tag{6}$$

DSDP applies a Cholesky factorization on (6) to test the condition. If $X(\nu) \in \hat{K}$, a new upper bound

$$\bar{z} := \langle C, X(\nu)\rangle = b^T y + \langle X(\nu), S\rangle = b^T y + \nu\left(\Delta_\nu y^T \mathcal{A}S^{-1} + n\right) \tag{7}$$

can be obtained without explicitly computing $X(\nu)$. The dual-scaling algorithm does not require $X(\nu)$ to compute the step direction defined by (4), so DSDP does not compute it unless specifically requested. This feature characterizes the algorithm and its performance.

For semidefinite blocks, DSDP uses two techniques for computing M. Both techniques use a decomposition of each data matrix as a sum of vector outer products. That is,

$$A_i = \sum_{r=1}^{rank} \alpha_{i,r} a_{i,r} a_{i,r}^T$$

such that $\alpha_{i,r} \in \mathbb{R}$ and $a_{i,r} \in \mathbb{R}^n$. An eigenvalue/eigenvector decomposition, for example, satisfies this form. Since M is symmetric, DSDP computes only half of the matrix, which we will assume is the lower half matrix. After inverting S, it computes each row i of M using one the following two techniques.

Technique M1:
Set $V \in \mathbb{S}^n$ to zero;
for $r \leftarrow 1 : rank(A_i)$ **do**
  $w \leftarrow S^{-1}a_{i,r}$;
  $V \leftarrow V + \alpha_{i,r} w w^T$;
end for
for $j \leftarrow 1 : i$ **do**
  $M_{i,j} \leftarrow M_{i,j} + \langle A_i, V\rangle$;
end for

Technique M2:
for $r \leftarrow 1 : rank(A_i)$ **do**
  $w \leftarrow S^{-1}a_{i,r}$;
  for $j \leftarrow 1 : i$ **do**
    $M_{i,j} \leftarrow M_{i,j} + \alpha_{j,k} w^T A_j w$;
  end for
end for

The first technique is most common, but the second is used when rank of A_i is small. For semidefinite blocks such that S is sparse, DSDP does not invert S. Instead, it solves $Sw = a_{i,r}$ using the Cholesky factorization of S. Both of these techniques exploit the rank structure of the data matrices instead of the sparsity structure that is exploited by other interior-point solvers [Fujisawa et al. 1997].

The preconditioned conjugate gradient (CG) method computes $\Delta' y$ and $\Delta'' y$. Initially, the preconditioner is the diagonal of M; but once the number of CG iterations exceeds $m/50$, DSDP preconditions it using the Cholesky factorization of M. Using the explicit factor, CG requires a single iteration until the condition number deteriorates. Usually, M is a dense matrix that uses LAPACK to implement the Cholesky factorization, triangular solves, and matrix-vector multiplication. Sparse matrix formats that also implement these operations get used when appropriate.

The improvement provided by the step direction and the convergence of the dual-scaling algorithm depends on the proximity of the current solution to a point on the central path. Let $\rho > 0$ and $\bar{z} = \langle C, X \rangle$ for some feasible X. The dual potential function

$$\psi(y) := \rho \log(\bar{z} - b^T y) - \ln \det S \tag{8}$$

has a gradient

$$\nabla \psi = -\frac{\rho}{\bar{z} - b^T y} b + \mathcal{A} S^{-1} \tag{9}$$

and a minimum (y_ν, S_ν) for $\nu = \frac{\bar{z} - b^T y}{\rho}$. The first term in (8) improves the objective value of (y, S), and the second term forces it away from the boundary of the cone. Because $\nabla \psi = 0$ at (y_ν, S_ν), norms of the gradient are a measure of the distance between the point and (y_ν, S_ν). The norm of

$$P(\nu) = \frac{1}{\nu} S^{.5} X(\nu) S^{.5} - I \tag{10}$$

can be interpreted as the scaled distance between $X(\nu)$ and X_ν. Furthermore,

$$\begin{aligned} \|P(\nu)\|^2 = & = \|S^{.5}(S^{-1} - S^{-1}\Delta S S^{-1})S^{.5} - I\|^2 \\ & = \|S^{-.5}\Delta S S^{-.5}\|^2 \\ & = \langle S^{-.5}\Delta S S^{-.5}, S^{-.5}\Delta S S^{-.5}\rangle \\ & = \langle \Delta S, S^{-1}\Delta S S^{-1}\rangle \\ & = \textstyle\sum_i \sum_j \Delta_\nu y_i \langle A_i, S^{-1} A_j S^{-1}\rangle \Delta_\nu y_j \\ & = \Delta_\nu y^T M \Delta_\nu y \\ & = -\nabla \psi^T \Delta_\nu y \\ & = \|\nabla \psi\|^2_{M^{-1}}. \end{aligned}$$

DSDP sets $\rho = (\bar{z} - b^T y)/\nu$ and selects a steplength $\alpha_d \in (0, 1]$ such that $y^+ := y + \alpha_d \Delta_\nu y$, $C - \mathcal{A}^* y^+ \in \hat{K}$, and y^+ achieves sufficient reduction in (8). Since M is positive definite, the step direction will reduce the potential function for sufficiently small step sizes. For $\|P(\nu)\| > 1$, the line search reduces the potential function by at least 0.05. To find α_d, DSDP computes the distance to the boundary of the cone, which is the reciprocal of the largest eigenvalue of $-L^{-1}\Delta S L^{-T}$, where L is the lower triangular Cholesky factorization of S and $\Delta S \leftarrow -\mathcal{A}^* \Delta_\nu y$. The Lanczos procedure analyzed by Toh [2002] computes this eigenvalue using the forward and backward solutions involving L and matrix-vector multiplications involving ΔS. Denoting the maximum eigenvalue as λ_{max}, DSDP initially sets $\alpha_d = \min\{1.0, 0.95/\lambda_{max}\}$ and backtracks until it achieves sufficient descent or the steplength is less than a prescribed termination tolerance. Line searches based on a simple backtracking strategy were used in earlier versions, but the Lanczos strategy reduced the number of iterations by about 30% over a broad test set.

Before taking the dual step, however, DSDP updates the bound $\bar{z}$ and selects a new value for ν. A simple strategy discussed in Ye [1997] sets $\nu = (\bar{z} - b^T y)/\rho$ for $\rho > n + \sqrt{n}$. DSDP allows users to specify a parameter $\rho_n > 1$ that will set $\rho = n \times \rho_n$. Another simple strategy fixes ν for several iterations and decreases it only when $\|P(\nu)\|$ is sufficiently small.

With the current value of ν denoted as ν^k, computational tests show that if $X(\nu^k) \in \hat{K}$, better performance can be achieved by using the smallest value of ν such that $X(\nu) \in \hat{K}$. To estimate this value, let $\nu = \sigma\nu^k$ for $0 < \sigma < 1$ and use the identity $\frac{1}{\sigma} = 1 + (\frac{1-\sigma}{\sigma})$ to write

$$\begin{aligned}\Delta_\nu y &= \tfrac{1}{\nu}\Delta' y - \Delta'' y \\ &= \tfrac{1}{\nu^k}\Delta' y - \Delta'' y + (\tfrac{1-\sigma}{\sigma})\tfrac{1}{\nu^k}\Delta' y \\ &= \Delta_{\nu^k} y + (\tfrac{1-\sigma}{\sigma})\tfrac{1}{\nu^k}\Delta' y.\end{aligned}$$

The Lanczos line search computes the largest step α_ν such that

$$C - \mathcal{A}^*(y - \Delta_{\nu^k} y) + \alpha_\nu \mathcal{A}^*(\frac{1}{\nu^k}\Delta' y) \in K.$$

If one sets $\alpha_\nu = \frac{1-\sigma}{\sigma}$, then $\sigma = \frac{1}{1+\alpha_\nu}$. Using the steplength, DSDP sets $\nu := \frac{1}{1+0.95\alpha_\nu}\nu^k$. If $X(\nu^k) \notin \hat{K}$, one line search finds the largest α_p such that $\hat{S} \leftarrow C - \mathcal{A}^*(y - \alpha_p \Delta_{\nu^k} y) \in \hat{K}$, a second line search computes α_ν such that $\hat{S} - \alpha_\nu \mathcal{A}^*(\alpha_p/\nu^k)\Delta' y \in \hat{K}$, and $\nu = \alpha_p \nu^k/(1+\alpha_\nu) + (1-\alpha_p)\mu^k$. In either case, an upper bound of $\mu^k = (\bar{z} - b^T y)/n$ and a lower bound of μ^k/ρ_n ($\rho_n > 1$) limit the change in the barrier parameter.

To get closer to the central path and make further use of M, whose computation and factorization usually dominate the computation time, DSDP generates a sequence of *corrector* steps. The corrector steps compute $\mathcal{A}S^{-1}$ using the current S and $\Delta^c y := M^{-1}\mathcal{A}S^{-1}$. Since the computation of M used previous values of S, the corrector step

$$\Delta^c_\nu y := \frac{1}{\nu}\Delta' y - \Delta^c y$$

is not a Newton step. A new barrier parameter

$$\nu := \left(\frac{b^T \Delta' y}{b^T \Delta^c y}\right)\left(\frac{n}{n+\sqrt{n}}\right)$$

applies when the numerator and denominator of the first term are positive. The first term in this heuristic is the value such that $b^T \Delta^c_\nu y = 0$ and the second term slightly reduces the parameter. A line search computes a step length, α_c, that improves the merit function

$$\phi_\nu(y) := b^T y + \nu \ln\det S, \tag{11}$$

whose gradient differs from (9) by a factor of $-\nu$. Between 0 and 12 corrector steps are applied each iteration. The exact number is chosen heuristically based on the square of the ratio of m and the dimension of the largest semidefinite block. Fewer corrector steps may be used if the steplength falls below a tolerance or the relative duality gap is less than 10^{-5}. This choice of corrector steps is different from a corrector step that solves $M\Delta^c y = \mathcal{A}\left(S^{-1}\Delta S S^{-1}\Delta S S^{-1}\right)$ and takes a single step.

However, the step in DSDP is cheaper to compute, and computational tests verify that it works well.

The primal, dual, and corrector steps are designed to reduce $\mu(X, S)$ and the dual potential function at each iteration. As a safeguard, the last step of the algorithm checks that $\nu \leq (\bar{z} - b^T y)/n$. The barrier parameter at the beginning of each iteration should decrease monotonically, but the first $X(\nu) \in \hat{K}$ may increase the initial $\bar{z}$ and increase the barrier parameter. Computational experience confirms that since the initial upper bound and barrier parameter is arbitrary, resetting these parameters based on a feasible $X(\nu)$ improves performance.

As the algorithm converges, numerical difficulties will eventually inhibit the accurate computation of the Newton step and prevent further progress. Although DSDP lets users terminate the solver through tolerances on the relative duality gap, DSDP will also terminate if the step sizes are too short or consecutive iterations show no progress in the solution.

Although DSDP does not compute $X(\nu)$ at each iteration, it can compute it using ν, y, and $\Delta_\nu y$. The following algorithm for computing $X(\nu)$ is typically used after the solver converges.

Compute X:
Set $S \leftarrow C - \mathcal{A}^* y$ and invert it.
Set $X \in \mathbb{S}^n$ to zero
for $i \leftarrow 1 : m$ **do**
  for $r \leftarrow 1 : rank(A_i)$ **do**
    $w \leftarrow S^{-1} a_{i,r}$
    $X \leftarrow X + (\nu \Delta_\nu y_i \alpha_{i,r}) w w^T$
  end for
end for
$X \leftarrow X + \nu S^{-1}$

This technique is numerically stable and yields high accuracy in the solution. Afterward, DSDP will add a small multiple of the identity matrix to this solution until a Cholesky factorization verifies that it is positive definite.

4. BRIEF NOTE ON CONVERGENCE

If we define $P(\nu)$ and $X(\nu)$ as in (5) and (10), the following two lemmas from [Benson et al. 2000] provide some insight about convergence of the algorithm.

LEMMA 1. *Let* $\mu^k = \frac{\bar{z} - b^T y}{n}$, $\mu = \frac{\langle C, X(\nu) \rangle - b^T y}{n}$, $\nu < \frac{\bar{z} - b^T y}{n + \sqrt{n}}$, *and* $\alpha < 1$. *If*

$$\|P(\nu)\| < min(\alpha \sqrt{\frac{n}{n + \alpha^2}}, 1 - \alpha),$$

then the following three inequalities hold:

(1) $X(\nu) \in \hat{K}$;
(2) $\|S^{.5} X(\nu) S^{.5} - \mu I\| \leq \alpha \mu$;
(3) $\mu \leq (1 - .5\alpha/\sqrt{n})\mu^k$.

LEMMA 2. *Let* $\nu < \frac{\bar{z} - b^T y}{n + \sqrt{n}}$, *and* $\alpha < 1$. *If* $y^+ = y + \frac{\alpha}{\|P(\nu)\|} \Delta_\nu y$, *then the following two inequalities hold:*

```
Setup data structures and factor A_i.
Choose y such that S ← C − A*y ∈ K̂.
Choose an upper bound z̄ and a barrier parameter ν.
for k ← 0, ..., k_max do
   Monitor solution and check for convergence.
   Compute M and AS^{-1}.
   Solve MΔ'y = b, MΔ''y = AS^{-1}.
   if C − A*(y − Δ_ν y) ∈ K̂ then
      z̄ ← b^T y + ν (Δ_ν y^T AS^{-1} + n).
      ȳ ← y, Δ̄y ← Δ_ν y, μ̄ ← ν.
   end if
   Select ν.
   Find α_d to reduce ψ, and set y ← y + α_d Δ_ν y, S ← C − A*y.
   for kk = 1, ..., kk_max do
      Compute AS^{-1}.
      Solve MΔ^c y = AS^{-1}.
      Select ν.
      Find α_c to reduce φ_ν, and set y ← y + α_c Δ^c_ν y, S ← C − A*y.
   end for
end for
Optional: Compute X using ȳ, Δ̄y, μ̄.
```

(1) $y^+ \in \hat{K}$;

(2) $\nabla\psi^T(y^+ - y) = -\alpha\|P(\nu)\|$.

In other words, when $\|P(\nu)\|$ is small, the dual-scaling algorithm generates an improved X, and when $\|P(\nu)\|$ is large, the new points (y, S) reduce the dual potential function. Either (y, S) or X reduces the Tanabe-Todd-Ye primal-dual potential function

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

enough at each iteration to achieve linear convergence. When $\rho > n$, the infimum of the potential function occurs at an optimal solution; and when $\rho > n + \sqrt{n}$, convergence in the worst case is bounded above by $\mathcal{O}\left(\sqrt{n}\log(\epsilon)\right)$ iterations, where n is the dimension of the cone and $\epsilon > 0$ is the desired fractional reduction in the duality gap.

5. FEASIBLE POINTS, INFEASIBLE POINTS, AND STANDARD FORM

The convergence of the algorithm assumes that both (P) and (D) have an interior feasible region and the current solutions are elements of the interior. To satisfy these assumptions, DSDP bounds the variables y such that $l \le y \le u$ where $l, u \in \mathbb{R}^m$. By default, $l_i = -10^7$ and $u_i = 10^7$ for each i from 1 through m. Furthermore, DSDP bounds the trace of X by a penalty parameter Γ whose default value is $\Gamma = 10^{10}$. Including these bounds and their associated Lagrange variables $x^l \in \mathbb{R}^m$, $x^u \in \mathbb{R}^m$, and r, DSDP solves following pair of problems:

$$
\begin{array}{lllllllll}
(PP) & \text{minimize} & \langle C, X\rangle & + & u^T x^u & - & l^T x^l & & \\
 & \text{subject to} & \mathcal{A}X & + & x^u & - & x^l & = & b, \\
 & & \langle I, X\rangle & & & & & \leq & \Gamma, \\
 & & X \in K, & & x^u \geq 0, & & x^l \geq 0. & &
\end{array}
$$

$$
\begin{array}{lllll}
(DD) & \text{maximize} & b^T y - \Gamma r & & \\
 & \text{subject to} & C - \mathcal{A}^* y + Ir & = & S \in K, \\
 & & l \leq y \leq u, & & r \geq 0.
\end{array}
$$

The reformulations (PP) and (DD) are bounded and feasible, so the optimal objective values to this pair of problems are equal. Furthermore, (PP) and (DD) can be expressed in the form of (P) and (D).

Unless the user provides a feasible point y, DSDP uses the y values provided by the application (usually all zeros) and increases r until $C - \mathcal{A}^* y + Ir \in \hat{K}$. Large values of r improve robustness, but smaller values often improve performance. In addition to bounding X, the parameter Γ penalizes infeasiblity in (D) and forces r toward zero. The nonnegative variable r increases the dimension m by one and adds an inequality to the original problem. The M matrix treats r separately by storing the corresponding row/column as a separate vector and applying the Sherman-Morrison-Woodbury formula. Unlike other inequalities, DSDP allows r to reach the boundary of the cone. Once $r = 0$, it is fixed and effectively removed from the problem.

The bounds on y add $2m$ inequality constraints to the original problem; and, with a single exception, DSDP treats them the same as the constraints on the original model. The lone difference between these bounds and the other constraints is that DSDP explicitly computes the corresponding Lagrangian variables x^l and x^u at each iteration to quantify the infeasibility in (P). The bounds l and u penalize infeasiblity in (P), force x^l and x^u toward zero, and prevent numerical difficulties created by variables with large magnitude.

The solution to (PP) and (DD) is a solution to (P) and (D) when the optimal objective values of (P) and (D) exist and are equal, and the bounds are sufficiently large. DSDP identifies unboundedness or infeasibility in (P) and (D) through examination of the solutions to (PP) and (DD). Given parameters ϵ_P and ϵ_D,

—if $r \leq \epsilon_r$, $\|\mathcal{A}X - b\|_\infty / \langle I, X\rangle > \epsilon_P$, and $b^T y > 0$, it characterizes (D) as unbounded and (P) as infeasible;

—if $r > \epsilon_r$ and $\|\mathcal{A}X - b\|_\infty / \langle I, X\rangle \leq \epsilon_P$, it characterizes (D) as infeasible and (P) as unbounded.

Normalizing unbounded solutions will provide an approximate certificate of infeasibility. Larger bounds may improve the quality of the certificate of infeasibility and permit additional feasible solutions, but they may also create numerical difficulties in the solver.

6. THE SOFTWARE

The DSDP software package implements the dual-scaling method using ANSI C. It is designed as a subroutine library that requires users to call a sequence of subroutines

that create a solver structure, set the data into the structure, apply the dual-scaling algorithm, and retrieve the solution from the solver. The use of these subroutines is documented in a user manual [Benson and Ye 2004], HTML pages created by Doxygen, and several examples.

One example is a *mex* function that calls DSDP from the Matlab environment. Most users of DSDP use the solver from Matlab, so several Matlab examples and a separate user guide are provided in the distribution.

Another example reads input files in SDPA format and prints the output to solution files. Since other SDP solvers also accept data in this format, its use is particularly convenient for those not interested in learning the syntax of a specific solver. Other examples provided in the distribution read a graph from a file, formulate the semidefinite relaxation of a combinatorial optimization problem, and call the DSDP solver.

DSDP has been compiled successfully by using GCC, Intel, Microsoft, and Solaris compilers. It must also be linked to an implementation of the BLAS and LAPACK. Reference Fortran, Atlas, and MKL implementations of these libraries have all been linked successfully.

6.1 Data Structures

The DSDP solver computes $\Delta' y$, $\Delta'' y$, $\Delta_\nu y$, $\|P(\nu)\|$, $\langle X(\nu), S\rangle$, and other quantities using operations on vectors, the Schur matrix, and the cones. Vector operations include sums and inner products. Operations on the Schur matrix include inserting elements and factoring the matrix. Objects representing a cone implement routines for computing its dual matrix S from y, evaluating the logarithmic barrier function, computing $\mathcal{A}S^{-1}$, and computing M. The solver object computes M, for example, by calling the corresponding operations on each cone and summing the results. The solver computes $\Delta' y$ through calls to the Schur matrix object that can factor the matrix and solve systems of linear equations.

The LP cone and SDP cone objects implement the same interface for cones, but the implementation of these operations depends on whether the cone is a semidefinite cone, linear programming cone, or another type. The solver structure operates on the cone objects without knowing whether it is an SDP cone or another type. DSDP uses opaque pointers and function pointers to achieve polymorphic behavior.

The semidefinite cone uses a dense array, V, to represent $S^{-1}A_iS^{-1}$, $X(\nu)$, and arbitrary symmetric matrices. By default, the array has $n(n+1)/2$ elements ordered

$$[\ a_{1,1}\ \ a_{2,1}\ \ a_{2,2}\ \ a_{3,1}\ \ a_{3,2}\ \ a_{3,3}\ \ \ldots\ \ a_{n,n}\ ]. \tag{12}$$

This format is also known as packed symmetric format. Data matrices sum into this array and take inner products with this array. Sparse, dense, and low rank data structures represent the data. From the dense array, the S and ΔS matrix insert nonzeros into their data structures. Although packed symmetric sparse formats have been implemented for S, DSDP usually uses the "upper" half of a $n \times n$ dense array because the Atlas BLAS provide good performance on this format. DSDP also supports the matrix V in full format with $n \times n$ array elements, and coordinates the different representation with the other data structures. Many problems have multiple semidefinite blocks. These blocks may be placed in separate semidefinite cones, but it is often more efficient to couple the blocks together in a single semidefinite

Table II. Summary of primary data structures and their functionality.

Solver: Implements an algorithm for linear and semidefinite programming. *Operations:* $\Delta' y$, $\Delta'' y$, $\Delta_\nu y$, $\Delta^c y$, $\|P(\nu)\|$, $\langle X(\nu), S\rangle$, reduce ν. *Implementations:* Dual-Scaling Algorithm. *Instances:* one. *Requires:* Vector, Cone, Schur.
Vector: Represents y, $\mathcal{A}S^{-1}$, b, $\Delta' y$, $\Delta'' y$, $\Delta_\nu y$, and other internal work vectors. *Operations:* sum, inner product, norm. *Implementations:* dense one-dimensional array. *Instances:* about two dozen.
Schur: Represents M, the Schur complement of Newton equations. *Operations:* add to row, add to diagonal, factor, solve, vector-multiply. *Implementations:* sparse, dense, parallel dense. *Instances:* one.
Cone: Represents data C and A_i. *Operations:* check if $S \leftarrow C - \mathcal{A}^* y \in K$, $\ln\det S$, $\mathcal{A}S^{-1}$, M, $X(\nu)$. *Implementations:* SDP Cone, LP Cone, Bounds on y, Variable $r \geq 0$. *Instances:* three or more. *Requires:* Vector, Schur. *SDP Cone requires:* SDP V Matrix, SDP Data Matrices, SDP S Matrix, SDP DS Matrix.
SDP V Matrix: Represents X, $S^{-1}A_i S^{-1}$, and a buffer for $C - \mathcal{A}^* y$. *Operations:* $V \leftarrow 0$, $V \leftarrow V + \gamma w w^T$, get array. *Implementations:* dense. *Instances:* one per block.
SDP Data Matrix: Represents a symmetric data matrix. *Operations:* $V \leftarrow V + \gamma A$, $\langle V, A\rangle$, $w^T A w$, get rank, get eigenvalue/vector. *Implementations:* sparse, dense, identity, low-rank. *Instances:* up to $m+1$ per block.
SDP S Matrix: Represents S and checks whether $X(\nu) \succ 0$. *Operations:* $S \leftarrow V$, Cholesky factor, forward solve, backward solve, invert. *Implementations:* sparse, dense. *Instances:* two per block.
SDP DS Matrix: Represents ΔS. *Operations:* $\Delta S \leftarrow V$, $w \leftarrow \Delta S v$. *Implementations:* dense, sparse, diagonal. *Instances:* one per block.

cone and specify the block structure.

Since M is symmetric, it may store only the lower half matrix, upper half matrix, or half of some permutation of the matrix. Each of these representations for M has been implemented in DSDP, and the cones work with each representation. The operations implemented for M allow cones to query it for the required indices and add the corresponding elements into the matrix. This interface is the basis for the using DSDP in parallel [Benson 2003], which distributes M and the computation of M over multiple processors.

Table II shows eight of the primary data structures used in DSDP, the operations they implement, and the other objects required to implement those operations. For dense matrix structures, DSDP uses BLAS and LAPACK to operate on the data.

6.2 Parameters

Table III summarizes the significant parameters, options, and default values. An asterisk (*) indicates that significant improvements in performance can often be

Table III. Summary of important parameters and initial values.

r: Dual infeasibility. *Default:* heuristic (large) *Suggested Values:* $10^2 - 10^{12}$ **Comments:* Larger values ensure robustness, but smaller values can signficantly improve performance.
$\mathbf{y}$: Initial solution. *Default:* 0 *Suggested Values:* Depends on data *Comments:* Initial points that improve performance can be difficult to find.
ρ_n: Bound ρ above by $n \times \rho_n$ and influence the barrier parameter. *Default:* 3.0 *Suggested Values:* $2.0 - 5.0$ *Comments:* Smaller values ensure robustness, but larger values can significantly improve performance.
kk_{max} : Maximum number of corrector steps. *Default:* 4 *Suggested Values:* $0 - 15$ **Comments:* For relatively small block sizes, increase this parameter.
Γ: The penalty parameter r and the bound on the trace of X. *Default:* 1e10. *Suggested Values:* $10^3 - 10^{15}$ *Comments:* Other values can improve performance.
l,u: Bounds on the variables y. *Default:* $-10^7, 10^7$ *Suggested Values:* Depends on the data. *Comments:* Tighter bounds do not necessarily improve performance.
$\bar{z}$: Upper bound on (D). *Default:* 10^{10} *Suggested Values:* Depends on the data. *Comments:* A high bound is usually sufficient.
ν: Dual barrier parameter. *Default:* Heuristic *Suggested Values:* Depends on the current solution. *Comments:* The default method sets $\nu = (\bar{z} - b^T y)/\rho$.
k_{max} : Maximum number of dual-scaling iterations. *Default:* 200 *Suggested Value:* $50 - 500$ *Comments:* Iteration counts of 20-60 are common.
η: Terminate when $(\bar{z} - b^T y)/(\lvert b^T y\rvert + 1)$ is less than η. *Default:* 10^{-6} *Suggested Values:* $10^{-2} - 10^{-6}$ *Comments:* Many problems do not require high accuracy.
ρ : Either a dynamic of a fixed value can be used. *Default:* Dynamic. *Suggested Values:* Dynamic *Comments:* The fixed strategy sets $\rho = n \times \rho_n$ and $\nu = (\bar{z} - b^T y)/\rho$, but its performance is usually inferior to the dynamic strategy.
ϵ_r, ϵ_P: Classify solutions as feasible. *Default:* 10^{-8}, 10^{-4} *Suggested Values:* $10^{-2} - 10^{-10}$ *Comments:* Adjust if the scaling of the problem is poor.

Table IV. Impact of parameters on the time and number of iterations.

Problem	Test 1 It. (Sec.)	Test 2 It. (Sec.)	Change
arch0	49(5)	36(3)	Change initial r from 10^7 to 10^2.
truss8	33(19)	21(12)	Change initial r from 10^6 to 10^2.
control10	31(114)	27(105)	Change bounds u and l from $\pm 10^7$ to ± 100.
qpG11	31(48)	27(43)	Change ρ from 3 to 5.
gpp500-1	42(23)	26(46)	Change number of corrector steps from 0 to 5.
vibra3	190(78)	92(74)	Change number of corrector steps from 0 to 8.

found by varying this parameter. The most important of these options is the initial variable r. By default, DSDP selects a value much larger than required to make $S \in \hat{K}$. Computational experience indicates that large values are more robust than smaller values. As Table IV shows, however, smaller values can significantly improve performance. DSDP then sets the initial values of $\bar{z} = 1e10$ and $\nu = (\bar{z} - b^T y + \Gamma r)/(n\rho_n)$. Users can manually set $\bar{z}$ and ν, but choices better than the defaults usually require insight into the solution. The number of corrector steps can also significantly improve performance. In some examples, corrector steps can reduce the number of iterations by half—although the impact in total computation time is not as significant. Computational experience suggests that the number of corrector steps should be between 0 and 12, and the time spent in these steps should not exceed 30%.

Instead of solving a problem with a single call to DSDP, users can apply a single iteration of the dual-scaling algorithm inside their own loop. By initializing the solver with the values of y, r, $\bar{z}$, and ν from the previous iteration, users can recreate the loop used by DSDP. This process may duplicate some computations that cause a loss in efficiency, but sequence of (X^k, y^k, S^k) will remain the same.

6.3 Event Profiling

Event profiling in DSDP helps users understand the dual-scaling algorithm, analyze the performance of the solver, and tune the parameters. The events listed in Table V are profiled by the software and consist of one or more subroutines. The columns left of the events indicate sets of mutually exclusive events. For each event, DSDP prints the number of times it was called, the time spent in it, and its percentage of the overall time. Only the most computationally dominant events are profiled, so the percentage of time in mutually exclusive events may not add to 100%.

Table VI shows some profiling data for three semidefinite programming problems from SDPLIB [Borchers 1999b]. In the `control` problem, the most computationally dominant event was computing M. The high percentage of time spent in `SDP Dot` indicates that DSDP used Technique M1 to compute M. Furthermore, a significant amount of time was spent factoring the data matrices. In the `theta` and `maxcut` problems, factoring M required more time than did any other event. Less time spent computing M and factoring the data reflects the simple structure of the data matrices. Both of these examples used Technique M2 to compute M, but the `maxcut` problem did not invert S as evidenced by the high percentage of time spent solving linear equations involving it. Furthermore, the `maxcut` problem did not use corrector steps, as evidenced by the absence of time spent in the corresponding events. The dimension of the semidefinite block in this example is the same as the dimension of M, so a corrector step would cost almost the same as a Newton step. About 20% of the time spent solving `maxcut` was spent computing $X(\nu)$ once at the end of the algorithm. The high cost illustrates the potential advantages of the dual-scaling algorithm over other interior-point algorithms when the block sizes are very large. In each of the three problems, the high percentage of time in factoring M and other dense array computations emphasizes the importance of linking to a fast implementation of BLAS and LAPACK.

Table V. Events profiled by DSDP and subsets that are mutually exclusive.

				Event and Description
•	•		•	`Cone Setup`: Compute eigenvalues and eigenvectors of $A_{i,j}$.
	•			`Cone Invert S`: Invert S from its factorization.
	•			`Cone RHS`: Compute $\mathcal{A}S^{-1}$.
	•		•	`Cone Compute Newton Eq.`: Compute M and $\mathcal{A}S^{-1}$.
	•			`Cone Max P Step Length`: Largest α_P such that $X \in K$.
	•			`Cone Compute and Factor SP`: Check if $SP \leftarrow C - \mathcal{A}^*(y - \alpha_P \Delta_\nu y) \in K$.
	•			`Cone Max D Step Length`: Largest α_D such that $S \in K$.
	•			`Cone Compute and Factor S`: Check if $S \leftarrow C - \mathcal{A}^*(y + \alpha_D \Delta_\nu y) \in K$.
	•			`Cone Potential`: Compute $\log\det S$ from the factorization of S.
•	•		•	`Cone Compute X`: Compute $X(\nu)$.
		•		`R Cone`: Cone operations for $r \geq 0$.
		•		`Bound Y Cone`: Cone operations for $l \leq y \leq u$.
		•		`LP Cone`: Cone operations for LP cone.
		•		`SDP Cone`: Cone operations for SDP cone.
				`SDP VecMatVec`: $v^T A v$
				`SDP SSolve`: Solve $Sv = a_i$ from its factorization.
				`SDP V+vv'`: $V \leftarrow V + \alpha v v^T$.
				`SDP Dot`: $\mathcal{A}V$.
	•	•	•	`Factor Newton Eq.`: Cholesky Factorization of M.
			•	`Direct Solve`: Direct solves using factorization of M.
	•	•		`CG Solve`: Compute Δy using CG.
			•	`Primal Step`: Compute α_D.
			•	`Dual Step`: Compute α_D and factor S.
			•	`Corrector Step`: Compute $\mathcal{A}S^{-1}$, Δy, and step length.
•				`DSDP Solve`: Apply dual-scaling algorithm to SDP.

6.4 Iteration Monitor

The progress of the DSDP solver can be monitored by using standard output printed to the screen. The data below shows an example of this output.

```
Iter  PP Objective     DD Objective    PInfeas  DInfeas    Nu     StepLength   Pnrm
-------------------------------------------------------------------------------------
0    1.00000000e+02  -1.13743137e+05   2.2e+00  3.8e+02  1.1e+05  0.00  0.00    0.00
1    1.36503342e+06  -6.65779055e+04   5.1e+00  2.2e+02  1.1e+04  1.00  0.33    4.06
2    1.36631922e+05  -6.21604409e+03   5.4e+00  1.9e+01  4.5e+02  1.00  1.00    7.85
3    5.45799174e+03  -3.18292092e+03   1.5e-03  9.1e+00  7.5e+01  1.00  1.00   17.63
4    1.02930559e+03  -5.39166166e+02   1.1e-05  5.3e-01  2.7e+01  1.00  1.00    7.58
5    4.30074471e+02  -3.02460061e+01   3.3e-09  0.0e+00  5.6e+00  1.00  1.00   11.36
...
11   8.99999824e+00   8.99999617e+00   1.1e-16  0.0e+00  1.7e-08  1.00  1.00    7.03
12   8.99999668e+00   8.99999629e+00   2.9e-19  0.0e+00  3.4e-09  1.00  1.00   14.19
```

The program will print a variety of statistics for each problem to the screen.

Table VI. Percentage of time in DSDP events for three examples.

Event	Control11	theta5	maxG11
Cone Setup	9	0	3
Cone Invert S	0	2	0
Cone RHS	1	0	-
Cone Compute Newton Eq.	57	31	40
Cone Max P Step Length	0	0	1
Cone Compute and Factor SP	0	0	2
Cone Max D Step Length	2	2	1
Cone Compute and Factor S	1	1	2
Cone Potential	0	0	0
Cone Compute X	0	1	20
R Cone	0	0	0
Bound Y Cone	0	0	0
SDP Cone	70	36	68
SDP VecMatVec	1	23	14
SDP Dot	40	0	0
SDP SSolve	7	1	20
SDP V+vv'	5	1	18
Factor Newton Eq.	25	46	25
Direct Solve	3	5	2
CG Solve	3	16	5
Primal Step	1	0	3
Dual Step	1	0	3
Corrector Step	6	17	-
DSDP Solve	90	99	77

`Iter`	the iteration number.
`PP Objective`	the upper bound $\bar{z}$ and objective value in (PP).
`DD Objective`	the objective value in (DD).
`PInfeas`	the primal infeasiblity in (P) is $\|x^u - x^l\|_\infty$.
`DInfeas`	the dual infeasibility in (D) is the variable r.
`Nu`	the barrier parameter ν.
`StepLength`	the multiple of the step-directions in (P) and (D).
`Pnrm`	the proximity to the central path: $\|\nabla\psi\|_{M^{-1}}$.

6.5 Performance

DSDP has been used in many applications such as minimum-error convex fitting [Roy et al. 2005], maximum likelihood decoding [Mobasher et al. 2005], frequency response functions [Rotea and D'Amato 2001], protein clustering [Lu et al. 2005], pattern analysis [Keuchel et al. 2003], binary imaging [Keuchel et al. 2001], wireless sensor network localization [Jin and Saunders 2005; Biswas and Ye 2004], Terwilleger algebras [Schrijver 2005], cell cycle regulatory genes [Bhargava and Kosaraju 2003], linear matrix inequalities [Löfberg 2001], and combinatorial optimization [ENSTA 2005].

Its performance has been benchmarked by Mittelmann [2005] relative to six other SDP solvers. His test include examples from SDPLIB [Borchers 1999b], DIMACS [Center for Discrete Mathematics and Theoretical Computer Science 2000], and his own collection of large sparse problems. His statistics are public and frequently

updated, so a snapshot of them will not be shown in this paper. They show, however, that DSDP is a competitive solver on a broad set of SDP problems. Of 108 tests, DSDP solved 102 and exhausted the 4 GB of memory on the other six problems. In most of the examples $\|\mathcal{A}X - b\|/\|b\| < 10^{-6}$, and the objective is correct to at least six digits of precision, although solutions whose norm is very large usually exhibited less precision.

On a subset of 25 problems, we evaluate the performance of DSDP, version 5.8, against five other "solvers": itself using no corrector steps (No Corrector), itself using no corrector steps and a simpler strategy for reducing the barrier parameter (NC & Fixed ρ), itself linked to reference implementation of BLAS and LAPACK instead of an Atlas implementation (Ref. BLAS), an earlier version of the solver (v. 4.7), and a competing interior-point solver for semidefinite programming (Other). Following the performance profiling methodology of [Dolan and Moré 2002], we identify the best of the six solvers for each of the 25 problems. Define $t_{p,s}$ as the computing time required to solve problem p by solver s, and let failure be denoted with a time greater than all other times. For each problem, compare the best solver to each of the solvers using the *performance ratio*

$$r_{r,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}$$

Figure 1 shows the percentage of problems for which each solver found a solution within a factor of τ (≥ 1) of the best solver. For $\tau = 1$, these profiles shows the percentage for which each each solver was the fastest. The right side of the graph shows the percentage problems for which each solver was successful. These tests used an Intel Pentium 4 CPU with 1.80 GHz, 256 KB cache, 512 MB RAM. DSDP was compiled using GCC 3.3 with O3 optimization, while the binary for competing solver was downloaded from the developer's website. Both solvers linked to the Atlas BLAS.

Figure 1 supports several conclusions. First, the speed and robustness of this version of DSDP is much superior to preliminary versions. Years of work and experience have been encapsulated into this piece of software. Some its improvement can be credited to the introduction of corrector steps and the use dense data structures exploited by LAPACK. The effects of these changes are shown in the figure. Other changes include the bounding of variables and better parameter choices have significantly improved robustness.

Additional conclusions concerns its performance relative to other solvers. DSDP was the fastest solver in more than half of the test problems, and within a factor of two of the best solver on over 80% of the problems. The other interior-point solver was the fastest on almost half of the problems, and within a factor of two on about 60% of the problems.

On some classes of problems, DSDP performs exceptionally well. In particular, problems with

—large semidefinite blocks,

—a sparse Cholesky factorization of S, or

—a low-rank constraint structure

especially benefit from the solver. The previous presentation of the dual-scaling

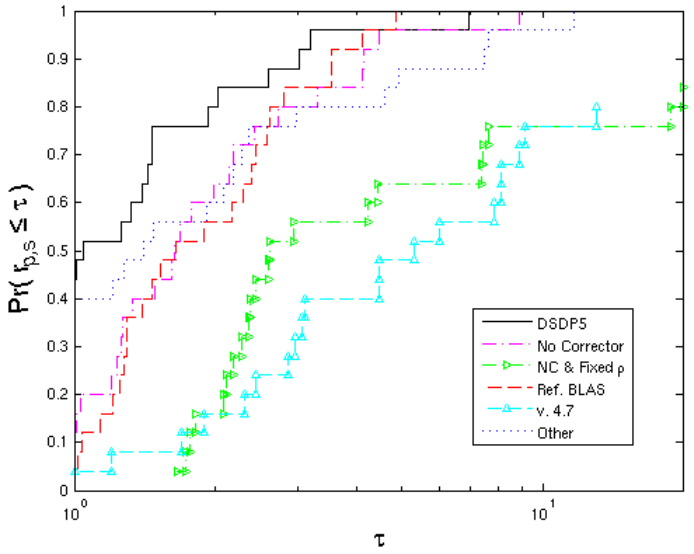

Fig. 1. Profiling the Performance of DSDP.

algorithm [Benson et al. 2000] explained why the algorithm is appropriate for the semidefinite relaxation of the max-cut problem. For these problems, the dimension of the semidefinite block equals the dimension of M, so the X variables are expensive to compute. By not maintaining these variables at each iteration, DSDP reduces the time and memory needed to solve these problems. Sparsity in the Cholesky factorization of S magnifies this advantage. Relaxations of hard combinatorial optimization problems and linear matrix inequalities arising in control theory exhibit low-rank structure in the constraint matrices. Both Technique M1 and Technique M2 take advantage of this structure, and DSDP performs well on these problems. Whereas the dual-scaling algorithm exploits the first two structural properties, our implementation of the algorithm exploits the third property.

For problems without a dual interior region, DSDP provides primal solutions near the bounds that suffer less precision. These problems arise, for instance, when an equality is split into two inequalities. The incorporation of equalities into the standard form (D) may be in future releases, but current models with many linear equations may find better performance in another solver or a reformulation of the model. Performance of the solver could also improve on examples where C equals zero and applications (see e.g. [Zhao et al. 2004]) with high-rank data matrices. Primal-dual algorithms in other solvers may enjoy an algorithmic advantage of the former class of problems, but their advantage on high-rank problems is primarily due to implementation choices of the developers.

Although the authors welcome use of DSDP, others may want to reimplement the dual-scaling algorithm. Valid reasons include a difference in design philosophy, licensing issues, and special structure in an application. The following list of suggestions may help write an efficient implementation. Some of these suggestions are

particular to the dual-scaling algorithm, while others generalize to interior-point methods, semidefinite programming, or numerical computing. These suggestions are probably known to experienced developers of numerical software, but they are worth repeating.

(1) *Bound y.* Some combinatorial problems, for instance, set $\langle 1, X \rangle = 0$ which permits no interior primal feasible point. The corresponding y variable grows very large and creates numerical difficulties. If $C = 0$, the dual feasible set is unbounded (or empty), no central path exists, and y grows very large. To prevent such problems, explicitly bound the variables.

(2) *Bound X.* Equalities expressed as a pair of inequalities in (D), for example, permit no interior dual feasible points and the norm of X grows very large. Bounding X allows the feasible-point algorithm to converge to an approximate solution according to the theory.

(3) *Large initial barrier parameter.* Some problems (trto, vibra, control, biggs) require a large initial r and a large initial barrier parameter. When these parameters are too small, many consecutive iterations can be characterized by $\|P(\nu)\| \sim 1.7$ and short primal steps that do not update the primal solution.

(4) *Reduce the barrier parameter.* Simple strategies may work well, but more complicated strategies significantly reduce the number of iterations for many problems.

(5) *Sparse data matrices.* Sparse versions of (12) usually permit efficient operations on the data.

(6) *Use LAPACK.* Dense matrix operations define much of the algorithm, so use dense data structures supported by fast implementations of LAPACK.

(7) *Use structure to compute M.* The high computational complexity of this event means there are many techniques compute it. Find a technique that exploits the structure in the problems of interest.

(8) *Solve M accurately.* Especially when y is large, it takes more than a factorization. Augmenting its diagonal elements and generating approximate step directions may suffice for the dual step, but the construction of a feasible X matrix requires accurate Newton step directions, so be careful.

(9) *Corrector step.* Corrector steps significantly reduce the number of iterations for some problems (truss, trto, vibra, buck), but they are expensive when the block sizes are large (gpp, maxcut, shmup).

(10) *Use the potential function.* Many interior-point methods simply take a fraction of the distance to the boundary of the cone, but this method is more robust when the line search also reduces the potential function.

In short, the formulation of the model matters, the choice of data structures is important, and the theory is very relevant.

7. EXTENSIONS

The use of function pointers and opaque structures enables polymorphic behavior in DSDP. More specifically, DSDP accepts data matrices that implement its interface but are not included in the distribution. Some combinatorial problems, for

example, have a data matrix that such that every element is 1. Data structures that individually represent each element may suffice, but other data structures may operate on the data more quickly and use computer memory more efficiently. An example of this extension is included in the distribution. Similar facilities exist for the future support of second-order cones and structured semidefinite cones for specific applications.

8. CONCLUSIONS

This version of DSDP was written to demonstrate the competitiveness of the dual-scaling algorithm for semidefinite programming, maintain exceptional performance on several classes of semidefinite programs, and provide computational scientists with a robust, efficient, and well-documented solver for their applications. Emphasis is given to semidefinite programming, but the solver is designed to allow its use for linear programming problems and extensions to the second-order cone and other structured cones. The software is freely available from the Mathematics and Computer Science Division at Argonne National Laboratory and the authors encourage its use with the terms the license.

Acknowledgments

We thank Xiong Zhang and Cris Choi for their help in developing this code. Xiong Zhang, in particular, was fundamental to the initial version of DSDP. We also thank Hans Mittelmann[Mittelmann 2005] for his efforts in testing and benchmarking the different versions of the code. Finally, we thank Johan Löfberg, Stefan Ratschan, and all of the users who have commented on previous releases and suggested improvements to the software. Their contributions have made DSDP a more reliable, robust, and efficient package.

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

REFERENCES

Anjos, M. F. 2005. An improved semidefinite programming relaxation for the satisfiability problem. *Math. Program. 102,* 3, 589–608.

Benson, S. J. 2003. Parallel computing on semidefinite programs. Tech. Rep. ANL/MCS-P939-0302, Mathematics and Computer Science Division, Argonne National Laboratory. March.

Benson, S. J. and Ye, Y. 2004. DSDP5 user guide – the dual-scaling algorithm for semidefinite programming. `http://www.mcs.anl.gov/~benson/dsdp`.

Benson, S. J., Ye, Y., and Zhang, X. 1999. Mixed linear and semidefinite programming for combinatorial and quadratic optimization. *Optimization Methods and Software 11*, 515–544.

Benson, S. J., Ye, Y., and Zhang, X. 2000. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization 10,* 2, 443–461.

Bhargava, A. and Kosaraju, S. 2003. Identifying cell cycle regulatory genes. `http://www.cs.jhu.edu/~ankur/`.

Biswas, P. and Ye, Y. 2004. Semidefinite programming for ad hoc wireless sensor network localization. In *3rd International Symposium on Information Processing in Sensor Networks (IPSN).*

Borchers, B. 1999a. CSDP 2.3 user's guide. *Optimization Methods and Software 11/12,* 1-4, 597–611.

Borchers, B. 1999b. SDPLIB 1.2, a library of semidefinite programming test problems. *Optimization Methods and Software 11*, 683–690.

Burer, S. and Monteiro, R. 2003. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (series B) 95,* 2, 329–357.

Center for Discrete Mathematics and Theoretical Computer Science. 2000. Seventh DIMACS implementation challenge: Semidefinite and related optimization problems. `http://dimacs.rutgers.edu/Challenges/Seventh/`.

Dolan, E. D. and Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming 91*, 201–213.

ENSTA. 2005. Optimisation combinatoire and recherche opérationnelle. `www.ensta.fr/~diam/ao303/index.php`.

Fujisawa, K., Kojima, M., and Nakata, K. 1997. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming B79*, 235–253.

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

Helmberg, C. and Rendl, F. 2000. A spectral bundle method for semidefinite programming. *SIAM Journal of Optimization 10,* 3, 673 – 696.

Henrion, D. and Lasserre, J. B. 2003. GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi. *ACM Transactions on Mathematical Software 29,* 2 (June), 165–194.

Jin, H. and Saunders, M. 2005. A distributed algorithm for sensor localization. Tech. rep., Stanford University.

Keuchel, J., Schellewald, C., Cremers, D., and Schnörr, C. 2001. Convex relaxations for binary image partitioning and perceptual grouping. In *Pattern Recognition (23rd DAGM Symposium, Munich), Lecture Notes in Computer Science*, S. F. e. B. Radig, Ed. Vol. 2191. Springer, Berlin, 353–360. Awarded a prize of the German Pattern Recognition Society (DAGM).

Keuchel, J., Schnörr, C., Schellewald, C., and Cremers, D. 2003. Binary partitioning, perceptual grouping, and restoration with semidefinite programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25,* 11 (Nov.), 1364–1379. Special Issue on Energy Minimization Methods in Computer Vision and Pattern Recognition.

Kocvara, M. and Stingl, M. 2003. PENNON - a code for convex nonlinear and semidefinite programming. *Optimization Methods and Software 18,* 3, 317–333.

Löfberg, J. 2001. *YALMIP, Yet another LMI parser.* University of Linköping, Sweden. Available at `http://www.control.isy.liu.se/~johanl`.

Lu, F., Keles, S., Wright, S., and Wahba, G. 2005. A framework for kernel regularization with application to protein clustering. Tech. Rep. 1107, Department of Statistics, University of Wisconsin-Madison. May.

Mittelmann, H. D. 2005. Benchmarks for optimization software. `ftp://plato.la.asu.edu/pub/{sdplib.txt,sparse_sdp.txt,dimacs.txt}`.

Mobasher, A., Taherzadeh, M., Sotirov, R., and Khandani, A. K. 2005. A near maximum likelihood decoding algorithm for MIMO systems based on semi-definite programming. Tech. Rep. UW-E&CE#2005-12, University of Waterloo.

Prajna, S., Papachristodoulou, A., Seiler, P., and Parrilo, P. A. 2004. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB.*

Rotea, M. A. and D'Amato, F. J. 2001. LFTB: An optimized algorithm to bound worst-case frequency response functions. In *Proceedings of the 2001 American Control Conference.* Washington, D.C., 3041 – 3048.

Roy, S., Chen, W., and Chen, C. C.-P. 2005. ConvexFit: An optimal minimum-error convex fitting and smoothing algorithm with application to gate-sizing. In *International Conference on Computer Aided Design (ICCAD).* San Jose, California.

Schrijver, A. 2005. New code upper bounds from the Terwilliger algebra and semidefinite programming. *IEEE Transactions on Information Theory 51*, 2859–2866.

So, A. and Ye, Y. 2005. Theory of semidefinite programming for sensor network localization. In *Proceedings of SODA.* To appear in Mathematical Programming.

Sturm, J. F. 1999. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software 11/12,* 1-4, 625–653.

Todd, M. J. 2001. *Semidefinite Optimization.* Vol. 10. Cambridge University Press, Cambridge, 515–560.

Toh, K., Todd, M., and Tutuncu, R. 1999. SDPT3 — A Matlab software package for semidefinite programming, version 2.1. *Optimization Methods and Software 11*, 545–581.

Toh, K. C. 2002. A note on the calculation of step-lengths in interior-point methods for semidefinite programming. *Computational Optimization and Applications 21*, 301–310.

Wolkowicz, H., Saigal, R., and Vandenberghe, L., Eds. 2000. *Handbook of Semidefinite Programming.* International Series in Operations Research and Management Science, vol. 27. Kluwer.

Yamashita, M., Fujisawa, K., and Kojima, M. 2003. Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optimization Methods and Software 18*, 491–505.

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

Yildirim, E. A. and Fan, X. 2005. On extracting maximum stable sets in perfect graphs using lovasz's theta function. *Computational Optimization and Applications 32*, 1–30.

Zhao, Z., Braams, B. J., Fukuda, M., Overton, M. L., and Percus, J. K. 2004. The reduced density matrix method for electronic structure calculations and the role of three-index representability. *Journal of Chemical Physics 120*, 2095–2104.

Zowe, J., Kočvara, M., and Bendsøe, M. 1997. Free material optimization via mathematical programming. *Mathematical Programming 79*, 445–466.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.