DSDP4 User Guide [1]

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.

March 28, 2002

[1]This technical report,ANL/MCS-TM-248, 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.

Abstract

DSDP is an implementation of the dual-scaling algorithm for semidefinite programming. The software package has been implemented in the C programming language and can be used as a subroutine library, and Matlab routine, and an executable that read SDPA files. New features in this version include a Lanczos procedure for determining the step size, more precise primal solutions, and improved performance on the standard test suites.

Contents

Chapter 1

Semidefinite Programming

The DSDP package uses a dual-scaling algorithm to solve semidefinite optimization problems of the form

$$\begin{array}{lll} & \text{Minimize} & C \bullet X \\ \text{(SDP)} & & \\ & \text{Subject to} & A_i \bullet X = b_i, \quad i = 1, \ldots, m, \\ & & X \in K. \end{array} \tag{1.1}$$

The variable X must be a symmetric positive semidefinite matrix. Here $K = K_1 \oplus K_2 \oplus \cdots \oplus K_p$ and K_l is the cone of $n_l \times n_l$ symmetric positive semidefinite matrices. The data matrices $C, A_i \in \Re^{n \times n}$ are symmetric. 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 positive (semi) 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 variables are $y \in \Re^m$ and the symmetric positive semidefinite matrix S. The dual problem of (SDP) can be written as follows:

$$\begin{array}{lll} & \text{Maximize} & b^T y \\ \text{(DSP)} & & \\ & \text{Subject to} & \displaystyle\sum_{i=1}^m y_i A_i + S = C, \quad S \in K, \end{array} \tag{1.2}$$

where b_i and y_i is the ith elements of the vectors b and y, respectively.

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

Chapter 2

Installation Instructions

The compressed tar file DSDP4.5.tar.gz contains an implementation of the dual-scaling algorithm for semidefinite programming optimization problems.

1. Create the DSDP4.5 directory structure with

   ```
   gunzip -d DSDP4.5.tar.gz
   tar -xvf DSDP4.5.tar
   ```

   This command produces the directory DSDP4.5 and several subdirectories.

2. Change directories to DSDP4.5. Several executables have been provided. If one of these runs on your architecture, proceed to step 3. Otherwise compile the executables using

   ```
   cd DSDP4.5
   make install
   ```

   This command creates libraries in each of the subdirectories and the executables for DSDP.

3. DSDP4 can be called from MATLAB version 6.0. Run the sample problems by starting MATLAB in the **DSDP4.5** directory and typing

   ```
   > check;
   ```

   Compare the output with the output in **demo.linux**. For help using the package, type `help dsdp4`.

   Run the executables by switching to directory **DSDP4.5/exec** and typing

```
dsdp4  truss1.dat-s
```

Compare the output with that in **demo.linux**. If the output from any of the tests differs significantly from the files, please report it to the developers.

2.1 LAPACK

The most common cause of trouble while compiling or running DSDP concerns linking the program to LAPACK. DSDP links with the LAPACK routine `dsyev` to compute the eigenvalues and eigenvectors of symmetric matrices. On most architectures, this routine is called from C programs using `dsyev_`. If this calling sequence is different on your machine, modify the file **DSDP4.5/src/vecmat/eigs.c** . This file contains the line

```
#define dsdpdsyev dsyev_
```

which can be modified to call the appropriate routine.

If you have an optimized version of LAPACK available, you may link to it by modifying the LAPACK macro in the file **DSDP4.5/src/Makefile**. Users with MATLAB 5 and lower should add the LAPACK variable from the `dsdp` target.

You may also have to change the MEX macro in the directory **DSDP4.5/src/Makefile**. This macro specifies the location of the MATLAB program and the mex functionality in particular.

2.2 Architectures

DSDP has been compiled using several different compilers and on several different architectures. In particular, it has been tested on linux, solaris, HP, and the Microsoft compiler. When compiling on a PC, the makefile may have to be modified. Futhermore, the application driver *sdpa.c* will have to define the constant `__DSDP_PC` for timing purposes.

Chapter 3

Using DSDP

3.1 Standard Output

The progress of the DSDP solver can be monitored using standard output printed to the screen. The following is an example output from a random problem.

```
Iter     Primal           Dual       Infeas.    Mu    Step  Pnrm
-----------------------------------------------------------------
0    0.00000000e+00  -2.47128801e+01 0.0e+00 1.0e+01  0.00  1.00
1   -1.28537231e+01  -1.64471021e+01 0.0e+00 1.2e+00  0.08 26.94
2   -1.28537231e+01  -1.59965110e+01 0.0e+00 2.3e-01  1.00  1.35
3   -1.28537231e+01  -1.59071616e+01 0.0e+00 2.1e-01  1.00  1.07
4   -1.28537231e+01  -1.55639753e+01 0.0e+00 1.5e-01  1.00  1.22
5   -1.50630757e+01  -1.53435713e+01 0.0e+00 1.4e-01  0.08 12.59
6   -1.51189868e+01  -1.52669146e+01 0.0e+00 1.0e-01  0.28  3.72
7   -1.51505856e+01  -1.52461229e+01 0.0e+00 1.0e-02  1.00  0.84
8   -1.52220358e+01  -1.52350997e+01 0.0e+00 9.2e-03  0.12  8.91
```

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

`Iter`	the current iterate number
`Primal`	the current estimate of the primal objective function
`Dual`	the current dual objective function
`Infeas`	the infeasibility in the current dual solution This number is the multiple of the identity matrix that has been added to the dual matrix
`Mu`	a central path parameter. This parameter decreases to zero as the points get closer to the solution

`Step` the multiple of the dual step-direction used

`Pnrm` The proximity to a point on the central path: $\|S^{.5}XS^{.5} - \hat{\mu}I\|$

3.2 DSDP with MATLAB

Help using the DSDP MATLAB interface can be found by simply typing `help dsdp4` in the directory **DSDP4.5**. The command

```
> [STAT, y, X] = DSDP(A,C,b)
```

attempts to solve the semidefinite program by using a dual-scaling algorithm. The arguments `A`, `C`, and `b`, are the same as the corresponding arguments in the SDPT3 [5] package for semidefinite programming. For a problem with p blocks and m constraints, `A` is a $p \times m$ cell array and `C` is a $p \times 1$ cell array. One block may contain LP variables, and the cells corresponding to this block should be a vector array. All other cells must contain a square, symmetric, real valued matrix. **The choice of sparse or dense matrix formats belongs to the user and will affect the performance of the solver.** The third argument `b` is a dense column vector of length m.

The second and third output arguments return the dual and primal solutions, respectively. The first output argument is a structure with several fields that describe the solution of the problem:

`obj`	an approximately optimal objective value
`primal`	an approximately optimal objective value if convergence to the solution was detected, `infeasible` if primal infeasibility is suspected, and `unbounded` if dual infeasibility is suspected
`dual`	an approximately optimal objective value if convergence to the solution was detected, `infeasible` if dual infeasibility is suspected, and `unbounded` if primal infeasibility is suspected
`iterates`	number of iterations used by the algorithm
`dual_infeasibity`	the multiple of the identity matrix added to $C - \mathcal{A}^T(y)$ in the final solution to make S positive definite
`gaphist`	a history of the duality gap
`infhist`	a history of the dual infeasibility
`termcode`	0 solution found, 1: primal infeasible, 2: dual infeasible

There are more ways to call the solver. The command

```
> [STAT, y, X] = DSDP(A,C,b,OPTIONS,y0)
```

specifies some options for the solver. The `OPTIONS` structure may contain any of the following parameters:

`gaptol`	tolerance for duality gap as a fraction of the value of the objective functions [default 1e-3]
`inftol`	tolerance for stopping because of suspicion of dual infeasibility. [default 1e-8]
`steptol`	tolerance for stopping because of small steps [default 1e-2]
`maxit`	maximum number of iterations allowed [default 100]
`matrixfreesize`	problems with more constraints than this number will be solved using a matrix-free method. These problems rely on an iterative solver and should not be solved to a high accuracy. [default 3000]
`printyes`	1, if want to display result in each iteration, else = 0 [default 1]
`dual_bound`	a bound for the dual solution. The solver stops when a feasible dual iterate with an objective greater than this value is found [default 1e+8].
`maxtrustradius`	maximum trust region radius used for step size. Smaller radii generally improve robustness of the algorithm but can also reduce the rate of convergence. If algorithm does not converge, for a problem, try this option. A radius of 3 is very robust. [default 1.0e8].
`r0`	multiple of the identity matrix added to the initial dual matrix. $S0 = C - \sum A_i y_i^0 + r0 * I$. If $r0 < 0$, a dynamic selection will be used. IF the solver does not converge, increase this number by an order of magnitude. To improve robustness and convergence, TRY this option. [default -1]

For instance, the commands

```
> OPTIONS.gaptol = 0.001
> [STAT, y, X] = DSDP(A,C,b,OPTIONS)
```

asks for a solutions with approximately three significant digits. The command

```
> [STAT, y, X] = DSDP(A,C,b,OPTIONS,y0)
```

specifies an initial dual vector `y0`. The default vector consists of positive and negative ones.

If only two output are used,

```
> [STAT,y] = DSDP()
```

returns only the solver statistics structure and an approximate dual solution `y` .

DSDP has also provides several utility routines. These routines include `read_sdpa()` which reads a file containing a problem in SDPA format and inserts it into the appropriate Matlab data structures. It also includes the utility `dimacserror()` with the errors in the solution, according the standards of the DIMACS Challenge. Finally, it has a `maxcut()` utility that creates a maximum cut problem from a graph.

3.3 Standalone version with SDPA files

DSDP can also be run without the MATLAB environment if the user has a problem written in sparse SDPA format. These executables have been put in the directory **DSDP4.5/exec/**. The file name should follow the executable. For example,

```
> dsdp4 truss4.dat-s
```

Other options can also be used with DSDP. These should follow the SDPA filename.

`-save <filename>` to save the solution into a file with a format similar to SDPA.

`-y0 <filename>` to specify an initial dual vector.

`-maxit <iter>` to stop the problem after a specified number of iterations.

`-gaptol <rtol>` to stop the problem when the relative duality gap is less than this number.

`-maxtrustradius <radius>` to limit the step size using a trust region. The default is 20. Smaller positive numbers improve the robustness of the solver, but can increase the number of iterations and the solution time.

3.4 Applying DSDP to Graph Problems

Within the directory **DSDP4.5/exec/** is a program **maxcut** which reads a file containing a graph, generates the SDP relaxation of a maximum cut problem, and solves the relaxtion. For example,

```
> maxcut Max2
```

The files that contain a graph should follow the DIMACS graph format. The first line should contain two integers. The first integer states the number of nodes in the graph, and the second integer states the number of edges. Subsequent lines have two or three entries separated by a space. The first two entries specify the two nodes that an edge connects. The optional third entry specifies the weight of the node. If no weight is specified, a weight of 1 will be assigned.

The same options that apply to reading SDPA files also apply here.

A similar program reads a graph from a file, formulates a minimum bisection problem or Lovasz theta problem, and solves it. For example,

```
> minbisection Max2
> theta Max2
```

reads the graph in the file **Max2** and solves this graph problem.

3.5 DSDP Application Programming Interface

DSDP4.5 can from within a C/C++ application through a set of subroutines. Examples of using the DSDP API can be found by looking at the files **DSDP4.5/src/dsdp.c** and **DSDP4.5/src/sdpa.c**, which read data from the MATLAB environment and a SDPA file, respectively, and solve the problem by using DSDP. Other files **DSDP4.5/src/maxcut.c** and **DSDP4.5/src/minbis.c** read a graph and formulate the maximum cut and minimum bisection problems. Each of these applications includes the header file **DSDP4.5/src/dsdp4.h** and links to the library **DSDP4.5/src/dsdplib.a**. All DSDP subroutines begin with the `DSDP` prefix and return an error code. A zero code indicates success, while a nonzero indicates that an error has occurred.

3.5.1 Basics

To call DSDP as a subroutine, the DSDP data structure must first be created with the command

```
int DSDPCreate(int, int*, int*, DSDP *);
```

The first argument is the number of constraints in the problem, the second argument is the number of blocks in the problem, and the third argument is an array of integers specifying the dimension of each block. The length of this array must be at least as long as the number of blocks in the problem and the dimension of each block corresponds the number of rows or columns of matrices in the block. The final argument should receive the address of a `DSDP` structure. This routine will allocate some resources for the solver and set the pointer.

After setting the data, which will be explained in the next section, DSDP must process the data with the routine

```
int DSDPSetup(DSDP);
```

This routine allocates some additional resources for the solver and computes the eigenvalues and eigenvectors of the constraint matrices. This routine should be called before solving the problem, and should be called only once for each DSDP solver created. The command

```
int DSDPSolve(DSDP);
```

attempts to solve the problem. This routine can be called more than once. For instance, the user may try solving the problem using different initial points. Each solver created should be destroyed with the command

```
int DSDPDestroy(DSDP);
```

This routine frees the work arrays and data structures allocated by the solver.

3.5.2 Set Data

A positive semidefinite programming programming problem may be defined by using the following "Set" routines. The first argument in each of these routines is a pointer to the DSDP solver. To set the dual objective function, the routine

```
int DSDPSetDualObj(DSDP, int, double* );
```

accepts an array of double precision variables that contain the dual objective function b. The second argument must equal the dimension of this objective vector and match the number of constraints specified the `DSDPCreate()` command. DSDP will copy this data but not use the array.

The data matrices can be specified by any of the following commands. The choice of data structures is up to the user, and the performance of the problem depends upon this choice of data structures. In each of these routines, the first four arguments are a pointer to the DSDP solver, the block number, and the constraint number, and the number of rows and columns in the matrix. The blocks must be numbered consecutively, beginning with the number 0. Constraints are numbered consecutively beginning with the number 1. The primal objective matrices are specified using constraint 0. The data that is passed to the DSDP solver will be used in the solver, but not modified. The user is responsible for freeing the arrays of data it passes to DSDP after solving the problem.

To set data in a dense matrix, the routine

```
int DSDPSetDenseMat(DSDP, int, int, int, double *);
```

hands DSDP a pointer to an array of length $ni \times ni$, where ni is the number of rows in the matrix. Since the matrix must be symmetric, the array may be in either row major or column major form. The routine

```
int DSDPSetSparseMat(DSDP, int, int, int, double *, int *, int *);
```

specifies a matrix in sparse form. The fifth argment is an array containing the nonzeros of the matrix. The sixth argument is an array of integers specifying the row number of each nonzero. The last argument specifies the number of nonzeros in each column. Row and column number must be numbered consecutively beginning with 0. The dimension of this integer array is one more than the number of rows and columns in the matrix. The first element in this array in 0. Subsequent elements in this array equal the previous element plus the number of nonzeros in that column. These indices indicate where in the fifth and sixth arguments to find the nonzeros for a particular column. This sparse column format, or sparse row format, given the symmetry of the matrix, is the same sparse format used by MATLAB. In the routines `DSDPCSetDenseMat()` and `DSDPCSetsparseMat()` the data passed into the solver will be used by the solver, and should not be freed until the solver is finished. The routine

```
int DSDPSetDenseMatWSparseData(DSDP, int, int, int, double*, int*, int*);
```

will create a dense matrix and initialize it with the values specified in the final three argments. The nonzero values in this matrix are specified in the final three arguments, which require the same format used in the routine `DSDPSetSparseMat()`.

A diagonal matrix can be specified with the command

```
int DSDPSetDiagMat(DSDP, int, int, int, double *);
```

The last argument must be an array equal to the rows and columns of the block. This command is especially appropriate when a block of linear programming variables is part of the problem. A diagonal matrix can also be specified in sparse format by using the command

```
int DSDPSetSpDiagMat(DSDP, int, int, int, double *, int *, int);
```

In this routine, the fifth argument is an array of nonzeros, the sixth column is an array of integers specifying the row/column number, and the seventh argument specifies the the length of the arrays in the fifth and sixth arguments which should equal the number of nonzeros in this constraint block. In the routines `DSDPCSetDiagMat()` and `DSDPCSetSpDiagMat()` the data passed into the solver will be used by the solver, and should not be freed until the solver is finished.

If every element in matrix can the same value, the command

```
int DSDPSetConstantMat(DSDP, int, int, int, double);
```

will create a data structure that will make the computations in the algorithm more efficient. Similarly, a program has a constraint where one block has a single nonzero element located along the diagonal. The routine

```
int DSDPSetSingletonMat(DSDP, int, int, int, int, double);
```

creates an appropriate data structure. The fifth argument specifies the diagonal element where the nonzero element resides and the sixth argument is the value of that element. Matrices with a 2×2 block form can be specified with the following command:

```
int DSDPSetTwoTwoMat(DSDP, int, int, int,
                     int i0, int i1, double d00, double d01, double d11);
```

The integers $i0$ and $i1$ specify the two row numbers of the nonzero block, while the next three digits define the elements of the block. The elements in the upper triangular matrix are specified in row major order. Matrices with a 3×3 block form can be specified by using

```
int DSDPSetThreeThreeMat(DSDP, int, int, int,
                         int i0,int i1,int i2,
                         double d00,double d01,double d02,
                         double d11,double d12,double d22);
```

The integers $i0$, $i1$, and $i2$ specify the three row numbers of the nonzero block. The six double precision arguments that follow define the upper triangular part of the matrix in row major form. Constraint matrix blocks that have rank one can be specified by using the command

```
int DSDPSetRank1Mat(DSDP, int, int, int, double, double *);
```

The sixth argument is an array of length equal to the number of rows and columns in the block. The constraint is the outer product of this vector with itself times a scalar, which is specified in the fifth argument. A zero matrix, often used in problems with multiple blocks, can be specified with the routine

```
int DSDPSetZeroMat(DSDP, int, int, int);
```

which specifies the constraint, block, and size of the matrix.

3.5.3 Improving Performance

The performance of the DSDP may be improved with the proper selection of parameters and data structures. DSDP accepts multiple data structures for the data matrices, including dense, sparse, and diagonal representations. The choice of data types belongs to the user and may affect the performance. In addition, DSDP may achieve greater efficiency by placing diagonal blocks before nondiagonal blocks and smaller blocks before larger blocks.

DSDP allows the user to specify the initial starting point. The command

```
int DSDPSetY0(DSDP, int, double *);
```

specifies the initial dual vector y to be used. The second argument is an integer corresponding to the number of constraints in the problem. The length of the array in the third argument must exceed this dimension. DSDP will copy this data into its own structure. The initial dual matrix can be partially controlled by using the command

```
int DSDPSetInitialInfeas(DSDP, double);
```

This command specifies a positive number r and sets $S^0 = C - \sum A_i y_i^0 + rI$, where I is the identity matrix. If $r < 0$, a default value will of r will be chosen. If S^0 is not positive definite, the routine `DSDPSolve()` will return an error code of -100.

The algorithm also used trust regions to determine step sizes. The size of the trust regions affects the convergence and robustness of the algorithm. By default, DSDP uses a relatively

large trust region. Smaller trust regions usually improve the robustness of the algorithm, allowing it to solve more problems, but larger trust regions often reduce the number of iterations needed to converge. The user may specify a maximum trust region radius using the command

```
int DSDPSetMaxTrustRadius(DSDP, double);
```

A maximum radius of 3 is a very robust choice, although a radius of 100 is chosen as a default for most problems.

3.5.4 Set Options

A variety of different options may be set when using DSDP. The precision of the solution can be set by using the routine

```
int DSDPSetGapTol(DSDP, double);
```

This command will terminate if the solver finds a sufficiently feasible solution such that the difference between the primal and dual objective values, normalized by the magnitude of the dual objective value, is less than the prescribed number. A tolerance of 0.001 provides roughly three digits of accuracy, while a tolerance of $1.0e-5$ provides roughly five digits of accuracy. The command

```
int DSDPSetInfeasTol(DSDP, double);
```

specifies how small the dual infeasibility constant r must be to be an approximate solution, and the routine

```
int DSDPSetMaxIts(DSDP, int);
```

specifies the maximum number of iterates. The routine

```
int DSDPSetDualBound(DSDP,double);
```

specifies an upper bound on the dual solution. The algorithm will terminate when it finds a point whose dual infeasibility is less than the prescribed tolerance and whose dual objective value is greater than this number. DSDP may print some information to the screen. This information can be turned off by passing a zero value into the second argument of the routine

```
int DSDPPrint(DSDP, int);
```

A nonzero value asks to print the information. The command

```
int DSDPSetScaling(DSDP, int, double);
```

does some scaling of the problem. DSDP will scale the constraint in the second argument by implicitly dividing the data in this constraint by the number in the third argument. A constraint value of zero divides the primal objective matrix by this number. An appropriate value is the magnitude of the largest element in the constraint or an average nonzero value in the constraint.

Very large problems may require more RAM than a machine has to offer. In these problems, the Schur matrix can be solved used an iterative solver that does not create the large dense matrix associated with this method. Problems with constraints higher than a specified tolerance will use this the matrix free method. Specify this tolerance using the routine

```
int DSDPSetMatrixFreeSize(DSDP,int);
```

Problems solved using this method should use a low stopping tolerance.

3.5.5 Viewing Solution

The dual solution vector y can be viewed by using the command

```
int DSDPGetY(DSDP, int, double *);
```

The user passes the size of the dual vector and an array of double to DSPP, which will copy the dual solution into this array. The routine

```
int DSDPGetInfeasiblity(DSDP, double *);
```

returns the dual infeasibility r used in the final iteration. To view the primal solution, the user must call either the command

```
int DSDPSetDenseXMat(DSDP, int, int, double *);
```

or

```
int DSDPSetDiagXMat(DSDP, int, int, double *);
```

before solving the problem. The user should use the first command for SDP blocks and the second command for linear programming variable blocks. The first, second, and third arguments to these routines are a pointer to the DSDP solver, the block number, and the number of rows and columns in the block,respectively. In the fourth argument, the user passes an array of double precision variables long enough to store either a full dense matrix of the size designated in the third argument, or a vector whose dimension equal the number of rows or columns in the block. The command

```
int DSDPNoXMat(DSDP, int);
```

specifies a block for which the dual solution is not needed. Perfomance in the routine `DSDPSetup()` is generally better when the nondiagonal primal solution matrices X have been requested because it uses this array as work space instead of dynamically allocating it for every matrix.

The success of DSDP can be interpreted with the command

```
int DSDPStopReason(DSDP, int *);
```

This command sets the second argument to an integer. If this integer equals 1, DSDP found a primal and dual solution. If the integer is 2 DSDP terminated due to small steps. If the integer equals 3, DSDP asserts that the problem is primal infeasible and the dual problem is unbounded, and if the integer equals 4, DSDP asserts that the problem is dual infeasible and primal unbounded. If the integer is -5 DSDP terminated after reaching the maximum number of iterations.

Information about the convergence of the solver can be obtained with the commands

```
int DSDPGetGapHistory(DSDP, int, double *);
```

and

```
int DSDPGetInfeasHistory(DSDP, int, double *);
```

These commands retrieve the history of the duality gap and the dual infeasibility for up to 100 iterations. The user passes an array of doubles and the length of this array.

3.5.6 Monitors

The user can also monitor the solution at each iteration. The command A user can write a routine of the form

```
int (*monitor)(int,double,double,double,double,double,double);
```

These arguments represent the iteration number, primal objective value, dual objective value, dual residual norm, μ, step size, and $\|P\|$. This routine can print these stats and do other things. To set this routine, use the command,

```
int DSDPSetMonitor(DSDP dsdp,
       int (*monitor)(int,double,double,double,double,double,double));
```

In this routine, the first argument is the solver and the second argument is the monitoring routine.

Chapter 4

Parallel DSDP

The DSDP package can also be run in parallel using multiple processors.

4.1 Installation

To compile PDSDP using the following steps.

1. Install PETSc[1][2] Version 2.1.1, and set the `PETSC_ARCH` and `PETSC_DIR` environmental variables accordingly. This package contains parallel linear solvers and is freely available to the public.
2. Copy the files **DSDP4.5/src/PDSDP/dsdpalgebra.h**, **DSDP4.5/src/PDSDP/dsdpkernal.h**, and **DSDP4.5/src/PDSDP/petscroutines.c** into the directory **DSDP4.5/src/vecmat/**.
3. Copy the files **DSDP4.5/src/PDSDP/Makefile** and **DSDP4.5/src/PDSDP/sdpa.c**, into the directory **DSDP4.5/src/**.
4. From the directory **DSDP4.5/src/**, type `make dsdpsdpa BOPT=O`. Other drivers for maximum cut problems and theta problems are also available.

4.2 Usage

PDSDP can be used much like the standalone version of DSDP that reads SDPA files and graphs. Given a SDPA file such as `truss1.dat-s`, the command

```
mpirun -np 2 dsdp4  truss1.dat-s -log_summary
```

will solve the problem using two processors. Additional processors may also be used. This implementation is best suited for very large problems.

Chapter 5

Dual-Scaling Algorithm

This section summarizes the dual-scaling algorithm, which is a modification of the linear programming algorithm. Convergence of the algorithm has been discussed previously[3].

Following conventional notation, let

$$\mathcal{A}X = \begin{bmatrix} A_1 \bullet X & \dots & A_m \bullet X \end{bmatrix}^T \quad \text{and} \quad \mathcal{A}^T y = \sum_{i=1}^{m} A_i y_i,$$

5.1 Step Direction

Given a dual point (y, S) such that $\mathcal{A}^T y + S - C = R$ (R is the infeasiblity matrix) and $S \succ 0$, and a barrier parameter $\hat{\mu} > 0$, each iteration of the dual-scaling algorithm computes a step direction Δy by solving the linear system

$$\begin{pmatrix} A_1 S^{-1} \bullet S^{-1} A_1 & \cdots & A_1 S^{-1} \bullet S^{-1} A_m \\ \vdots & \ddots & \vdots \\ A_m S^{-1} \bullet S^{-1} A_1 & \cdots & A_m S^{-1} \bullet S^{-1} A_m \end{pmatrix} \Delta y = \frac{1}{\hat{\mu}} b - \mathcal{A}(S^{-1}) - \mathcal{A}(S^{-1} R S^{-1}) \qquad (5.1)$$

and $\Delta S = -\mathcal{A}^T \Delta y - R$. For notational convenience, we label the left-hand matrix of (5.1) M. For any feasible X, this linear system (5.1) can be derived by taking the Schur complement of the equations

$$\mathcal{A}(\Delta X) = 0 \qquad \mathcal{A}^T(\Delta y) + \Delta S = -R \qquad \hat{\mu} S^{-1} \Delta S S^{-1} + \Delta X = \hat{\mu} S^{-1} - X, \qquad (5.2)$$

which are the Newton equations for the nonlinear system

$$\mathcal{A}X = b \qquad \mathcal{A}^T y + S = C \qquad \hat{\mu} S^{-1} = X, \qquad (5.3)$$

at a point (X, y, S) such that $\mathcal{A}X = b$.

For a feasible dual point, the step direction also maximizes the function

$$\phi(y) = b^T y + \hat{\mu} \ln \det S \tag{5.4}$$

subject to $\mathcal{A}^T y + S = C$ and 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 [7].

Using the dual step direction and (5.2), the primal matrix

$$X(S, \hat{\mu}) = \hat{\mu} S^{-1} - \hat{\mu} S^{-1} \Delta S S^{-1} \tag{5.5}$$

and satisfies the constraints $\mathcal{A}X(S, \hat{\mu}) = b$.

Given a feasible dual starting point and appropriate choices for steplength and $\hat{\mu}$, 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.

A more thorough explanation of the the dual-scaling algorithm used to solve these problems can be found in the papers *Solving Large-Scale Sparse Semidefinite Programs for Combinatorial Optimization*[4] and *DSDP 3: Dual-Scaling Algorithm for General Positive Semidefinite Programs*. The C code that implements the algorithms can be downloaded from the Web site at *http://www.mcs.anl.gov/∼benson/* or *http://dollar.biz.uiowa.edu/col/*.

More information about the convergence of the algorithm and performance of the software can be found at [3] and [4].

5.2 Step Length

The algorithm then selects a step size $\beta_k \in (0, 1]$ such that $y^{k+1} = y^k + \beta_k \Delta y$ and $S^{k+1} = S^k + \beta_k \Delta S \succ 0$. New in DSDP4 is the use of a Lanczos procedure [6] that approximates the distance to the boundary of the positive definite cone. DSDP4 also limits the step size with a trust region. Minimizing the dual potential function over a trust region with radius α, gives the solution

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

DSDP uses a radius of 100 most problems, but smaller values often make the solver more robust, although also force more time and iterations to find a solution.

5.3 Central Path

The linear algebra used to compute and solve the linear system (5.1) affects the cost of each iteration. However, the choice of parameters in the algorithm, especially $\hat{\mu}$, has an enormous impact upon number of iterations required for the algorithm to converge.

The heuristic that we use looks for

$$\begin{array}{rll} \hat{\mu} = & \arg\min & X(S,\mu) \bullet S \\ & \text{subject to} & X(S,\mu) \succeq 0. \end{array}$$

New to DSDP4 is a Lanczos procedure that estimates this quantity. When this value cannot be found, DSDP computes the value of μ that creates $b^T \Delta y = 0.$ and sets $\hat{\mu} = 0.7\mu$.

Chapter 6

Acknowledgements

We thank Cris Choi and Xiong Zhang for their help in developing this code, and Hans Mittelmann for his efforts in testing and benchmarking it.

Bibliography

[1] S. Balay, W. D. Gropp, L. McInnes, B. F. Smith. PETSc Home Page, http://www.mcs.anl.gov/petsc, 1998.

[2] S. Balay, W. D. Gropp, L. McInnes, B. F. Smith. PETSc 2.0 Users Manual. Technical Report ANL-95/11 - Revision 2.1.1, Argonne National Laboratory, 1998.

[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] S. J. Benson and Y. Ye. DSDP3: Dual-Scaling Algorithm for General Positive Semidefinite Programs. Technical Report ANL/MCS-P851-1000, Argonne National Laboratory, November, 2000.

[5] 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.

[6] K. C. Toh, A Note on the Calculation of Step-lengths in Interior-Point methods for Semidefinite Programming , Computational Optimization and Applications, to appear.

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