DSDP5 User Guide — Software For Semidefinite Programming

Steven J. Benson
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL U.S.A.
`http://www.mcs.anl.gov/~benson`

Yinyu Ye
Department of Management Science and Engineering
Stanford University
Stanford, CA U.S.A
`http://www.stanford.edu/~yyye`

Technical Report ANL/MCS-TM-277

September 21, 2005

Abstract

DSDP implements the dual-scaling algorithm for semidefinite programming. The source code if this interior-point solver, 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 files. Initiated in 1997, DSDP has developed into an efficient and robust general purpose solver for semidefinite programming. Although the solver is written with semidefinite programming in mind, it can also be used for linear programming and other constraint cones.

The features of DSDP include:

- a robust algorithm with a convergence proof and polynomially bounded complexity under mild assumptions on the data,
- primal and dual solutions,
- feasible solutions when they exist or approximate certificates of infeasibity,
- 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. We welcome and encourage further use under the terms of the license included in the distribution.

1 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

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

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

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 1, 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

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

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 1: 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	$\sum \ln s_j$	$\ln \det S$	$\sum \ln \det S_j$	$\ln \det S$

2 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

$$\begin{aligned} \mathcal{A}(X + \Delta X) &= b, & (1)\\ \mathcal{A}^*(\Delta y) + \Delta S &= 0, & (2)\\ \nu S^{-1} \Delta S S^{-1} + \Delta X &= \nu S^{-1} - X. & (3) \end{aligned}$$

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.

Either (y, S) or X reduces the the dual potential function

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

enough at each iteration to achieve linear convergence.

```
 1: Setup data structures and factor $A_i$.
 2: Choose $y$ such that $S \leftarrow C - \mathcal{A}^* y \in \hat{K}$.
 3: Choose an upper bound $\bar{z}$ and a barrier parameter $\nu$.
 4: for $k \leftarrow 0, \ldots, k_{max}$ do
 5:    Monitor solution and check for convergence.
 6:    Compute $M$ and $\mathcal{A}S^{-1}$.
 7:    Solve $M\Delta' y = b$, $M\Delta'' y = \mathcal{A}S^{-1}$.
 8:    if $C - \mathcal{A}^*(y - \Delta_\nu y) \in \hat{K}$ then
 9:       $\bar{z} \leftarrow b^T y + \nu \left(\Delta_\nu y^T \mathcal{A}S^{-1} + n\right)$.
10:       $\overline{y} \leftarrow y$, $\overline{\Delta y} \leftarrow \Delta_\nu y$, $\overline{\mu} \leftarrow \nu$.
11:    end if
12:    Select $\nu$.
13:    Find $\alpha_d$ to reduce $\psi$, and set $y \leftarrow y + \alpha_d \Delta_\nu y$, $S \leftarrow C - \mathcal{A}^* y$.
14:    for $kk = 1, \ldots, kk_{max}$ do
15:       Compute $\mathcal{A}S^{-1}$.
16:       Solve $M\Delta^c y = \mathcal{A}S^{-1}$.
17:       Select $\nu$.
18:       Find $\alpha_c$ to reduce $\phi_\nu$, and set $y \leftarrow y + \alpha_c \Delta^c_\nu y$, $S \leftarrow C - \mathcal{A}^* y$.
19:    end for
20: end for
21: Optional: Compute $X$ using $\overline{y}$, $\overline{\Delta y}$, $\overline{\mu}$.
```

3 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 $l \leq y \leq u$ 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:

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

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

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.

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   2.9-19  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.

`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: $\|\nabla\psi\|_{M^{-1}}$.

5 DSDP with MATLAB

Additional help using the `DSDP` can be found by typing `help dsdp` in the directory `DSDP5.X`. The command

```
> [STAT, y, X] = dsdp(b, AC)
```

attempts to solve the semidefinite program by using a dual-scaling algorithm. The first argument is the objective vector b in (D) and the second argument is a cell array that contains the structure and data for the constraint cones. Most data has a block structure, which should be specified by the user in the second argument. For a problem with p cones of constraints, `AC` is a $p \times 3$ cell array. Each row of the cell array describes a cone. The first element in each row of the cell array is a string that identifies the type of cone. The second element of the cell array specifies the dimension of the cone, and the third element contains the cone data.

5.1 Semidefinite Cones

If the *jth* cone is a semidefinite cone consisting of a single block with n rows and columns in the matrices, then the first element in this row of the cell array is the string `'SDP'` and the second element is the number `n`. The third element in this row of the cell array is a sparse matrix with $n(n+1)/2$ rows and $m+1$ columns. Columns 1 to m of this matrix represent the constraints $A_{1,j}, \ldots, A_{m,j}$ for this block and column $m+1$ represents C_j.

The square symmetric data matrices $A_{i,j}$ and C_j map to the columns of $\texttt{AC\{j,3\}}$ through the operator $\mathbf{dvec}(\cdot) : \mathbb{R}^{n \times n} \to \mathbb{R}^{n(n+1)/2}$, which is defined as

$$\mathbf{dvec}(A) \quad = [a_{1,1} \quad a_{1,2} \quad a_{2,2} \quad a_{1,3} \quad a_{2,3} \quad a_{3,3} \quad \ldots \quad a_{n,n}]^T.$$

In this definition, $a_{k,l}$ is the element in row k and column l of A. This ordering is often referred to as symmetric packed storage format. The inverse of `dvec( )` is $\mathbf{dmat}(\cdot) : \mathbb{R}^{\mathbf{n(n+1)/2}} \to \mathbb{R}^{\mathbf{n \times n}}$, which converts the vector into a square symmetric matrix. Using these operations,

$$A_{i,j} = \texttt{dmat}(\texttt{AC\{j,3\}}(:,\texttt{i})), \qquad C_j = \texttt{dmat}(\texttt{AC\{j,3\}}(:,\texttt{m}+1))$$

and

$$\texttt{AC\{j,3\}} = [\ \texttt{dvec}(\texttt{A}_{1,j}) \quad \ldots \quad \texttt{dvec}(\texttt{A}_{m,j}) \quad \texttt{dvec}(\texttt{C}_j)\ ];$$

For example, the problem:

$$\begin{array}{lccccc}
\text{Maximize} & & y_1 & + & & y_2 \\
\text{Subject to} & \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & y_1 & + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} & y_2 & \preceq \begin{bmatrix} 4 & -1 \\ -1 & 5 \end{bmatrix}
\end{array}$$

can be solved by:

```
> b = [ 1 1 ]';
> AAC = [ [ 1.0 0 0 ]' [ 0 0 1.0 ]' [ 4.0 -1.0 5.0 ]' ];
> AC{1,1} = 'SDP';
> AC{1,2} = [2];
> AC{1,3} = sparse(AAC);
> [STAT,y,X]=dsdp(b,AC);
> XX=dmat(X{1});
```

The solution y is the column vector `y' = [ 3 4 ]'` and the solution X is a $p \times 1$ cell array. In this case, `X = [ 3 x 1  double ]`, `X{1}'=[ 1.0 1.0 1.0 ]`, and `dmat(X{1})=[ 1 1 ; 1 1 ]`.

Each semidefinite block can be stated in a separate row of the cell array; only the available memory on the machine limits the number of cones that can be specified.

Each semidefinite block may, however, be grouped into a single row in the cell array. To group these blocks together, the second cell entry must be an array of integers stating the dimension of each block. The data from the blocks should be concatenated such that the number of rows in the data matrix increases whereas the number of columns remains constant. The following lines indicate how to group the semidefinite blocks in rows 1 and 2 of cell array `AC1` into a new cell array `AC2`

```
> AC2{1,1} = 'SDP';
> AC2{1,2} = [AC1{1,2}  AC1{2,2}];
> AC2{1,3} = [AC1{1,3}; AC1{2,3}]
```

The new cell array `AC2` can be passed directly into `DSDP`. The advantage of grouping multiple blocks together is that it uses less memory – especially when there are many blocks and many of the matrices in these blocks are zero. The performance of `DSDP` measured by execution time, will change very little.

This distribution contains several examples files in SDPA format. A utility routine called `readsdpa( · )` can read these files and put the problems in `DSDP` format. They may serve as examples on how to format an application for use by the `DSDP` solver. Another example can be seen in the file `maxcut( · )` , which takes a graph and creates an SDP relaxation of the maximum cut problem from a graph.

5.2 LP Cones

A cone of LP variables can specified separately. For example a randomly generated LP cone $A^T y \leq c$ with 3 variables y and 5 inequality constraints can be specified in the following code.

```
> n=5; m=3;
> b = rand(m,1);
> At=rand(n,m);
> c=rand(n,1);
> AC{1,1} = 'LP';
> AC{1,2} = n;
> AC{1,3} = sparse([At c]);
> [STAT,y,X]=dsdp(b,AC);
```

Multiple cones of LP variables may be passed into the `DSDP` solver, but for efficiency reasons, it is best to group them all together. This cone may also be passed to the `DSDP` solver as a semidefinite cone, where the matrices A_i and C are diagonal. For efficiency reasons, however, it is best to identify them separately as belonging the the cone of 'LP' variables.

Although y variables that are fixed to a constant can be preprocessed and removed from a model, it is often more convenient to leave them in the model. It is more efficient for to identify fixed variables to `DSDP` than to model these constraints as a pair of linear inequalities. The following example sets variables 1 and 8 to the values 2.4 and -6.1, respectively.

```
> AC{j,1} = 'FIXED'; AC{j,2} = [ 1 8 ]; AC{j,3} = [ 2.4 -6.1 ];
```

The corresponding variables x to these constraints may be positive or negative.

5.3 Solver Options

There are more ways to call the solver. The command

```
> [STAT,y,X] = DSDP(b,AC,OPTIONS)
```

specifies some options for the solver. The `OPTIONS` structure may contain any of the following fields that may *significantly* affect the performance of the solver. Options that affect the formulation of the problem are:

`r0`	initial value for r in (DD). If $r0 < 0$, a heuristic will select a very large number (1e10). IMPORTANT: To improve convergence, use a smaller value. [default -1 (Heuristic)].
`zbar`	an upper bound $\bar{z}$ on the objective value at the solution [default 1.0e10].
`penalty`	penalty parameter Γ in (DD) that enforces feasibility in (D). IMPORTANT: This parameter must be positive and greater than the trace of the solution X of (P). [default 1e8].
`boundy`	determines the bounds l and u on the variables y in (DD). That is, $-boundy = l \leq y_i \leq u = boundy$ for all i. [default: 1e7].

Fields in the `OPTIONS` structure that affect the stopping criteria for the solver are:

`gaptol`	tolerance for duality gap as a fraction of the value of the objective functions [default 1e-6].
`maxit`	maximum number of iterations allowed [default 1000].
`steptol`	tolerance for stopping because of small steps [default 1e-2].
`pnormtol`	$\|P(\nu)\|$ of solution should also be less than [default 1e30].
`inftol`	the value r in (DD) must be less than this tolerance to classify the final solution of (D) as feasible. [default 1e-8].
`dual_bound`	Terminate the solver when it finds a feasible point of (D) with an objective greater than this value. (Helpful in branch-and-bound algorithms.) [default 1e+30].

Fields in the `OPTIONS` structure that affect printing are:

`print`	= k to display output at each k iteration, else = 0 [default 10].
`logtime`	=1 to profile the performance of DSDP subroutines, else =0. (Assumes proper compilation flags.)
`cc`	add this constant the objective value. This parameter is algorithmically irrelevant, but it can make the objective values displayed on the screen more consistent with the underlying application [default 0].

Other fields recognized in `OPTIONS` structure are:

`rho`	to set the potential parameter ρ in the function (8) to this multiple of the conic dimension n. [default: 3] IMPORTANT! Increasing this parameter to 4 or 5 may significantly improve performance.
`dynamicrho`	to use dynamic rho strategy. [default: 1].
`bigM`	if > 0, the variable r in (DD) will remain positive (as opposed to nonegative). [default 0].
`mu0`	initial barrier parameter ν. [default -1: use heuristic]
`reuse`	sets a maximum on the number of times the Schur complement matrix can be reused. Larger numbers reduce the number of iterations but increase the cost of each iteration. Applications requiring few iterations (¡60) should consider setting this parameter to 0. [default: 4]

For instance, the commands

```
> OPTIONS.gaptol = 0.001;
> OPTIONS.boundy = 1000;
> OPTIONS.rho = 5;
> [STAT,y,X] = DSDP(b,AC,OPTIONS);
```

asks for a solution with approximately three significant digits, bound the y variables by -1000 and $+1000$, and use a potential parameter ρ of 5 times the conic dimension. Some of these fields, especially `rho`, `r0`, and `ybound` can significantly improve performance of the solver.

Using a fourth input argument, the command

```
> [STAT,y,X] = DSDP(b,AC,OPTIONS,y0);
```

specifies an initial solution `y0` in (D). The default starting vector is the zero vector.

5.4 Solver Performance and Statistics

The second and third output arguments return objective values for (D) and (P), respectively.

The first output argument is a structure with several fields that describe the solution of the problem:

`stype`	`PDFeasible` if the solutions to both (D) and (P) are feasible, `Infeasible` if (D) in infeasible, and and `Unbounded` if (D) is unbounded.
`obj`	an approximately optimal objective value.
`pobj`	objective value of (PP).
`dobj`	objective value of (DD).
`stopcode`	equals 0 if the solutions to (PP) and (DD) satisfy the prescribed tolerances and equals nonzero if the solver terminated for other reasons.

Additional fields describe characteristics of the solution:

`tracex`	the trace of the solution X of (P).
`r`	the multiple of the identity element added to $C - \mathcal{A}^T(y)$ in the final solution to make S positive definite.
`mu`	the final barrier parameter (ν).
`ynorm`	the largest element of y (infinity norm).
`boundy`	the bounds placed on the magnitude of each variable y.
`penalty`	the penalty parameter Γ used by the solver, which must be greater than the trace of the variables X in (P). (see above).

Additional fields provide statistics from the solver:

`iterations`	number of iterations used by the algorithm.
`pstep`	the final step length.in (PP)
`dstep`	the final step length in (DD).
`pnorm`	the final value $\|P(\nu)\|$.

`rho`	the potential parameter (as a multiple of the total dimension of the cones).
`gaphist`	a history of the duality gap.
`infhist`	a history of the variable r in (DD).
`datanorm`	the Frobenius norm of C, A and b.

DSDP has also provides several utility routines. The utility `derror( · )` verifies that the solution satisfies the constraints and that the objective values (P) and (D) are equal. The errors are computed according to the the standards of the DIMACS Challenge[7].

A Brief History

DSDP began as a specialized solver for combinatorial optimization problems. Over the years, improvements in efficiency and design have enabled its use in many applications. Below is a brief history of DSDP.

1997 At the University of Iowa the authors release the initial version of DSDP. It solved the semidefinite relaxations of the maximum cut, minimum bisection, s-t cut, and bound constrained quadratic problems [6].

1999 DSDP version 2 increased functionality to address semidefinite cones with rank-one constraint matrices and LP constraints [5]. It was used specifically for combinatorial problems such as graph coloring, stable sets [2], and satisfiability problems.

2000 DSDP version 3 was a preliminary implementation of a general purpose SDP solver that addressed applications from the Seventh DIMACS Implementation Challenge on Semidefinite and Related Optimization Problems [7]. It ran in serial and parallel [1].

2002 DSDP version 4 added new sparse data structures to improve efficiency and precision. A Lanczos based line search and efficient iterative solver were added. It solved all problems in the SDPLIB collection that includes examples from control theory, truss topology design, and relaxations of combinatorial problems [3].

2004 DSDP version 5 [4] features a new efficient interface for semidefinite constraints, a corrector direction, and extensibility to structured applications in conic programming. Existence of the central path was ensured by bounding the variables. New applications from computational chemistry, global optimization, and sensor network location motivated the improvements in efficiency in robustness.

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[8] 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

[1] S. J. Benson. Parallel computing on semidefinite programs. Technical Report ANL/MCS-P939-0302, Mathematics and Computer Science Division, Argonne National Laboratory, March 2003.

[2] S. J. Benson and Y. Ye. Approximating maximum stable set and minimum graph coloring problems with the positive semidefinite relaxation. In *Applications and Algorithms of Complementarity*, volume 50 of *Applied Optimization*, pages 1–18. Kluwer Academic Publishers, 2000.

[3] S. J. Benson and Y. Ye. DSDP3: Dual-scaling algorithm for general positive semidefinite programming. Technical Report ANL/MCS-P851-1000, Mathematics and Computer Science Division, Argonne National Laboratory, February 2001.

[4] S. J. Benson and Y. Ye. DSDP5: Software for semidefinite programming. Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, September 2005. Submitted to ACM Transactions of Mathematical Software.

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

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

[7] DIMACS. The Seventh DIMACS Implementation Challenge: 1999-2000. `http://dimacs.rutgers.edu/Challenges/Seventh/`, 1999.

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