Computational Optimization Laboratory Positive Semidefinite Programming User Guide[1]

Steven Benson Yinyu Ye Xiong Zhang

This software package uses a dual scaling algorithm to solve the positive semidefinite relaxation of combinatorial optimization problems. It solves problems of the form:

$$
\text{(SDP)} \quad \begin{array}{ll} \text{Minimize} & C \bullet X + c^T x \\ \text{Subject to} & (a_i a_i^T) \bullet X + \hat{a}_i^T x = b_i, \quad i = 1, \ldots, m. \\ & X \succeq 0, \; x \geq 0. \end{array} \tag{1}
$$

where

- $X \in \Re^{n \times n}$: the positive semidefinite and symmetric variable matrix,
- $x \in \Re^p$: the positive variable vector,
- $C \in \Re^{n \times n}$: the symmetric objective coefficient matrix,
- $c \in \Re^p$: the objective coefficient vector,
- $a_i \in \Re^n$: the column vectors that define rank one matrix constraints,
- $\hat{a}_i \in \Re^p$: the column vectors that define the vector constraints,
- $b \in \Re^m$: the constraint right-hand-side vector,

The algorithm is designed to solve large scale problems with sparsity and structure in the data. Examples of problems for which this program was specially designed include:

1. Maximum/Minimum cut problem: For $i = 1, \ldots, n$, this problem sets $b = e, a_i = e_i$, and c and $\hat{a}_i$ as null. The vector e is the vector of ones and e_i is the i^{th} coordinate vector.
2. Maximum/Minimum equal/unequal cut problem: These problems contain the n constraints of the maximum cut problem and add one additional constraint defined by $a_{n+1} = e$ and $b_{n+1} = \kappa^2$.
3. the box-qp problem: In this problem, the diagonal elements of the matrix X is bounded above by one. The constraints are $b = e, a_i = \hat{a}_i = e_i, i = 1, \ldots, n$,

[1] We would like to thank Cris Choi for his many helps in developing this code.

4. the max-cover/stable-set problem with $n-1$ nodes: $b = e, a_i = e_i, i = 1, \ldots, n-1, n$; and $a_{n+j} = e_{o,d,n}$ for any (o,d) being a link in the graph, $j = 1, \ldots, L$, where L is the total number of links.

The dual of the problem is

$$\text{(DSDP)} \qquad \begin{array}{ll} \text{Maximize} & b^T y \\ \text{Subject to} & \sum_{i=0}^{m} (a_i a_i^T) y_i + S = C, \quad S \succeq 0 \\ & \sum_{i=0}^{m} \hat{a}_i y_i + s = c, \qquad s \geq 0 \end{array} \tag{2}$$

where

- $y \in R^m$: the variable vector,
- $S \in \Re^{n \times n}$: the positive semidefinite and symmetric variable matrix,
- $s \in \Re^p$: the positive variable vector,

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* and *Mixed linear and semidefinite programming for combinatorial optimization.* The C code that implements the algorithms can be downloaded from the COPL web site at *http://dollar.biz.uiowa.edu/col/.*

1 Setup

An executable code named **dsdp** can be created using the following steps.

1. Download all the files into a directory of your choice. The files are:
   - **Makefile** to compile the code and create executables
   - Header files:

     `SDPdef.h` `SDPfun.h` `SDPfun2.h`
   - Source files:

```
sdpalgm.c   sdpdual.c  sdpinput.c  sdporder.c  sdprank.c  sdpuser.c
sdpalloc.c  sdpexit.c  sdpmain.c   sdpparam.c  sdpsmw.c   sdpvect.c
sdpcheck.c  sdpheur.c  sdpmatx.c   sdpprint.c  sdpsymb.c  sdpxlist.c
sdpcres.c   sdpinit.c  sdpnfac.c   sdpproc.c   sdptime.c
```

- Example data files:

  `MaxTest  MaxG11  EcutG11  BoxQpG11  MinEcutG11  MaxCoverTest StableG00`

  The problem **MaxTest** is very small and will be used as an example in this documentation.

2. Type command `make` to compile all source codes and link them all together to form program **dsdp**. This target should compile and link all of the C files. If compiling on a PC, edit the first two lines of the file **SDPdef.h** to comment out the line `#define UNIXMACH` and remove the comments from the second line: `/* #define PCMACHINE */`.

2 Usage

The user can execute the program as follows:

```
dsdp [-f {filename}] [-s {filename}]
```

The contents in '`[]`' are optional, and the contents in "{}" are required if the corresponding option appears in the command line. Here

`-f` specifies the input data file name. If this option is omitted, the system will prompt the user for a file name.

`-s` specifies the specification file name. The system will use **COPLSDP.spc** as the default specification file name. The system will use default settings if the file is not found.

For example, to solve problem **MaxTest** using specification file **MYSPC**, type the command line

`dsdp -f MaxTest -s MYSPC`.

Or to solve problem **MaxG11** with the default specification file, type

`dsdp -f MaxG11`.

Alternatively the user may type `dsdp` and receive the following prompt:

```
Please type the name of your data file:
```

After entering the name of the file that contains the problem data, the program will process data and return its answer.

3 Input file format

The input file to be processed by COPL_SDP must conform to the following format. Each file must contain the key words `ROWS`, `POBJV`, `POBJV`, `CONM`, `CONV`, and `ENDDATA`. Each of this keywords indicates which portion of the problem data the subsequent lines describe. Subsequent lines that contain the data must begin with a blank space. The items in each row are separated by one or more spaces. The data C, c, a_i, and $\hat{a}_i$ are generally sparse, so only the nonzero entries get entered. The other values are assumed to be zero. No empty rows are allowed in the file.

`ROWS` This section will contain one line for each constraint. Each line must contain three entries.

- The first entry of each row must be one of the letters "`E`","`L`", or "`G`". Use "`E`" if the constraint is of the form $(a_i a_i^T) \bullet X + \hat{a}_i^T x = b_i$. Use "`L`" if the constraint is of the form: $(a_i a_i^T) \bullet X + \hat{a}_i^T x \leq b_i$. If the constraint has the form $(a_i a_i^T) \bullet X + \hat{a}_i^T x \geq b_i$ use "`G`".
- The second entry in the line is an integer that refers to the constraint number. The constraints should be numbered from 1 to m.
- The third entry in each line refers to the right hand side number b_i.

`POBJM` This section describes the primal objective matrix C. Since this matrix is symmetric, only the upper triangular part of the matrix needs to be entered. Each line in this section describes one nonzero element of the matrix.

- the first entry in each row of this section represents is an integer that corresponds to the row number. This integer is always greater than or equal to 1 and less than or equal to n.
- The second number represents the column number. The column numbers must not be less than row numbers, meaning the file describes just the upper triangular part of the symmetric matrix.
- The third number represents the value of the element of the matrix described by the first two entries.

`POBJV` This section describes the primal objective vector c. This section has one line for each nonzero entry. Each line contains two entries.

- The first entry is an integer that represents the index of the element in the vector. This number should be between 1 and p, where p is the length of the vectors c and $\hat{a}_i$.
- The second entry is the value of this element in the vector

`CONM` This section describes the rank one constraint matrices $a_i a_i^T$. Each row will have three entries.

- The integer in the first column refers the constraint number.
- The integer is the second column refers to the index of the nonzero in the constraint vector a_i.

- The third column represents the value of the element in the constraint.

`CONV` This section describes the vectors $\hat{a}_i$. Each line in this section describes one nonzero value and has three entries. The format for this section matches the format in the `CONM` section.

- The integer in the first column refers the constraint number.
- The integer is the second column refers to the index of the nonzero in the constraint vector $\hat{a}_i$.
- The third column represents the value of the element in the constraint.

`ENDDATA` This entry indicates the end of the file. It is required for all input files.

For example, suppose we want to solve the maximum cut problem for a graph with the following adjacency matrix:

$$\begin{bmatrix} 0 & 0.3 & 0 & 2.7 & 0 & 1.5 \\ 0.3 & 0 & -1.0 & 0 & 1.45 & 0 \\ 0 & -1.0 & 0 & -0.2 & 0 & 0 \\ 2.7 & 0 & -0.2 & 0 & 1.2 & 0 \\ 0 & 1.45 & 0 & 1.2 & 0 & 2.1 \\ 1.5 & 0 & 0 & 0 & 2.1 & 0. \end{bmatrix}$$

The problem can be described with th following input file. (An option described in Section 5 can indicate that this is a maximization problem instead of a minimization problem. Another option can create the Laplacian matrix, which is also helpful for some combinatorial problems.)

```
ROWS
 E 1  1
 E 2  1
 E 3  1
 E 4  1
 E 5  1
 E 6  1
POBJM
 1 2 .3
 1 4 2.7
 1 6 1.5
 2 3 -1.0
 2 5 1.45
 3 4 -0.2
 4 5 1.2
 5 6 2.1
POBJV
CONM
```

```
 1  1  1
 2  2  1
 3  3  1
 4  4  1
 5  5  1
 6  6  1
CONV
ENDATA
```

4 Interpreting Output

There are three types of output the program can produce. this output is described in the following subsections.

4.1 Standard Output

The program will print a variety of statistics for each problem to the screen. The output for the maximum cut MaxTest problem described in Section 3, with the described options, is shown below.

```
 ==========================================
 Computational Optimization Program Library

     (  C  O  P  L  )     Version 2.0

 Computational Optimization Lab.
 The University of Iowa (1998)
 ==========================================

 *************************** COPL STARTS ***************************


 dimension n ..................................................... 6
 dimension p ..................................................... 0
 dimension m ..................................................... 6
 density of S ............................................... 61.11%
 density of S_cholesky ...................................... 76.19%

ITER  |P(z)|      POBJ           DOBJ        RGAP     RHO   STEP  ||y||
-------------------------------------------------------------------------
  0     0.00  -0.000000e+00   1.528600e+01  2.0e+00    0.00  0.00  3e+00
  1     0.68   5.755407e+00   1.304330e+01  7.0e-01    1.20  0.95  3e+00
```

```
 2     0.66   6.318069e+00   1.144251e+01  5.2e-01    1.20  0.95  3e+00
 3     0.69   6.900322e+00   1.032865e+01  3.6e-01    1.20  0.95  2e+00
 4     0.71   7.576119e+00   9.711550e+00  2.2e-01    1.20  0.95  2e+00
 5     1.06   8.153929e+00   9.274277e+00  1.2e-01    1.49  0.95  2e+00
 6     0.81   8.677153e+00   9.281988e+00  6.1e-02    1.59  0.95  2e+00
 7     0.54   8.964277e+00   9.280009e+00  3.1e-02    1.60  0.95  2e+00
 8     0.69   9.099027e+00   9.260754e+00  1.6e-02    1.72  0.95  2e+00
 9     0.48   9.178289e+00   9.256109e+00  7.6e-03    1.87  0.95  2e+00
------------------------------------------------------------------------


 Using rows of primal matrix, best solution:         9.250000
 Using    6 random unit vectors, best solution is:   9.250000
 Using the dominant eigenvector, objective value:    9.250000


 time for initial data input ............................. 0.03 sec
 time to solve semidefinite program ...................... 0.03 sec
 time for vector solution generation ..................... 0.00 sec
 time for the whole process .............................. 0.06 sec

 **************************** COPL ENDS ****************************
```

After processing the data, the program will print out several statistics. Among them is the dimension of the matrices X and S, as well as the density of the full matrix S and the density of its Cholesky factor.

Upon entering the main loop of the algorithm, several more statistics will be printed at every iteration. These statistics are:

- `POBJ`: the primal objective function. If an asterisk appears on the left hand side, a feasible primal point has not been found and an approximate point is being used.
- `DOBJ`: the dual objective function.
- `RGAP`: the relative duality gap. This statistic is defined here to mean the difference between the primal and dual objective values divided by one plus the absolute value of their mean.
- `|P(z)|` a measure used in interior point algorithms.
- `STEP` the multiple of the dual step-direction used.
- `||y||` the infinity norm of the dual vector y.
- `Pot'l` the value of the dual potential function using ($\rho = 1.3 * (n + p)$). This statistic is not available when dense constraints are treated separately from sparse constraints.

At the end the of algorithm, the user may select some randomized algorithms to define a vector v. Letting $X = vv^T$, this primal objective value of these matrices will be evaluated. The best objective values found get printed below the semidefinite program information. If the user selects not to use the randomized rank reduction algorithms, these statistics will not appear.

Finally, the times required for the various stages of the algorithm get printed at the end.

An option can also be selected in the specification file that will provide a copy of the standard output into a file. The name of the file that constraints the output will be the name of the file with the input data appended with the suffix **.rec**.

A "*" may appear on the left of the iteration statistics, which means that a feasible primal matrix solution has not been generated. The primal objective value in this iteration represents the best estimation of a feasible primal objective value.

A copy of this output can be printed to a file by selecting the appropriate option in the specification file.

4.2 Dual Solution File

If the parameter `save_dual_solution` is set to "`yes`", the program will create a file that contains the dual solution. The name of the file will be the name of the data file with the suffix **.ysol** appended to the end. The file will contain m lines. Each line contains one number. The variables are ordering corresponding the order of the constraints in the data file.

For example, the file **MaxTest.ysol**, should look something like:

```
-2.25079856597928
-0.87541274810011
-0.00249308613519
-1.95094765322936
-2.37481732859028
-1.80133313542638
```

4.3 Rank one solution file

If the parameter `save_rank_one_solution` is set to "`yes`", the program will create a file that contains the vector v with the best objective value $C \bullet (vv^T)$. The name of the file will be the name of the data file with the suffix **.vsol** appended to the end. The file will contain n lines. Each line contains one number.

For example, the file **MaxTest.vsol** should look something like:

```
 1
-1
```

```
-1
-1
 1
-1
```

5 Specification file

The program COPL_SDP uses several parameters that affect its performance and output. The user can change the default parameter values by modifying the specification file **COPLSDP.spc** . An example of this file is shown below with the default values for the specifications. The symbol `%` can be placed in any line to make it a comment to be ignored by the program.

```
min_or_max(min/max)                                  : max
stopping_tolerance                                   : 1.0e-3
%dual_objective_stop                                 : 0
maximum_iterations                                   : 200
initial_dual_solution(automatic/filename)            : automatic
%constant_rho_factor(>1)                              : 1.3
create_laplacian(no/yes)                             : yes
dense_constraint_threshold(0,1)                      : 0.5
pivoting_tolerance                                   : 1.0e-13
rank_reduction_type(maxcut/maxcover/boxqp/ecut/user): maxcut
rank_one_method(no/xrow/rand/one/all)                : rand
save_rank_one_solution(no/yes)                       : no
save_dual_solution(no/yes)                           : no
print_transcript_to_file(yes/no)                     : no
```

1. `min_or_max(min/max)`: The user can input a minimization problem of the form (1) or a maximization problem of the form:

$$\begin{array}{ll} \text{Maximize} & C \bullet X + c^T x \\ \text{Subject to} & (a_i a_i^T) \bullet X + \hat{a}_i^T x = b_i, \quad i = 1, \ldots, m. \\ & X \succeq 0, \ x \geq 0. \end{array} \tag{3}$$

   The dual of this maximization problem is:

$$\begin{array}{lll} \text{Minimize} & b^T y & \\ \text{Subject to} & \sum_{i=0}^{m} (a_i a_i^T) y_i - S = C, & S \succeq 0 \\ & \sum_{i=0}^{m} \hat{a}_i y_i + s = c, & s \geq 0 \end{array} \tag{4}$$

   If your problem is of form (1), select `min`. If your problem is of form (3), select `max`.

2. `stopping_tolerance`: When the relative duality gap falls below is value, the algorithm terminates. The value should be between 0 and 1. The default value of 1.0e-3 is sufficient for most combinatorial problems. In this program the relative duality gap is defined to be the difference between the primal and dual objective value divided by the absolute value of their mean plus one.

3. `dual_objective_stop`: An alternative stopping criteria may also be specified. The program can be terminated when the dual solution crosses the value specified by this parameter. If the problem entered is a minimization problem of the form (1), the program will stop if the dual function ever becomes greater that this value. For maximization problems of the form (3), the program will stop if the dual objective function is ever less that this value. By default, this option is not activated by denoting the line a comment.

4. `maximum_iterations`: This parameter places an upper bound on the number of iterations computed by the program.

5. `initial_dual_solution(automatic/filename)`: The program can either accept an initial dual point from the user or try to generate one automatically. The default option is `automatic`, which will try to find an initial point by looking for constraints that bound the diagonal of the matrix X. The user may also create their own starting point and let the program read it from a file. This file, created by the user, should have one line for each dual variable. Each line should contain one real number. To read an initial point from a file, replace the default setting of `automatic` with the name of the file with the initial point. For more information concerning this option, see Section 6

6. `constant_rho_factor(>1)` An important parameter in the algorithm is ρ. This parameter, by default, is generated automatically and changes from one iteration to the next. However, the user can specify the parameter in the specification file by entering a multiple of $n+p$. This multiple will be used in all iterations. The right selection of this parameter can reduce the time for each iteration and the number of iterations required. Experience indicates that values near `1.3--1.9` work fairly well. You may set this factor to `1.9` for solving maxcut and ecut type problems; while set it to `1.3` for solving maxcover type problems. By default, the program will automatically generate appropriate values of ρ.

7. `cache_memory_size`: This integer specifies the kilobytes of cache memory in your machine. If this option is 0, the whole memory will be assumed to be cache memory. If you are not sure the cache memory size of your machine, it is better to give a relatively small integer.

8. `create_laplacian(no/yes)`: If this option is set to `yes`, the program will create the Laplacian matrix from the objective matrix entered by the user and use it internally as the objective matrix. The Laplacian matrix of a matrix G is defined to be

$$C = \frac{1}{4}\left(Diag(G*e) - G\right)$$

where e is the vector of ones and the operator $Diag()$ takes a vector and makes it a diagonal matrix. For maximum/minimum cut problems, the user can enter the adjacency matrix of the graph as C and use this option to generate the formulate the problem. The default, this option is `off`, which does not modify the objective matrix.

9. `dense_constraint_threshold`: The program is able to exploit sparsity in the algorithm in several steps of the program to reduce the computation time. One constraint, however, can make the dual matrix S, completely dense and remove the advantages that sparsity offers. By setting this parameter to a value in the interval $(0,1)$, the program will look for constraints with more than this fraction of its entries that are nonzero and treat them separately to retain the advantages of sparsity. By default, this parameter is 0.5. If your problem does not offer any sparsity that can be exploited, this option should be set to 1.0.

10. `pivoting_tolerance` The pivoting tolerance used in matrix decomposition should be between 0 and 1. The default value is 1.0e-13.

11. `rank_reduction_type` Some positive semidefinite programs are the relaxation of combinatorial optimization problems. In these problems, the program can use the solution to the semidefinite program to find a vector that approximately solves the unrelaxed optimization methods. By decomposing the primal solution matrix X as $X = VV^T$ and selecting a vector from the unit sphere, let $\hat{v} = Vu$. If the user specifies `maxcut`, the program defines $v = sign(\hat{v})$.

    If the user specifies `boxqp`, the program defines $v = sign(\hat{v}) \circ d$ where d is the vector with entries equal to the square root of the diagonal of the matrix X and $\circ$ means componentwise multiplication.

    If the user specifies `ecut`, the program will look for a constraint of the form $(ee^T) \bullet (X) = \kappa^2$. and seek to satisfy $v \in \{-1,1\}^n$ and $e^T v = \kappa$. Problems that contain this kind of constraint are often called graph bisection, equal cut, or unequal cut problems.

    If the user selects `maxcover`, the program will look to satisfy the constraints of the maximum cover problem. These problems have the require that $v \in \{-1,1\}^n$ such that $|v_i + v_j + v_n| = 1$ if the pair (i,j) represents an edge or link of a graph with $n-1$ vertices.

    Finally, the user can specify their own method of determining a vector v from $\hat{v}$. If the user specifies the option `user`, the program will call the function

    ```
    void UserRule(double * vhat, double * v, int n)
    ```

    that is located in the file **sdpuser.c**. This file can be modified by the user to implement their own rule. The routines's input variables are the array of doubles `vhat` and the dimension of the array, `n`. Upon entering the routine, the vector `v` should be a positive vector with entries corresponding to the diagonal of the primal solution matrix X. The user will then define the routines's output array `v` of `n` doubles.

    With any of the rules listed above, the computer will evaluate the objective function $C \bullet (vv^T)$, save the vector with the best objective value. By default, this specification is set to `no` so the program will terminate after the positive semidefinite problem is solved and before this rank reduction algorithm occurs.

12. `rank_one_method`: The vectors $\hat{v}$ mentioned above can be found in several ways. The option `rand` will randomly select n vectors u from the unit sphere. The option `xrow` will use the values of $\hat{v}$ that correspond to each row of the primal solution matrix X. Finally, selecting `one` will select $\hat{v}$ that approximates the eigenvector corresponding to the largest eigenvalue

of the primal matrix. The vector is not the exact eigenvector; it is found by using several power method iterations. If the user selects `all`, the program will use all three of the above methods to find a vector $\hat{v}$.

13. `save_rank_one_solution` If this option is `yes`, the program will save this solution in a file called *inputfile*.vsol. It will write the solution, in order, one entry per line. If `rank_reduction_type` is set to `maxcut`, `maxcover`, or `ecut`, the output be integers of `-1` or `1`. Otherwise, the output with be floating point decimals.

14. `save_dual_solution` If this parameter is `on`, an approximate dual solution will be saved into the filename *inputfile*.**ysol**. If the parameter is set to `off`, the information will not be saved to the file.

15. `print_transcript_to_file(yes/no)`: If this option is set to `yes`, a copy of the standard output will be written to a file called *inputfile*.**rec**.

6 Initial Starting Point

The program will try to find an initial feasible starting point automatically. The heuristic for finding this point tries to locate constraints that bound the diagonal of the primal matrix above and then creates a dual matrix that is diagonally dominant. If such diagonal constraints are not present, or if the constraint matrices $\hat{a}_i$ are not null or simple slack vectors, the point generated by the program may not be feasible. For the program to solve the problem entered, the user must provide a starting point y that creates a matrix S that is positive definite and a vector s that is positive.

The file that contains the initial point should contain m lines with one entry on each line. The ordering of the dual variables should correspond with the ordering of the matrix constraints in the input file. The specification file should also contain the option

```
initial_dual_solution(automatic/filename) :
```

with the filename containing the point listed to the right of the colon.

7 Primal Infeasibility

In some problems, the primal feasible set defined by (1) or (3) does not have a interior. As an example, consider the equal cut problem. This problem has a constraint of the form $(ee^T) \bullet X = 0$. The only positive semidefinite matrices that satisfy this constraint are singular. For these problems, the algorithm may not converge. The program displays the norm of the dual point at every iteration. If the norm gets large and the program does not converge, your problem will have to be reformulated to create an primal interior.

One remedy is called the "big M" method. To create an interior, add an artificial variable x and a large negative penalty parameter M, to the objective function. In case of the equal cut problem, formulate it as,

$$\begin{array}{lll} \text{Maximize} & C \bullet X + Mx & \\ \text{Subject to} & \text{diag(X)} & = e, \\ & (ee^T) \bullet X - x & = 0. \\ & X \succeq 0, x \geq 0 & \end{array} \tag{5}$$

The penalty parameter M forces x to be very small so the original constraint is almost satisfied.

8 Sample Problems

Included among the files proved are five sample data files. The file **MaxTest** contains a small maximum cut problem. This file was shown in Section 3. It has variable matrices of size six and has six constraints. Each constraint sets a diagonal element of the matrix X equal to one. When it is solved as a maximization problem and the program creates the Laplacian matrix of the graph, the optimal solution should be 9.25.

The file **MaxG11** describes the maximum cut problem for the G-set graph G11. The entire collection of more than 60 Gset problems can be found at the *Computational Optimization Laboratory* web-site. These problems are a standard set of test problems that represent graphs with 800 to 10000 vertices and a varying number of edges. This particular graph contains 800 vertices. Solved as a maximization problem with the Laplacian matrix, the SDP relaxation upper bound is 629.19 and the best cut value is 534.

The file **EcutG11** describes the max equal-cut problem for the graph G11. Note that this file uses the "big M" method described in the previous section in formulating the problem. The solution, when solved in the same manner as **MaxG11**, has an SDP relaxation upper bound of 629.17. For an indication of how much sparsity can be exploited, solve this problem with the option `dense_constraint_threshold(0,1) :  0.5` and then solve it again after changing the parameter to `1.0`. The first choice of option will treat the one dense constraint in this problem separately from the sparse constraints while the latter choice of option will treat it in an identical manner. Solved as a maximization problem with the Laplacian matrix, the best equal-cut value is 532. Note that for solving this problem the option in COPLSDP.spc is

```
...
rank_reduction_type(maxcut/maxcover/boxqp/ecut/user): ecut
...
```

The file **BoxQpG11** describes the boxqp problem. This problem comes from the positive semidefinite relaxation of the problem $v^T Cv$ such that $-e \leq v \leq e$. In this problem, the diagonals of the matrix X are bounded above by one. When solved as a maximization problem with a Laplacian matrix, the SDP upper bound is $6.349e+02$ and the best vector solution value is 547.09. Note that for solving this problem the option in COPLSDP.spc is

```
...
```

```
rank_reduction_type(maxcut/maxcover/boxqp/ecut/user): boxqp
...
```

The file **MinEcutG11** describes the equal cut problem for the graph G11 with all edge weights are equal to 1, and the number of cuts is to be minimized. This is called the graph-partitioning problem. This file also uses the "big M" method described in the previous section in formulating the problem. When solved as a minimization problem with a Laplacian matrix, the SDP lower bound is .7893 and the best cut value is 16. Note that for solving this problem the option in COPLSDP.spc is

```
min_or_max(min/max)                                 : min
create_laplacian(no/yes)                            : yes
...
rank_reduction_type(maxcut/maxcover/boxqp/ecut/user): ecut
...
```

The file **MaxCoverTest** gives an example of a max covering problem. This problem has one hundred vertices, so the dimension of the positive semidefinite program is 101. There are 526 links in the graph which makes a total of 627 constraints. When solved as a maximization problem without using the Laplacian, the optimal objective value is 33, which happened to be the exact optimal cover value. Note that for solving this problem the option in COPLSDP.spc is

```
min_or_max(min/max)                                 : max
create_laplacian(no/yes)                            : no
...
rank_reduction_type(maxcut/maxcover/boxqp/ecut/user): maxcover
...
```

The file **StableG00** gives an example of a min stable set problem. This problem has 64 vertices, so the dimension of the positive semidefinite program is 65. There are 107 links in the graph which makes a total of 172 constraints. When solved as a maximization problem without using the Laplacian, the optimal objective value is 2, which, added by $n/2 = 32$, happened to be the exact minimal stable set with 34 vertices or the maximal cover set with 30 vertices. Note that for solving this problem the option in COPLSDP.spc is

```
min_or_max(min/max)                                 : min
create_laplacian(no/yes)                            : no
...
rank_reduction_type(maxcut/maxcover/boxqp/ecut/user): maxcover
...
```

More test problems can be found on the side "ftp://dollar.biz.uiowa.edu/pub/yyye/Gset/".