ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

User's Manual for SparseCoLO: Conversion Methods for **SPARSE CO**nic-form **L**inear **O**ptimization Problems

K. Fujisawa, S. Kim, M. Kojima,
Y. Okamoto, and M. Yamashita

February 2009, B–453
Revised September 2009

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-453 User's Manual for **SparseCoLO**: Conversion Methods for **SPARSE CO**nic-form **L**inear **O**ptimization Problems

Katsuki Fujisawa[⋆], Sunyoung Kim[†], Masakazu Kojima[‡], Yoshio Okamoto[‡], and Makoto Yamashita[‡]

February 2009, Revised July 2009

Abstract.

SparseCoLO is a Matlab package for implementing the four conversion methods, proposed by Kim, Kojima, Mevissen, and Yamashita, via positive semidefinite matrix completion for an optimization problem with matrix inequalities satisfying a sparse chordal graph structure. It is based on a general description of optimization problem including both primal and dual form of linear, semidefinite, and second-order cone programs with equality/inequality constraints. Among the four conversion methods, two methods utilize the domain-space sparsity of a semidefinite matrix variable and the two other methods the range-space sparsity of a linear matrix inequality (LMI) constraint of the given problem. **SparseCoLO** can be used as a preprocessor to reduce the size of the given problem before applying semidefinite programming solvers. The website for this package is

http://www.is.titech.ac.jp/∼kojima/SparseCoLO

where the package **SparseCoLO** and this manual can be downloaded.

Key words.

Semidefinite program, Sparsity exploitation, Positive semidefinite matrix completion, Conversion methods, Matlab software package.

⋆ Department of Industrial and Systems Engineering, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan. *fujisawa@indsys.chuo-u.ac.jp*

† Department of Mathematics, Ewha W. University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. S. Kim's research was supported by KRF 2008-531-C00013. *skim@ewha.ac.kr*

‡ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. This work was supported by Global COE Program "Computationism as a Foundation for the Sciences" and Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science. *kojima@is.titech.ac.jp*, *okamoto@is.titech.ac.jp*, *Makoto.Yamashita@is.titech.ac.jp*

1 Introduction

We introduce a Matlab package SparseCoLO for solving a conic-form linear optimization problem (LOP) of the form:

$$\text{minimize} \quad \boldsymbol{c}^T\boldsymbol{x} \text{ subject to } \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \in \mathrm{J},\ \boldsymbol{x} \in \mathrm{K}, \tag{1}$$

where

$$\begin{aligned}
\mathrm{K} &= \mathrm{K.f} \times \mathrm{K}.\ell \times \mathrm{K.q} \times \mathrm{K.s},\\
\mathrm{K.f} &= \text{a column vector space to denote free variables},\\
\mathrm{K}.\ell &= \text{an LP cone},\\
\mathrm{K.q} &= \text{an SOCP cone or a product of SOCP cones},\\
\mathrm{K.s} &= \text{a column vector space corresponding to an SDP cone or}\\
&\quad\ \text{a product of SDP cones},\\
\mathrm{J} &= \mathrm{J.f} \times \mathrm{J}.\ell \times \mathrm{J.q} \times \mathrm{J.s},\\
\mathrm{J.f} &= \{\boldsymbol{0}\} \subset \text{a column vector space to denote equality constraints},\\
\mathrm{J}.\ell &= \text{an LP cone},\\
\mathrm{J.q} &= \text{an SOCP cone or a product of SOCP cones},\\
\mathrm{J.s} &= \text{a column vector space corresponding to an SDP cone or}\\
&\quad\ \text{a product of SDP cones},\\
\boldsymbol{c}^T &= (\boldsymbol{c}_f^T, \boldsymbol{c}_\ell^T, \boldsymbol{c}_q^T, \boldsymbol{c}_s^T),\\
\boldsymbol{A} &= \begin{pmatrix} \boldsymbol{A}_{ff} & \boldsymbol{A}_{f\ell} & \boldsymbol{A}_{fq} & \boldsymbol{A}_{fs} \\ \boldsymbol{A}_{\ell f} & \boldsymbol{A}_{\ell\ell} & \boldsymbol{A}_{\ell q} & \boldsymbol{A}_{\ell s} \\ \boldsymbol{A}_{qf} & \boldsymbol{A}_{q\ell} & \boldsymbol{A}_{qq} & \boldsymbol{A}_{qs} \\ \boldsymbol{A}_{sf} & \boldsymbol{A}_{s\ell} & \boldsymbol{A}_{sq} & \boldsymbol{A}_{ss} \end{pmatrix},\ \boldsymbol{b} = \begin{pmatrix} \boldsymbol{b}_f \\ \boldsymbol{b}_\ell \\ \boldsymbol{b}_q \\ \boldsymbol{b}_s \end{pmatrix}.
\end{aligned}$$

The main feature of SparseCoLO is that it can detect and utilize two types of sparsities, characterized in terms of a sparse chordal graph structure, in the variable $\boldsymbol{x} \in \mathrm{K}$ and in the constraint $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \in \mathrm{J}$ to reduce the size of the problem. It can be used as a preprocessor to reduce the size of (1) before applying semidefinite programming solvers such as CSDP [2], SDPA [4], SDPT3 [10], SeDuMi [9]. Currently, one of three SDP solvers, SeDuMi, SDPA or SDPT3, can be selected in SparseCoLO to solve the resulting problem after the conversion methods are applied to the problem (1).

The problem (1) indeed represents various problems. For instance, if $\mathrm{J}.\ell = \emptyset$, $\mathrm{J.q} = \emptyset$ and $\mathrm{J.s} = \emptyset$, (1) includes a primal form LOP (an equality standard form LOP) in the SeDuMi format [9] as a special case:

$$\text{minimize} \quad \boldsymbol{c}^T\boldsymbol{x} \text{ subject to } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \in \mathrm{K}.$$

In the case that $\mathrm{K}.\ell = \emptyset$, $\mathrm{K.q} = \emptyset$ and $\mathrm{K.s} = \emptyset$, the problem (1) becomes a dual form LOP (an LMI standard form LOP) in the SeDuMi format:

$$\text{maximize} \quad -\boldsymbol{c}^T\boldsymbol{x} \text{ subject to } \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \in \mathrm{J},$$

where $\boldsymbol{x}$ denotes a free variable vector. Thus, we may regard the conic-form LOP (1) as a unified extension of primal and dual standard form LOPs. This form of LOP will provide not only flexibility in modeling practical problems in terms of LOPs, but also a convenient framework for the domain- and range-space conversion methods proposed in the paper [5].

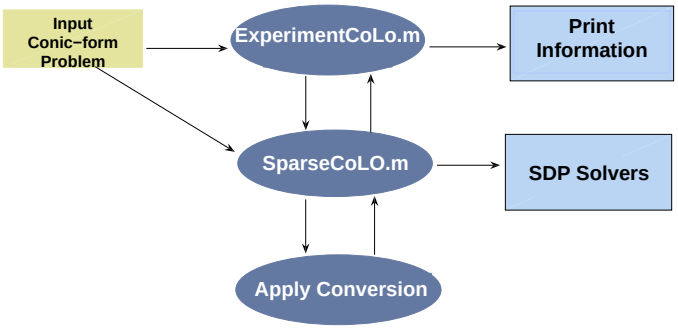

Figure 1: The structure of SparseCoLO

We can rewrite the conic-form LOP (1) as

$$\left.\begin{array}{ll} \text{minimize} & \boldsymbol{c}_f^T\boldsymbol{x}_f + \boldsymbol{c}_\ell^T\boldsymbol{x}_\ell + \boldsymbol{c}_q^T\boldsymbol{x}_q + \boldsymbol{c}_s^T\boldsymbol{x}_s \\ \text{subject to} & \boldsymbol{A}_{ff}\boldsymbol{x}_f + \boldsymbol{A}_{f\ell}\boldsymbol{x}_\ell + \boldsymbol{A}_{fq}\boldsymbol{x}_q + \boldsymbol{A}_{fs}\boldsymbol{x}_s - \boldsymbol{b}_f \in \text{J.f}, \\ & \boldsymbol{A}_{\ell f}\boldsymbol{x}_f + \boldsymbol{A}_{\ell\ell}\boldsymbol{x}_\ell + \boldsymbol{A}_{\ell q}\boldsymbol{x}_q + \boldsymbol{A}_{\ell s}\boldsymbol{x}_s - \boldsymbol{b}_\ell \in \text{J.}\ell \\ & \boldsymbol{A}_{qf}\boldsymbol{x}_f + \boldsymbol{A}_{q\ell}\boldsymbol{x}_\ell + \boldsymbol{A}_{qq}\boldsymbol{x}_q + \boldsymbol{A}_{qs}\boldsymbol{x}_s - \boldsymbol{b}_q \in \text{J.q} \\ & \boldsymbol{A}_{sf}\boldsymbol{x}_f + \boldsymbol{A}_{s\ell}\boldsymbol{x}_\ell + \boldsymbol{A}_{sq}\boldsymbol{x}_q + \boldsymbol{A}_{ss}\boldsymbol{x}_s - \boldsymbol{b}_s \in \text{J.s}, \\ & \boldsymbol{x}_f \in \text{K.f},\ \boldsymbol{x}_\ell \in \text{K.}\ell,\ \boldsymbol{x}_q \in \text{K.q},\ \boldsymbol{x}_s \in \text{K.s}. \end{array}\right\} \tag{2}$$

We use the quintuplet $(\boldsymbol{A},\ \boldsymbol{b},\ \boldsymbol{c},\ \text{K},\ \text{J})$ to denote the conic-form linear optimization problem (2).

The dual of (2) above is of the form

$$\begin{array}{ll} \text{minimize} & -\boldsymbol{b}_f^T\boldsymbol{y}_f - \boldsymbol{b}_\ell^T\boldsymbol{y}_\ell - \boldsymbol{b}_q^T\boldsymbol{y}_q - \boldsymbol{b}_s^T\boldsymbol{y}_s \\ & \boldsymbol{c}_f - \boldsymbol{A}_{ff}^T\boldsymbol{y}_f - \boldsymbol{A}_{\ell f}^T\boldsymbol{y}_\ell - \boldsymbol{A}_{qf}^T\boldsymbol{y}_q - \boldsymbol{A}_{sf}^T\boldsymbol{y}_s \in \text{K.f}^*, \\ & \boldsymbol{c}_\ell - \boldsymbol{A}_{f\ell}^T\boldsymbol{y}_f - \boldsymbol{A}_{\ell\ell}^T\boldsymbol{y}_\ell - \boldsymbol{A}_{q\ell}^T\boldsymbol{y}_q - \boldsymbol{A}_{s\ell}^T\boldsymbol{y}_s \in \text{K.}\ell, \\ & \boldsymbol{c}_q - \boldsymbol{A}_{fq}^T\boldsymbol{y}_f - \boldsymbol{A}_{\ell q}^T\boldsymbol{y}_\ell - \boldsymbol{A}_{qq}^T\boldsymbol{y}_q - \boldsymbol{A}_{sq}^T\boldsymbol{y}_s \in \text{K.q}, \\ & \boldsymbol{c}_s - \boldsymbol{A}_{fs}^T\boldsymbol{y}_f - \boldsymbol{A}_{\ell s}^T\boldsymbol{y}_\ell - \boldsymbol{A}_{qs}^T\boldsymbol{y}_q - \boldsymbol{A}_{ss}^T\boldsymbol{y}_s \in \text{K.s}, \\ & \boldsymbol{y}_f \in \text{J.f}^*,\ \boldsymbol{y}_\ell \in \text{J.}\ell,\ \boldsymbol{y}_q \in \text{J.q},\ \boldsymbol{y}_s \in \text{J.s}, \end{array}$$

where K.f*($= \{\boldsymbol{0}\}$) is the dual cone of K.f, that is, the space of column vectors for equality constraints, and similarly, J.f*(= the Euclidean space with the same dimension) is the dual cone of J.f, the space of column vectors for free variables.

The package SparseCoLO contains sparseCoLO.m as the main function, experimentCoLO.m and evaluateCoLO.m for numerical experiments, a set of test problems in the directory examples, and the functions for conversion methods in the directory conversion. The structure of SparseCoLo is illustrated in Figure 1.

In the program sparseCoLO.m, we denote the dual LOP as the quintuplet $(-\boldsymbol{A}',\ -\boldsymbol{c},\ -\boldsymbol{b},\ \text{J},\ \text{K})$, where $\boldsymbol{A}'$ denotes the transpose of the matrix $\boldsymbol{A}$. Under the assumption that the conic-form LOP (2) satisfies a sparsity characterized by a chordal graph structure, sparseCoLO.m

1. transforms the conic-form LOP (2) into another LOP of smaller size by applying the domain- and range-space conversion methods developed in [5], and then
2. calls SeDuMi [9], SDPA [4] or SDPT3 [10] to solve the resulting LOP efficiently.

In Section 2, we describe how to select the conversion methods using parameters. The usage of SparseCoLO is illustrated with commands in Section 3. Section 4 includes the description of the parameters and output. Section 5 summarizes the manual. In Appendix, we demonstrate the effectiveness of SparseCoLO using the max-cut problem, the norm minimization problem, the sensor network localization problem [1] and some SDPs from sdplib [8].

2 The domain- and range-space conversion methods

SparseCoLO includes implementation of the four conversion methods developed in [5]:

(a) The d-space conversion method using clique trees. See Section 3.1 of [5].

(b) The d-space conversion method using basis representation. See Section 3.2 of [5].

(c) The r-space conversion method using clique trees. See Section 5.1 of [5].

(d) The r-space conversion method using matrix decomposition. See Section 5.2 of [5].

We can choose either (a) or (b) by specifying the parameter parCoLO.domain = 1 or parCoLO.domain = 2, respectively, while either (c) or (d) can be chosen by specifying the parameter parCoLO.range = 1 or parCoLO.range = 2, respectively. If no conversion is used, both of the parameters should be set to 0.

After applying the conversion methods specified by the parameters parCoLO.domain and parCoLO.range, the resulting LOP is further converted into an equality standard form or a linear matrix inequality (LMI) standard form for SeDuMi, SDPA or SDPT3 to solve the problem with parCoLO.EQorLMI = 1 or parCoLO.EQorLMI = 2.

3 Examples and Sample execution

Let us consider a norm minimization problem

$$\text{minimize} \quad \left\| \boldsymbol{F}_0 + \sum_{k=1}^{r} \boldsymbol{F}_k z_k \right\|_2 \quad \text{subject to } \|\boldsymbol{z}\| \leq 1.$$

Here $\boldsymbol{z} = (z_1, z_2, \ldots, z_r) \in \mathbb{R}^r$ denotes a variable vector, $\|\boldsymbol{z}\|$ the Euclidean norm of $\boldsymbol{z}$, $\boldsymbol{F}_k$ a $p \times q$ matrix and $\|\boldsymbol{F}\|_2$ the operator norm of $\boldsymbol{F}$. We can formulate this problem as an LOP:

$$\begin{array}{ll} \text{minimize} & \zeta \\ \text{subject to} & z_0 = 1, \\ & \displaystyle\sum_{k=1}^{r} \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_k \\ \boldsymbol{F}_k^T & \boldsymbol{O} \end{pmatrix} z_k + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} \zeta + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0 \\ \boldsymbol{F}_0^T & \boldsymbol{O} \end{pmatrix} \\ & \qquad \in \text{an SDP cone with dimension } (p+q), \\ & \zeta \in \mathbb{R},\ (z_0, z_1, z_2, \ldots, z_r) \in \text{an SOCP cone with dimension } r. \end{array}$$

Thus, letting K.f = 1, K.ℓ = [ ], K.q = [1 + r], K.s = [ ], J.f = 1, J.ℓ = [ ], J.q = [ ], J.s = [q + r], we can describe the conic-form LOP above as in (2). Not that the coefficient matrices of the conic-form LOP become sparse when q is small.

The SparseCoLO package contains a MATLAB program to generate the norm minimization problem. For example, type

```
>> [A,b,c,K,J] = normMinSDP(200,5,10,2009);
```

to load the data $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$, K, J for the conic-form LOP formulation of the norm minimization problem with $p = 200$, $q = 5$, and $r = 10$. We note that the last input, 2009 in the example, specifies a seed for the random number generator. Then, type

```
>> parCoLO.domain = 0; parCoLO.range = 1; parCoLO.EQorLMI = 1;
```

to specify the parameters for no conversion in the domain space, the r-space conversion using clique trees and the conversion into an equality standard form. (The formulated LOP ($\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$, K, J) does not involve any variable matrix, so that the domain-space conversion is not relevant). Finally, issue a command to solve the norm minimization problem as follows:

```
>> [x,y,infoCoLO,cliqueDomain,cliqueRange,LOP] = sparseCoLO(A,b,c,K,J,parCoLO);
parCoLO.domain = 0; parCoLO.range = 1; parCoLO.EQorLMI = 1
Apply the r-space conversion method using clique trees.
Conversion into an equality standard form
SeDuMi 1.1R3 by AdvOL, 2006 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, Adaptive Step-Differentiation, theta = 0.250, beta = 0.500
Split 241 free variables
eqs m = 2540, order n = 770, dim = 5287, blocks = 19
nnz(A) = 13910 + 0, nnz(ADA) = 1365658, nnz(L) = 868957
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            2.70E-02 0.000
  1 :   3.12E+01 5.47E-03 0.000 0.2027 0.9000 0.9000  -0.95  1  1  4.7E+00
  2 :   1.00E+02 1.03E-03 0.000 0.1878 0.9000 0.9000  -0.72  1  1  2.7E+00
  3 :   5.33E+01 2.61E-04 0.000 0.2539 0.9000 0.9000   0.66  1  1  6.5E-01
  4 :   2.81E+01 1.05E-04 0.000 0.4035 0.9000 0.9000   2.31  1  1  1.4E-01
  5 :   1.26E+01 2.83E-05 0.000 0.2696 0.9000 0.9000   2.83  1  1  1.7E-02
  6 :   9.55E+00 8.49E-06 0.000 0.2996 0.9000 0.9000   1.75  1  1  4.0E-03
  7 :   9.17E+00 2.46E-06 0.000 0.2897 0.9000 0.9000   1.14  1  1  1.1E-03
  8 :   9.09E+00 5.70E-07 0.000 0.2317 0.9000 0.9000   1.02  1  1  2.6E-04
  9 :   9.08E+00 3.54E-08 0.000 0.0620 0.9000 0.5415   1.01  1  1  9.7E-05
 10 :   9.07E+00 5.41E-09 0.000 0.1531 0.9105 0.9000   1.00  1  1  1.7E-05
 11 :   9.07E+00 1.84E-10 0.340 0.0339 0.9903 0.9900   1.00  1  1  6.1E-07
 12 :   9.07E+00 7.61E-12 0.103 0.0414 0.9675 0.9683   1.00  2  2  2.5E-08
 13 :   9.07E+00 4.97E-13 0.476 0.0653 0.9900 0.9900   1.00  2  2  1.6E-09
 14 :   9.07E+00 9.58E-14 0.000 0.1929 0.9011 0.9000   1.00  3  3  3.1E-10

iter seconds digits       c*x               b*y
 14      8.1   Inf  9.0699974057e+00  9.0699974538e+00
|Ax-b| =   4.1e-10, [Ay-c]_+ =   1.5E-10, |x|=  2.1e+02, |y|=  1.6e+00

Detailed timing (sec)
   Pre          IPM          Post
6.300E-01    8.120E+00    8.000E-02
Max-norms: ||b||=1.997582e+00, ||c|| = 1,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 46.1161.
```

The last input argument parCoLO can be omitted. Then, sparseCoLO.m calls defaultParCoLO to select suitable conversion methods for the given LOP ($\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \mathrm{K}, \mathrm{J}$). For example, type

```
>>  [A,b,c,K,J] = normMinSDP(200,5,10,2009);
>> [x,y,infoCoLO,cliqueDomain,cliqueRange,LOP] = sparseCoLO(A,b,c,K,J);
```

Then,

```
parCoLO.domain = 0; parCoLO.range = 2; parCoLO.EQorLMI = 1
```

```
Also try:
    parCoLO.domain = 0; parCoLO.range = 1; parCoLO.EQorLMI = 2

Apply the r-space conversion method using matrix decomposition.
LOP to be converted into equality standard form is already equality standard form.
SeDuMi 1.1R3 by AdvOL, 2006 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, Adaptive Step-Differentiation, theta = 0.250, beta = 0.500
Split 1 free variables
eqs m = 1216, order n = 1205, dim = 7214, blocks = 202
nnz(A) = 4610 + 10001, nnz(ADA) = 83225, nnz(L) = 42221
Handling 11 + 1 dense columns.
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            2.32E+01 0.000
  1 :   1.46E+02 7.85E-01 0.000 0.0339 0.9900 0.9900  -0.95  1  1  2.3E+02
  2 :   8.48E+01 5.28E-01 0.000 0.6729 0.9000 0.9000   1.41  1  1  9.7E+01
  3 :   5.02E+01 3.21E-01 0.000 0.6071 0.9000 0.9000   3.79  1  1  2.4E+01
  4 :   2.32E+01 1.12E-01 0.000 0.3500 0.9000 0.9000   3.60  1  1  3.0E+00
  5 :   1.13E+01 3.79E-02 0.000 0.3380 0.9000 0.9000   3.07  1  1  4.7E-01
  6 :   9.32E+00 1.09E-02 0.000 0.2880 0.9000 0.9000   1.55  1  1  1.1E-01
  7 :   9.16E+00 3.82E-03 0.000 0.3495 0.9000 0.9000   1.09  1  1  3.8E-02
  8 :   9.12E+00 1.82E-03 0.000 0.4759 0.9000 0.9000   1.03  1  1  1.8E-02
  9 :   9.09E+00 5.11E-04 0.000 0.2811 0.9031 0.9000   1.01  1  1  5.4E-03
 10 :   9.08E+00 7.79E-05 0.000 0.1525 0.9243 0.9000   1.00  1  1  1.3E-03
 11 :   9.07E+00 1.63E-06 0.000 0.0210 0.9266 0.9000   1.00  1  1  1.0E-04
 12 :   9.07E+00 4.49E-09 0.000 0.0027 0.9906 0.9900   1.00  1  1  1.3E-06
 13 :   9.07E+00 8.72E-10 0.123 0.1942 0.9000 0.6601   1.00  1  2  3.5E-07
 14 :   9.07E+00 1.94E-11 0.000 0.0222 0.9901 0.9900   1.00  1  1  9.4E-09
 15 :   9.07E+00 1.59E-12 0.412 0.0820 0.9900 0.9900   1.00  3  3  7.8E-10

iter seconds digits       c*x               b*y
 15      2.5   Inf  9.0699974751e+00  9.0699976838e+00
|Ax-b| =   9.1e-10, [Ay-c]_+ =   3.1E-10, |x|=  1.3e+02, |y|=  5.9e-01

Detailed timing (sec)
   Pre          IPM          Post
2.800E-01    2.500E+00    2.000E-02
Max-norms: ||b||=1.997582e+00, ||c|| = 1,
Cholesky |add|=0, |skip| = 2, ||L.L|| = 765.641.
```

If one wants to know the objective values and the feasibility errors for the solution obtained, type

```
>> [primalObjValue, dualObjValue, primalfeasibility, dualfeasibility] = ...
              evaluateCoLO(x,y,A,b,c,K,J,cliqueDomain,cliqueRange)
primalObjValue =  9.0700
dualObjValue =  9.0700
primalfeasibility =   2.1253e-11
dualfeasibility =   6.5280e-09
```

For application of sparseCoLO.m to the LOP provided by maxG11.mat, type

```
>> parCoLO.domain = 2; parCoLO.range = 0; parCoLO.EQorLMI = 2;
```

to specify the parameters for the d-space conversion method using basis representation, no conversion in the range space, and the conversion into an LMI standard form. Then

```
>> load 'maxG11.mat';
>> clear J; J.f = size(A,1);
>> [x,y,infoCoLO,cliqueDomain,cliqueRange,LOP] = sparseCoLO(A,b,c,K,J,parCoLO);
```

We have a primal approximation to an optimal solution $\boldsymbol{x}$ (or a dual approximation to an optimal solution $\boldsymbol{y}$) of the LOP. It should be noted that the $\boldsymbol{x}$ (or $\boldsymbol{y}$) involves only partial elements of an (approximate) optimal solution vector because the d-space conversion method using basis representation is applied. In fact, we have

```
>> size(x), nnz(x)
ans = 640000           1
ans = 15866
```

Notice that all the elements with value 0 have not been determined because they are not relevant to the objective and constraint functions. To retrieve all the elements of the (approximate) optimal solution vector, type

```
>> [x] = psdCompletion(x,K,cliqueDomain);
the minimum eigenvalue of a completed SDP variable matrix = +2.9e-09
```

Then,

```
>> size(x), nnz(x)
ans = 640000           1
ans = 640000
```

All unspecified elements are completed, thus, the variable matrices become (approximately) positive semidefinite;

```
>> XMat = reshape(x,800,800); min(eig(XMat))
ans =  2.8664e-09
```

See the paper [3, 5, 7] for more details.

We can apply a MATLAB version of SDPA [4] to solve an equality standard form SDP instead of SeDuMi. To solve a randomly generated max cut problem, type

```
>> [A,b,c,K,J] = maxCutSDP(1,200,4,2009);
>> parCoLO.SDPsolver = 'sdpa';
>> parCoLO.sdpaOPTION.print = '';
>> [x,y,infoCoLO,cliqueDomain,cliqueRange] = sparseCoLO(A,b,c,K,J,parCoLO);
parCoLO.domain = 1; parCoLO.range = 0; parCoLO.EQorLMI = 1
Apply the d-space conversion method using clique trees.
LOP to be converted into equality standard form is already equality standard form.
-SeDuMi Wrapper for SDPA Start-
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
>> [primalObjValue, dualObjValue, primalfeasibility, dualfeasibility] = ...
    evaluateCoLO(x,y,A,b,c,K,J,cliqueDomain,cliqueRange);
primalObjValue    = -1.35180821e+02, dualObjValue = -1.35180823e+02, gap = +2.72e-06
primalfeasibility = +2.75e-13
dualfeasibility   = +0.00e+00
```

We note that the current version of SDPA can not handle SOCP cones.

SDPT3 [10] can be used to solve an equality standard form SDP instead of SeDuMi. To solve a randomly generated max cut problem using SDPT3, type

```
>> [A,b,c,K,J] = maxCutSDP(1,200,4,2009);
>> parCoLO.SDPsolver = 'sdpt3';
>> parCoLO.sdpt3OPTIONS.printlevel = 2;
>> [x,y,infoCoLO,cliqueDomain,cliqueRange] = sparseCoLO(A,b,c,K,J,parCoLO);

SparseCoLO 1.11
by K.Fujisawa, S.Kim, M.Kojima, Y.Okamoto and M. Yamashita,
September 2009

parCoLO.domain = 1; parCoLO.range = 0; parCoLO.EQorLMI = 1
Apply the d-space conversion method using clique trees.
LOP to be converted into equality standard form is already equality standard form.

 num. of constraints = 506
 dim. of sdp    var  = 284,   num. of sdp  blk  = 16
*****************************************************************
   SDPT3: Infeasible path-following algorithms
*****************************************************************

  stop: max(relative gap, infeasibilities) < 1.00e-08
-----------------------------------------------------------------
 number of iterations   = 14
 primal objective value = -1.35180823e+02
 dual   objective value = -1.35180823e+02
 gap := trace(XZ)       = 6.62e-08
 relative gap           = 2.44e-10
 actual relative gap    = 2.45e-10
 rel. primal infeas     = 2.33e-12
 rel. dual   infeas     = 1.50e-12
 norm(X), norm(y), norm(Z) = 1.2e+01, 4.4e+01, 4.9e+01
 norm(A), norm(b), norm(C) = 3.6e+01, 4.5e+00, 3.5e+01
 Total CPU time (secs)  = 3.0
 CPU time per iteration = 0.2
 termination code       =  0
 DIMACS: 8.4e-12  0.0e+00  2.6e-11  0.0e+00  2.4e-10  2.4e-10
-----------------------------------------------------------------
>> [primalObjValue, dualObjValue, primalfeasibility, dualfeasibility] = ...
    evaluateCoLO(x,y,A,b,c,K,J,cliqueDomain,cliqueRange);
primalObjValue    = -1.35180823e+02, dualObjValue = -1.35180823e+02, gap = +6.64e-08
primalfeasibility = +1.12e-12
dualfeasibility   = +0.00e+00
```

The package SparseCoLO contains a MATLAB program experimentCoLO.m for numerical experiments using sparseCoLO.m. To use this program, "parameterSet" needs to be specified in the program. It is a $k \times 3$ matrix with each row of values for parCoLO.domain, parCoLO.range and parCoLO.EQorLMI. For example, let

```
>> parameterSet = [0,0,1; 1,0,1; 2,0,2];
```

Then,

```
>> load 'maxG11.mat';
>> J.f = size(A,1);
>> parCoLO.SDPsolver = 'sedumi';
>> experimentCoLO(A,b,c,K,J,parCoLO,parameterSet);

SparseCoLO 1.11
by K.Fujisawa, S.Kim, M.Kojima, Y.Okamoto and M. Yamashita,
September 2009

parCoLO.domain = 0; parCoLO.range = 0; parCoLO.EQorLMI = 1
LOP to be converted into equality standard form is already equality standard form.
primalObjValue    = -6.29164783e+02, dualObjValue = -6.29164783e+02, gap = -3.47e-10
primalfeasibility = +9.56e-11
dualfeasibility   = +2.68e-10

SparseCoLO 1.11
by K.Fujisawa, S.Kim, M.Kojima, Y.Okamoto and M. Yamashita,
September 2009

parCoLO.domain = 1; parCoLO.range = 0; parCoLO.EQorLMI = 1
Apply the d-space conversion method using clique trees.
LOP to be converted into equality standard form is already equality standard form.
primalObjValue    = -6.29164779e+02, dualObjValue = -6.29164779e+02, gap = -5.41e-09
primalfeasibility = +1.64e-09
dualfeasibility   = +0.00e+00

SparseCoLO 1.11
by K.Fujisawa, S.Kim, M.Kojima, Y.Okamoto and M. Yamashita,
September 2009

parCoLO.domain = 2; parCoLO.range = 0; parCoLO.EQorLMI = 2
Apply the d-space conversion method using basis representation.
LOP to be converted into LMI standard form is already LMI standard form.
primalObjValue    = -6.29164783e+02, dualObjValue = -6.29164783e+02, gap = -9.09e-12
primalfeasibility = +3.24e-11
dualfeasibility   = +0.00e+00

%by SeDuMi
%slover    parCoLO  |     cpu time |          matrix A        . . . .
%         d  r EQ/LMI  cpuC     cpuS          sizeA           . . . .
sedumi & 0   0   1 &    0.0 &  516.8  &    800 x 640000 &     . . . .
sedumi & 1   0   1 &    2.9 &   27.4  &   2432 x  76672 &     . . . .
sedumi & 2   0   2 &    1.6 &   50.9  &   8333 x  48698 &     . . . .
% max primal objective value over the  3 cases = -6.291647793586038e+02
% min primal objective value over the  3 cases = -6.291647829879826e+02
% max primal obj. value- min primal obj. value = +3.63e-06
```

For the meaning of each number for the table in the LaTeX format above, see the comments at the end of the program experimentCoLO.m.

4 Parameters and output

Tables 1 and 2 show the parameters and output of sparseCoLO.m.

Parameter	Value	Description
parCoLO.domain	0	No conversion in the domain space. If K.s = ∅, take this Value because the domain space conversion is irrelevant.
	1	The domain space conversion method using clique trees
	2	The domain space conversion method using basis representation
parCoLO.range	0	No conversion in the range space If J.s = ∅, take this value because the range space conversion is irrelevant.
	1	The range space conversion method using clique trees
	2	The range space conversion method using matrix decomposition
parCoLO.EQorLMI	1	Conversion into an equality form
	2	Conversion into an LMI form
parCoLO.SDPsolver	[]	Output information on the conversion without solving a given LOP
	'sedumi'	Solve the resulting LOP by SeDuMi — default
	'sdpa'	Solve the resulting LOP by sedumiwrap (SDPAM)
	'sdpt3'	Solve the resulting LOP by SDPT3

parCoLO.sedumipar		Parameters for SeDuMi, *e.g*,
		parCoLO.sedumipar.fid = 0, 1, 'temp.out';
		parCoLO.sedumipar.free = 1;
		parCoLO.sedumipar.eps = 1.0e-7;
		See the user guide of SeDuMi.

parCoLO.sdpaOPTION		Parameters for sedumiwrap (SDPAM), *e.g*, parCoLO.sdpaOPTION.print = '', 'display'; parCoLO.sdpaOPTION.maxIteration = 100; parCoLO.sdpaOPTION0.resultFile = 'temp.out'; See ∼/sdpa/sdpa.7.3.1/mex/param.m.

parCoLO.sdpt3OPTIONS		Parameters for SDPT3, *e.g*,
		parCoLO.sdpt3OPTIONS.printlevel = 0, 1, 2, 3;
		parCoLO.sdpt3OPTIONS.smallblkdim = 20;
		See the user guide of SDPT3.

Table 1: Input parameters for sparseCoLO.m

5 Concluding remarks

The structure and usage of the Matlab package SparseCoLO have been described using the conic-form LOP, which generalizes the representation of linear, second-order, and semidefinite equality/ inequality-constraint problems.

Output	Description
$\boldsymbol{x}$	An approximate primal optimal solution when parCoLO.method = 2
	[ ] when parCoLO.method = [ ]
$\boldsymbol{y}$	An approximate dual optimal solution when parCoLO.method = 2
	seqLOPs, which includes information of sequence of LOPs from the original LOP through the final LOP to be solved when parCoLO.method = [ ]
infoCoLO.SDPsolver	The information from the SDP solver used
infoCoLO.CPUdomain	CPU time in seconds for the d-space conversion
infoCoLO.CPUrange	CPU time in seconds for the r-space conversion
infoCoLO.CPUEQorLMI	CPU time in seconds for the conversion into an equality form or an LMI form
infoCoLO.CPUtotal	The total CPU time in seconds
cliqueDomain	Clique information for the d-space conversion used; if this set is nonempty, only the components of sdp variables in $\boldsymbol{x}$ with indices contained in some cliques are computed, others are set to zero. See the last paragraph of Section 3.
cliqueRange	Clique information for the r-space conversion used; the same is applied with replacing $\boldsymbol{x}$ by $\boldsymbol{y}$ as in cliqueDomain
LOP	LOP.A, LOP.b, LOP.c, LOP.K, LOP.J to described the final conic form of linear optimization problem

Table 2: Output from sparseCoLO.m

With SparseCoLO, users can experiment any combination of the four conversion methods proposed in [5] by setting the appropriate parameters, depending on the problem. This increases possibility to solve large-sized problems that are otherwise hard to solve. In addition, SparseCoLO is versatile in that it can be implemented with any of SDP solvers to reduce the size of the problems.

Numerical results in Appendix and [5] demonstrate the computational advantage of using SparseCoLO for solving the problems.

Acknowledgments
The authors would like to thank Dr. Hayato Waki for his Matlab programs on the early version of the conversion methods.

References

[1] P. Biswas and Y. Ye (2004) "Semidefinite programming for ad hoc wireless sensor network localization," in *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, ACM, New York, pp. 46–54.

[2] B. Borcher (1999), "CSDP, a C library for semidefinite programming", *Optimization Methods and Software*, **11** 613-623.

[3] M. Fukuda, M. Kojima, K. Murota and K. Nakata (2001) "Exploiting sparsity in semidefinite programming via matrix completion I: General framework," *SIAM Journal on Optimization*, **11** 647-674.

[4] K. Fujisawa, M. Fukuda, K. Kobayashi, M. Kojima, K. Nakata, M. Nakata and M. Yamashita (2008) SDPA (SemiDefinite Programming Algorithm) user's manual — Version 7.05, Research Report B-448, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552, Japan.

[5] S. Kim, M. Kojima, M. Mevissen and M. Yamashita (2009) "Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion," Research Report B-452, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552, Japan, January 2009.

[6] S. Kim, M. Kojima and H. Waki (2009) "Exploiting sparsity in SDP relaxation for sensor network localization", *SIAM Journal on Optimization*, **20** 192-215.

[7] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima and K. Murota (2003) "Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results," *Mathematical Programming*, **95** 303-327.

[8] SDPLIB 1.2 (1998) http://infohost.nmt.edu/∼sdplib/

[9] J. F. Sturm (1999) "SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, **11 & 12** 625-653.

[10] K. Toh, M. J. Todd, R. H. Tütüntü (1998) SDPT3 — a MATLAB software package for semidefinite programming, Dept. of Mathematics, National University of Singapore, Singapore.

Appendix: Numerical results

We solve a standard SDP relaxation of the max-cut problem, a standard SDP relaxation of the norm minimization problem, the sensor network localization problem [1] and some SDPs from sdplib [8] to demonstrate and compare the effectiveness of the conversion methods. Numerical experiments were performed on 2×2 2.8 GHz Quad Core Intel Xeon with 4GB memory.

In the description of numerical results, the three numbers in the column parCoLO denote parCoLO.domain, praCoLO.range and parCoLO.EQorLMI, respectively. "cpuC" and "cpuS" mean CPU time for conversion and executing SeDuMi, respectively, and "sizeA," "nnzA," "nnzS," and "nnzL" denote the size of A, the number of nonzero elements in A, the number of nonzero elements in the Schur complement matrix, and the number of nonzero elements of the Cholesky factor of the Schur complement matrix. And, "noBl" indicates the number of SDP blocks and "maxBl" the maximum size of SDP blocks.

	parCoLO	cpuC	cpuS	sizeA	nnzA	nnzS	nnzL	noBl	maxBl
Solver	maxCutSDP500d4; a sparse graph, 500 nodes , deg. parameter 4 on [0,1] × [0,1]								
sedumi	0 0 1	0.0	60.4	500 x 250000	500	250000	125250	1	500
sedumi	1 0 1	0.8	6.4	2337 x 22329	7206	708245	420848	38	54
sedumi	2 0 2	0.4	8.4	4397 x 25605	25605	624929	319955	369	23
sdpa	0 0 1	0.0	2.7	500 x 250000	500	250000	125250	1	500
sdpa	1 0 1	0.8	2.0	2337 x 22329	7206	708245	420848	38	54
sdpa	2 0 2	0.4	2.9	4397 x 25605	25605	624929	319955	369	23
sdpt3	0 0 1	0.0	5.0	500 x 250000	500	250000	125250	1	500
sdpt3	1 0 1	0.8	13.0	2337 x 22329	7206	708245	420848	38	54
sdpt3	2 0 2	0.4	42.1	4397 x 25605	25605	624929	319955	369	23
Solver	maxCutSDP1000d4; a sparse graph, 1000 nodes , deg. parameter 4 on [0,1] × [0,1]								
sedumi	0 0 1	0.0	465.9	1000 x1000000	1000	1000000	500500	1	1000
sedumi	1 0 1	2.2	29.3	5910 x 67157	19346	3829900	2852489	63	81
sedumi	2 0 2	1.3	43.8	10506 x 63489	63489	2888172	1520601	724	36
sdpa	0 0 1	0.0	19.7	1000 x1000000	1000	1000000	500500	1	1000
sdpa	1 0 1	2.2	10.0	5910 x 67157	19346	3829900	2852489	63	81
sdpa	2 0 2	1.3	15.4	10506 x 63489	63489	2888172	1520601	724	36
sdpt3	0 0 1	0.0	24.1	1000 x1000000	1000	1000000	500500	1	1000
sdpt3	1 0 1	2.2	82.4	5910 x 67157	19346	3829900	2852489	63	81
sdpt3	2 0 2	1.3	220.2	10506 x 63489	63489	2888172	1520601	724	36
Solver	maxG11 from sdplib								
sedumi	0 0 1	0.0	212.5	800 x 640000	800	640000	320400	1	800
sedumi	1 0 1	2.0	15.2	2432 x 76672	6944	1076876	650630	13	80
sedumi	2 0 2	1.0	24.1	8333 x 48698	48698	2164463	1086398	598	24
sdpa	0 0 1	0.0	12.2	800 x 640000	800	640000	320400	1	800
sdpa	1 0 1	2.0	4.6	2432 x 76672	6944	1076876	650630	13	80
sdpa	2 0 2	0.9	18.4	8333 x 48698	48698	2164463	1086398	598	24
sdpt3	0 0 1	0.0	14.8	800 x 640000	800	640000	320400	1	800
sdpt3	1 0 1	1.9	27.6	2432 x 76672	6944	1076876	650630	13	80
sdpt3	2 0 2	1.0	267.5	8333 x 48698	48698	2164463	1086398	598	24
Solver	maxG32 from sdplib								
sedumi	0 0 1	0.0	5832.6	2000 x4000000	2000	4000000	2001000	1	2000
sedumi	1 0 1	8.2	828.4	13600 x 435826	47120	33757554	26406429	21	210
sdpa	0 0 1	0.0	113.1	2000 x4000000	2000	4000000	2001000	1	2000
sdpa	1 0 1	8.3	120.0	13600 x 435826	47120	33757554	26406429	21	210
sdpt3	0 0 1	0.0	93.6	2000 x4000000	2000	4000000	2001000	1	2000
sdpt3	1 0 1	Out of mem.		13600 x 435826	47120	33757554	26406429	21	210

Table 3: Numerical results on the max-cut problem

	parCoLO	cpuC	cpuS	sizeA	nnzA	nnzS	nnzL	noBl	maxBl
Solver	normMinSDP2_500_10_10								
sedumi	0 0 2	0.0	67.5	11 x 260100	100510	121	66	1	510
sedumi	0 1 2	3.3	6.4	1111 x 24012	104510	199771	100441	21	34
sedumi	0 2 1	0.8	10.1	5555 x 60511	111010	25868025	14184290	500	11
sdpa	0 0 2	0.0	7.4	11 x 260100	100510	121	66	1	510
sdpa	0 1 2	3.4	1.0	1111 x 24012	104510	199771	100441	21	34
sdpa	0 2 1	0.8	150.0	5555 x 60511	111010	25868025	14184290	500	11
sdpt3	0 0 2	0.0	11.2	11 x 260100	100510	121	66	1	510
sdpt3	0 1 2	3.4	5.5	1111 x 24012	104510	199771	100441	21	34
sdpt3	0 2 1	0.8	30.3	5555 x 60511	111010	25868025	14184290	500	11
Solver	normMinSDP2_1000_10_10								
sedumi	0 0 2	0.0	488.0	11 x1020100	201010	121	66	1	1010
sedumi	0 1 2	21.0	13.6	2321 x 48116	209410	426041	214181	43	34
sdpa	0 0 2	0.0	35.6	11 x1020100	201010	121	66	1	1010
sdpa	0 1 2	21.5	2.1	2321 x 48116	209410	426041	214181	43	34
sdpt3	0 0 2	0.0	40.1	11 x1020100	201010	121	66	1	1010
sdpt3	0 1 2	21.3	12.4	2321 x 48116	209410	426041	214181	43	34

Table 4: Numerical results on the norm minimization problem.

	parCoLO	cpuC	cpuS	sizeA	nnzA	nnzS	nnzL	noBl	maxBl
Solver	sensor2Dim1000D; 1000 sensors on $[0,1] \times [0,1]$								
sedumi	0 0 1	0.0	2992.7	11010 x1018680	36656	13505598	6758304	1	1002
sedumi	1 0 1	2.5	237.3	19103 x 64458	66540	6360321	5671117	163	47
sedumi	2 0 2	1.3	43.8	22028 x 71721	93701	1619518	822774	914	34
sdpa	0 0 1	0.0	160.0	11010 x1018680	36656	13505598	6758304	1	1002
sdpa	1 0 1	2.6	56.0	19103 x 64458	66540	6360321	5671117	163	47
sdpa	2 0 2	1.4	19.4	22028 x 71721	93701	1619518	822774	914	34
sdpt3	0 0 1	Out of mem.		11010 x1018680	36656	13505598	6758304	1	1002
sdpt3	1 0 1	Out of mem.		19103 x 64458	66540	6360321	5671117	163	47
sdpt3	2 0 2	Out of mem.		22028 x 71721	93701	1619518	822774	914	34
Solver	sensor2Dim4000D; 4000 sensors on $[0,1] \times [0,1]$								
sedumi	1 0 1	34.7	401.7	65149 x 191521	221916	14606655	11443229	627	47
sedumi	2 0 2	15.9	227.2	85894 x 238645	332542	3168928	1633309	3892	37
sdpa	1 0 1	34.8	141.4	65149 x 191521	221916	14606655	11443229	627	47
sdpa	2 0 2	15.9	48.5	85894 x 238645	332542	3168928	1633309	3892	37
Solver	sensor2Dim6000D; 6000 sensors on $[0,1] \times [0,1]$								
sedumi	1 0 1	88.5	322.3	45607 x 172075	172977	17089305	13544542	846	41
sedumi	2 0 2	31.2	136.5	33388 x 205406	252593	3366412	1711139	5894	41
sdpa	1 0 1	89.8	101.3	45607 x 172075	172977	17089305	13544542	846	41
sdpa	2 0 2	31.5	42.0	33388 x 205406	252593	3366412	1711139	5894	41

Table 5: Numerical results on the sensor network localization problem [6]

	parCoLO	cpuC	cpuS	sizeA	nnzA	nnzS	nnzL	noBl	maxBl
Solver	arch8 from SDPLIB								
sedumi	0 0 1	0.0	8.5	174 x 26095	5634	30276	15225	1	161
sedumi	1 0 1	0.6	4.8	606 x 13044	7240	205098	102852	6	75
sdpa	0 0 1	0.0	1.4	174 x 26095	5634	30276	15225	1	161
sdpa	1 0 1	0.2	1.3	606 x 13044	7240	205098	102852	6	75
sdpt3	0 0 1	0.0	4.9	174 x 26095	5634	30276	15225	1	161
sdpt3	1 0 1	0.2	6.7	606 x 13044	7240	205098	102852	6	75
Solver	mcp250-1 from SDPLIB								
sedumi	0 0 1	0.0	10.1	250 x 62500	250	62500	31375	1	250
sedumi	1 0 1	0.7	7.3	2126 x 8540	7186	1112964	776622	68	27
sdpa	0 0 1	0.0	0.7	250 x 62500	250	62500	31375	1	250
sdpa	1 0 1	0.4	2.6	2126 x 8540	7186	1112964	776622	68	27
sdpt3	0 0 1	0.0	2.2	250 x 62500	250	62500	31375	1	250
sdpt3	1 0 1	0.3	8.8	2126 x 8540	7186	1112964	776622	68	27
Solver	mcp500-1 from SDPLIB								
sedumi	0 0 1	0.0	66.9	500 x 250000	500	250000	125250	1	500
sedumi	1 0 1	1.4	69.2	7222 x 22309	25982	10923144	9486287	152	44
sdpa	0 0 1	0.0	2.7	500 x 250000	500	250000	125250	1	500
sdpa	1 0 1	1.0	28.7	7222 x 22309	25982	10923144	9486287	152	44
sdpt3	0 0 1	0.0	5.5	500 x 250000	500	250000	125250	1	500
sdpt3	1 0 1	1.0	93.0	7222 x 22309	25982	10923144	9486287	152	44
Solver	qpG11 from SDPLIB								
sedumi	0 0 1	0.0	2755.8	800 x2560000	1600	640000	320400	1	1600
sedumi	1 0 1	10.1	12.4	2432 x 77472	7744	1076886	650635	813	80
sdpa	0 0 1	0.0	45.6	800 x2560000	1600	640000	320400	1	1600
sdpa	1 0 1	10.0	3.2	2432 x 77472	7744	1076886	650635	813	80
sdpt3	0 0 1	0.0	14.6	800 x2560000	1600	640000	320400	1	1600
sdpt3	1 0 1	10.1	32.4	2432 x 77472	7744	1076886	650635	813	80
Solver	thetaG11from SDPLIB								
sedumi	0 0 1	0.0	300.7	2401 x 641601	15201	5764801	2883601	1	801
sedumi	1 0 1	3.2	21.6	4237 x 78669	22137	2596851	1487816	13	81
sdpa	0 0 1	0.0	17.8	2401 x 641601	15201	5764801	2883601	1	801
sdpa	1 0 1	3.2	9.0	4237 x 78669	22137	2596851	1487816	13	81
sdpt3	0 0 1	0.0	39.7	2401 x 641601	15201	5764801	2883601	1	801
sdpt3	1 0 1	3.4	60.3	4237 x 78669	22137	2596851	1487816	13	81
Solver	truss8 from SDPLIB								
sedumi	0 0 1	0.0	2.5	496 x 11914	15927	246016	123256	34	19
sedumi	1 0 1	0.1	2.5	496 x 11914	15927	246016	123256	34	19
sdpa	0 0 1	0.0	2.7	496 x 11914	15927	246016	123256	34	19
sdpa	1 0 1	0.2	2.7	496 x 11914	15927	246016	123256	34	19
sdpt3	0 0 1	0.0	3.3	496 x 11914	15927	246016	123256	34	19
sdpt3	1 0 1	0.1	3.2	496 x 11914	15927	246016	123256	34	19

Table 6: Numerical results on some SDP problems from SDPLIB