PREPROCESSING SPARSE SEMIDEFINITE PROGRAMS VIA MATRIX COMPLETION

By

Katsuki Fujisawa

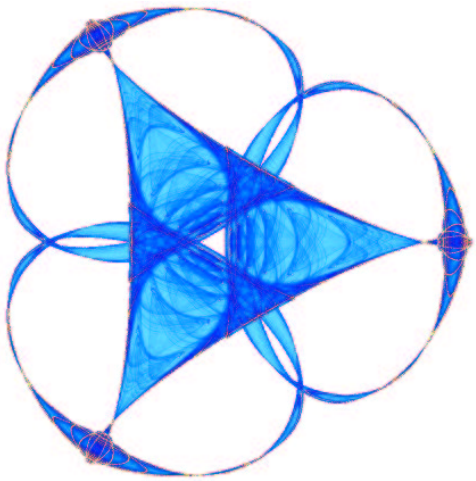
Mituhiro Fukuda

and

Kazuhide Nakata

IMA Preprint Series # 1969

(March 2004)



INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS

UNIVERSITY OF MINNESOTA
514 Vincent Hall
206 Church Street S.E.
Minneapolis, Minnesota 55455–0436

Phone: 612/624-6066 Fax: 612/626-7370
URL: http://www.ima.umn.edu

Preprocessing sparse semidefinite programs via matrix completion*

Katsuki Fujisawa (fujisawa@r.dendai.ac.jp)
Department of Mathematical Sciences, Tokyo Denki University,
Ishizaka, Hatoyama, Saitama, 350-0394, Japan

Mituhiro Fukuda (mituhiro@cims.nyu.edu)
Department of Mathematics,
Courant Institute of Mathematical Sciences, New York University,
251 Mercer Street, New York, NY, 10012-1185

Kazuhide Nakata (knakata@me.titech.ac.jp)
The Department of Industrial Engineering and Management, Tokyo Institute of Technology,
2-12-1 Oh-okayama, Meguro, Tokyo, 152-8552, Japan

March, 2004

Abstract

Considering that preprocessing is an important phase in linear programming, it should be systematically more incorporated in semidefinite programming solvers. The conversion method proposed by the authors (SIAM Journal on Optimization, vol. 11, pp. 647–674, 2000, and Mathematical Programming, Series B, vol. 95, pp. 303–327, 2003) is a preprocessing of sparse semidefinite programs based on matrix completion. This article proposed a new version of the conversion method which employs a flop estimation function inside its heuristic procedure. Extensive numerical experiments are included showing the advantage of preprocessing by the conversion method for very sparse semidefinite programs of certain classes.

Keywords: semidefinite programming, preprocessing, sparsity, matrix completion, clique tree, numerical experiments

*This manuscript was also issued as Research Report B-401, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-okayama Meguro, Tokyo, 152-8552, Japan, March 2004.

1 Introduction

Recently, Semidefinite Programming (SDP) has gained attention in several new fronts like in global optimization of problems involving polynomials [13, 14, 18] and in quantum chemistry [27] besides the well-known applications in system and control theory, in relaxation of combinatorial optimization problems, *etc.*

These new classes of SDPs characterize as being of large-scale and most of the time as challenging as even to load the problem data onto the physical memory of the computer. As a practical compromise, we often restrict ourselves to solve sparse instances of these large-scale SDPs.

Keeping in mind the above necessity to solve challenging SDPs, this article pushes further the preprocessing procedure named *conversion method* proposed in [8, 16]. The conversion method explores the structural sparsity of SDP data matrices, converting a given SDP into an equivalent SDP based on the matrix completion theory. If the SDP data matrices are very sparse, which means that matrix sizes are large at least, the conversion method produces an SDP which can be solved faster and requires less memory than the original SDP when solved by the primal-dual interior-point method [16].

The conversion method is an initial step to consider seriously the preprocessing phase of sparse SDPs as it is common in linear programming [1].

In this sense, we already proposed a general linear transformation which can enhance the sparsity of an SDP [8, Section 6]. Gatermann and Parrilo address another algebraic transformation that can be interpreted as a preprocessing of SDPs under special conditions which can transform the problems into block diagonal SDPs [9]. Also, Toh recognizes the importance of analyzing the data matrices to remove redundant constraints which can cause degeneracy [22]. All of above procedures can be used for sparse and even for dense SDPs.

We strong believe that further investigations are necessary to propose efficient preprocessing to solve large-scale, (sparse) and general SDPs.

The main idea of the conversion method is as follows.

Let $\mathcal{S}^n$ denote the space of $n \times n$ symmetric matrices with the Frobenius inner product $\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^n \sum_{j=1}^n X_{ij} Y_{ij}$ for $\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{S}^n$, and $\mathcal{S}^n_+$ the subspace of $n \times n$ symmetric positive semidefinite matrices. Given $\boldsymbol{A}_p \in \mathcal{S}^n$ $(p = 0, 1, \ldots, m)$ and $\boldsymbol{b} \in \mathbb{R}^m$, we define the *standard equality form SDP* by

$$\left\{ \begin{array}{ll} \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{A}_p \bullet \boldsymbol{X} = b_p \ (p = 1, 2, \ldots, m), \\ & \boldsymbol{X} \in \mathcal{S}^n_+, \end{array} \right. \tag{1}$$

and its dual by

$$\left\{ \begin{array}{ll} \text{maximize} & \displaystyle\sum_{p=1}^m b_p y_p \\ \text{subject to} & \displaystyle\sum_{p=1}^m \boldsymbol{A}_p y_p + \boldsymbol{S} = \boldsymbol{A}_0, \\ & \boldsymbol{S} \in \mathcal{S}^n_+. \end{array} \right. \tag{2}$$

In this article, we are mostly interested in solving sparse SDPs where the data matrices

$\boldsymbol{A}_p$ $(p = 0, 1, \ldots, m)$ are sparse, and the dual matrix variable $\boldsymbol{S} = \boldsymbol{A}_0 - \sum_{p=1}^{m} \boldsymbol{A}_p y_p$ inherits the sparsity of $\boldsymbol{A}_p$'s.

The sparse structure of an SDP can be represented by the *aggregate sparsity pattern* of data matrices (alternatively called *aggregate density pattern* in [5]):

$$E = \{(i,j) \in V \times V : \ [\boldsymbol{A}_p]_{ij} \neq 0 \text{ for some } p \in \{0, 1, \ldots, m\}\}.$$

Here V denotes the set $\{1, 2, \ldots, n\}$ of row/column indices of data matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$, and $[\boldsymbol{A}_p]_{ij}$ denotes the (i,j)th element of $\boldsymbol{A}_p \in \mathcal{S}^n$. It is also convenient to identify the aggregate sparsity pattern E with the *aggregate sparsity pattern matrix* $\boldsymbol{A}(E)$ having unspecified nonzero numerical values in E and zero otherwise.

In accordance with the ideas and definitions presented in [8, 16], consider a collection of nonempty subsets $C_1, C_2, \ldots, C_\ell$ of V satisfying

(i) $E \subseteq F \equiv \bigcup_{r=1}^{\ell} C_r \times C_r$;

(ii) Any partial symmetric matrix $\bar{\boldsymbol{X}}$ with specified elements $\bar{X}_{ij} \in \mathbb{R}$ $((i,j) \in F)$ has a *positive semidefinite matrix completion* (*i.e.,* given any $\bar{X}_{ij} \in \mathbb{R}$ $((i,j) \in F)$, there exists a positive semidefinite $\boldsymbol{X} \in \mathcal{S}^n$ such that $X_{ij} = \bar{X}_{ij} \in \mathbb{R}$ $((i,j) \in F)$) if and only if the submatrices $\bar{\boldsymbol{X}}_{C_r C_r} \in \mathcal{S}_+^{C_r}$ $(r = 1, 2, \ldots, \ell)$.

Here $\bar{\boldsymbol{X}}_{C_r C_r}$ denotes the submatrix of $\bar{\boldsymbol{X}}$ obtained by deleting all rows/columns $i \notin C_r$, $\mathcal{S}_+^{C_r}$ denotes the set of positive semidefinite symmetric matrices with elements specified in $C_r \times C_r$. We can assume without loss of generality $C_1, C_2, \ldots, C_\ell$ to be maximal sets with respect to set inclusion.

Then, an equivalent formulation of the SDP (1) can be written as follows [16]:

$$\left\{ \begin{array}{lll} \text{minimize} & \displaystyle\sum_{(i,j)\in F} [\boldsymbol{A}_0]_{ij} X_{ij}^{\hat{r}(i,j)} & \\ \text{subject to} & \displaystyle\sum_{(i,j)\in F} [\boldsymbol{A}_p]_{ij} X_{ij}^{\hat{r}(i,j)} = b_p & (p = 1, 2, \ldots, m), \\ & X_{ij}^r = X_{ij}^s & \left(\begin{array}{l} (i,j) \in (C_r \cap C_s) \times (C_r \cap C_s), \quad i \geq j, \\ (C_r, C_s) \in \mathcal{E}, \quad 1 \leq r < s \leq \ell \end{array} \right), \\ & \boldsymbol{X}^r \in \mathcal{S}_+^{C_r} & (r = 1, 2, \ldots, \ell), \end{array} \right. \tag{3}$$

where $\mathcal{E}$ of the clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ is defined in Section 2, and $\hat{r}(i,j) = \min\{r : \ (i,j) \in C_r \times C_r\}$ is introduced to avoid the addition of repeated terms. If we further introduce a block-diagonal symmetric matrix variable of the form

$$\boldsymbol{X}' = \begin{pmatrix} \boldsymbol{X}^1 & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{X}^2 & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{X}^\ell \end{pmatrix},$$

and appropriately rearrange all data matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$, and the matrices corresponding to the equalities $X^r_{ij} = X^s_{ij}$ in (3) to have the same block-diagonal structure as $\boldsymbol{X}'$, we obtain an equivalent standard equality primal SDP.

Observe that the original standard equality primal SDP (1) has a single matrix variable of size $n \times n$ and m equality constraints. After the conversion, the SDP (3) has

(a) ℓ matrices of size $n_r \times n_r$, $n_r \leq n$ $(r = 1, 2, \ldots, \ell)$, and

(b) $m_+ = m + \displaystyle\sum_{(C_r, C_s) \in \mathcal{E}} g(C_r \cap C_s)$ equality constraints where $g(C) = \dfrac{|C|(|C|+1)}{2}$,

where $n_r \equiv |C_r|$ denotes the number of elements of C_r.

In this article, we propose a new version of the conversion method which tries to convert a sparse SDP predicting *a priori* the flops of solving it by the primal-dual interior-point method. The original conversion method [8, 16] has a simple heuristic routine based only on the matrix sizes (see Subsection 3.2) which can be deficient in the sense of ignoring the actual computation of the numerical linear algebra in SDP solvers. This work is a tentative of refining it, and a flop estimation function is introduced for this purpose. The flops to compute the Schur Complement Matrix (SCM) [6] and other routines like solving the SCM linear equation, and computing eigenvalues can be roughly estimated as a function of equality constraints m, matrix sizes n's, and data sparsity. The parameters of the newly introduced function are estimated by a simple statistical method based on ANOVA (analysis of variance). Finally, this function is used in a new heuristic routine to generate equivalent SDPs.

The new version of the conversion method is compared with the original version with slight improvement and SDPs without conversion through extensive numerical experiments using SDPA 6.00 [25] and SDPT3 3.02 [23] on selected sparse SDPs from different classes as a tentative of detecting SDPs which are suitable for the conversion method. We can conclude that preprocessing by the conversion method becomes more advantageous as the SDPs are sparse. In particular, it seems that sparse SDPs which have less than 5% on the extended sparsity pattern (see Sections 2 for its definition) can be solved very efficiently in general. Preprocessing by the conversion method is very advisable for sparse SDPs since we can obtain a speed up of 10$\sim$100 times in some cases, and even in the eventual cases when solving the original problem is faster, preprocessed SDPs take at most two times more to solve in most of the cases considered here.

Some other related work that also explores sparsity and matrix completions are the completion method [8, 16], and its parallel version [17]. Also, Burer proposed a primal-dual interior-point method restricted on the space of partial positive definite matrices [5].

The rest of the article is written as follows. Section 2 revised some graph related theory which has a strong connection with matrix completion. Section 3 presents the general framework of the conversion method in a neat way, revises the original version in detail, and proposes a minor modification. Section 4 describes the newly proposed conversion method which estimates the flops of each iteration of primal-dual interior-point method solvers. Finally, Section 5 presents an extensive numerical results comparing the performance of the two conversion methods with SDPs without preprocessing.

2 Preliminaries

The details of this section can be found in [3, 8, 16] and references therein. Let $G(V, E^\circ)$ denote a graph where $V = \{1, 2, \ldots, n\}$ is the vertex set, and E° is the edge set defined as $E^\circ = E \backslash \{(i, i) : i \in V\}$, $E \subseteq V \times V$. A graph $G(V, F^\circ)$ is called *chordal*, *triangulated* or *rigid circuit* if every cycle of length ≥ 4 has a chord (an edge connecting two non-consecutive vertices of the cycle).

There is a close connection between chordal graphs and positive semidefinite matrix completions that has been fundamental in the conversion method [8, 16], *i.e.*, (ii) in Introduction holds if and only if the associate graph $G(V, F^\circ)$ of F given in (i) is chordal [8, 10]. We further observe that amazingly the same fact was proved independently in graphical models in statistics [15], and its known as decomposable models [24].

Henceforth, we assume that $G(V, F^\circ)$ denotes a chordal graph. We call F an *extended sparsity pattern* of E and $G(V, F^\circ)$ a *chordal extension* or *filled graph* of $G(V, E^\circ)$. Notice that obtaining a chordal extension $G(V, F^\circ)$ from $G(V, E^\circ)$ corresponds to adding new edges to $G(V, E^\circ)$ in order to $G(V, F^\circ)$ becomes a chordal graph.

Chordal graphs are well-known structures in graph theory, and can be characterized for instance as follows. A graph is chordal if and only if we can construct a *clique tree* from it. Although there are several equivalent ways to define clique trees, we employ the following one based on the clique-intersection property (CIP) which will be useful throughout the article.

Let $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ be any family of maximal subsets of $V = \{1, 2, \ldots, n\}$. Let $\mathcal{T}(\mathcal{K}, \mathcal{E})$ be a tree formed by vertices from $\mathcal{K}$ and edges from $\mathcal{E} \subseteq \mathcal{K} \times \mathcal{K}$. $\mathcal{T}(\mathcal{K}, \mathcal{E})$ is called a *clique tree* if it satisfies the *clique-intersection property* (CIP):

(CIP) For each pair of vertices $C_r, C_s \in \mathcal{K}$, the set $C_r \cap C_s$ is contained in every vertex on the (unique) path connecting C_r and C_s in the tree.

In particular, we can construct a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ from the chordal extension $G(V, F^\circ)$ if we take $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ as the family of all maximal cliques of $G(V, F^\circ)$, and define appropriately the edge set $\mathcal{E} \subseteq \mathcal{K} \times \mathcal{K}$ for $\mathcal{T}(\mathcal{K}, \mathcal{E})$ to satisfy the CIP.

Clique trees can be computed efficiently from a chordal graph. We observe further that clique trees are not uniquely determined for a given chordal graph. Though it is known that the multiset of separators $\{C_r \cap C_s : (C_r, C_s) \in \mathcal{E}\}$ is invariant for all clique trees $\mathcal{T}(\mathcal{K}, \mathcal{E})$ of a given chordal graph, a fact suitable for our purpose together with the CIP.

The following two lemmas will be very important in the development of the conversion method in the next section. Figure 1 illustrates Lemmas 2.1 and 2.2.

Lemma 2.1 *[16] Let $\mathcal{T}(\mathcal{K}, \mathcal{E})$ be a clique tree of $G(V, F^\circ)$, and suppose that $(C_r, C_s) \in \mathcal{E}$. We construct a new tree $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ merging C_r and C_s, i.e., replacing $C_r, C_s \in \mathcal{K}$ by $C_r \cup C_s \in \mathcal{K}'$, deleting $(C_r, C_s) \in \mathcal{E}$, and replacing $(C_r, C), (C_s, C) \in \mathcal{E}$ by $(C_r \cup C_s, C) \in \mathcal{E}'$. Then $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ is a clique tree of $G(V, F'^\circ)$, where $F' = \{(i, j) \in C'_r \times C'_r : r = 1, 2, \ldots, \ell'\}$ for $\mathcal{K}' = \{C'_1, C'_2, \ldots, C'_{\ell'}\}$, $\ell' = \ell - 1$. Moreover, let m_+ be defined as in (b) (in Introduction), and m'_+ be the corresponding one for $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$. Then $m'_+ = m_+ - g(C_r \cap C_s)$.*

Lemma 2.2 *[16] Let $\mathcal{T}(\mathcal{K}, \mathcal{E})$ be a clique tree of $G(V, F^\circ)$, and suppose that $(C_r, C_q), (C_s, C_q) \in \mathcal{E}$. We construct a new tree $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ in the following way:*

(i) If $C_r \cup C_s \not\supseteq C_q$, merge C_r and C_s, i.e., replace $C_r, C_s \in \mathcal{K}$ by $C_r \cup C_s \in \mathcal{K}'$ and replace $(C_r, C), (C_s, C) \in \mathcal{E}$ by $(C_r \cup C_s, C) \in \mathcal{E}'$;

(ii) Otherwise, merge C_r, C_s and C_q, i.e., replace $C_r, C_s, C_q \in \mathcal{K}$ by $C_r \cup C_s \cup C_q \in \mathcal{K}'$, delete $(C_r, C_q), (C_s, C_q) \in \mathcal{E}$ and replace $(C_r, C), (C_s, C), (C_q, C) \in \mathcal{E}$ by $(C_r \cup C_s \cup C_q, C) \in \mathcal{E}'$.

Then $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ is a clique tree of $G(V, F'^\circ)$, where $F' = \{(i, j) \in C'_r \times C'_r : r = 1, 2, \ldots, \ell'\}$ for $\mathcal{K}' = \{C'_1, C'_2, \ldots, C'_{\ell'}\}$, $\ell' = \ell - 1$ (case i) and $\ell' = \ell - 2$ (case ii). Moreover, let m_+ be defined as in (b) (in Introduction), and m'_+ be the corresponding one for $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$. Then, we have respectively

(i) $m'_+ = m_+ - g(C_r \cap C_q) - g(C_s \cap C_q) + g((C_r \cup C_s) \cap C_q)$ and;

(ii) $m'_+ = m_+ - g(C_r \cap C_q) - g(C_s \cap C_q)$.

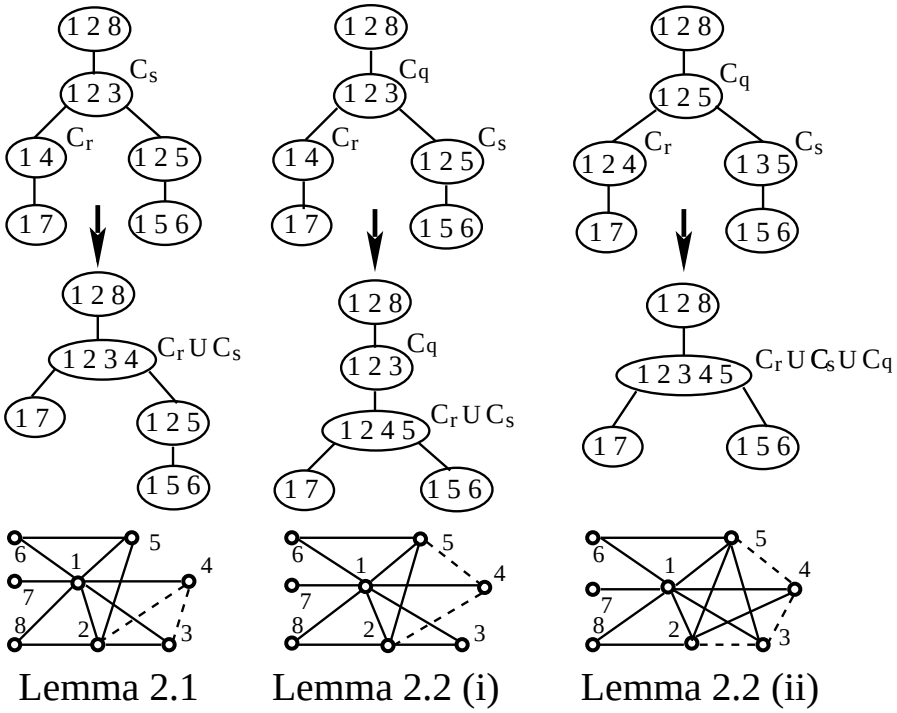


Figure 1: Illustration of Lemmas 2.1 and 2.2. Clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ before the merging (above), clique tree $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ after the merging (middle), and the associate chordal graph (bottom). Doted lines denote the edges added to the graph because of the clique merging.

3 Conversion Method

3.1 An Outline

An implementable conversion method is summarized in Algorithm 3.1. See [16] for details.

Algorithm 3.1 Input: *sparse SDP;* **Output:** *Equivalent SDP with small block matrices;*

Step 1. *Read the SDP data and determine the aggregate sparsity pattern E.*

Step 2. *Find an ordering of rows/columns $V = \{1, 2, \ldots, n\}$ which possibly provides lesser fill-in in the aggregate sparsity matrix $\boldsymbol{A}(E)$ (e.g., Spooles 2.2 [2] and METIS 4.0.1 [11]).*

Step 3. *From the ordering above, perform a symbolic Cholesky factorization onto $\boldsymbol{A}(E)$ associated with $G(V, E^\circ)$ to determine a chordal extension $G(V, F^\circ)$.*

Step 4. *Compute a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ from $G(V, F^\circ)$.*

Step 5. *Use some heuristic procedure to diminish the overlaps $C_r \cap C_s$ between adjacent cliques, i.e., $(C_r, C_s) \in \mathcal{E}$ such that $C_r, C_s \in \mathcal{K}$, and determine a new clique tree $\mathcal{T}^*(\mathcal{K}^*, \mathcal{E}^*)$.*

Step 6. *Convert the original SDP (1) into (3) using information on $\mathcal{T}^*(\mathcal{K}^*, \mathcal{E}^*)$.*

One of the most important issues of the conversion method is to obtain a suitable chordal extension $G(V, F^\circ)$ of $G(V, E^\circ)$ which allow us to apply a positive semidefinite matrix completion to the original sparse SDP (1).

We also known that a chordal extension $G(V, F^\circ)$ of $G(V, E^\circ)$ can be obtained easily if we perform a symbolic Cholesky factorization on the aggregate sparsity pattern matrix $\boldsymbol{A}(E)$ according to any reordering of $V = \{1, 2, \ldots, n\}$. Unfortunately, the problem of finding such an ordering which minimizes the fill-in in $\boldsymbol{A}(E)$ is $\mathcal{NP}$-complete. Therefore, we rely on some heuristic packages to determine an ordering which possible give less fill-in in Step 2.

Once we have a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ at Step 4, we can obtain an completely equivalent SDP to the original one with smaller block matrices but with larger number of equality constraints after Step 6. This step consists in visiting once each of the cliques in the clique tree $\mathcal{T}^*(\mathcal{K}^*, \mathcal{E}^*)$ in order to determine the overlapping elements of $C_r \cap C_s$ ($(C_r, C_s) \in \mathcal{E}^*$, $C_r, C_s \in \mathcal{K}^*$). However, as mentioned in Introduction, we obtain at last an SDP with

(a') $\ell^* = |\mathcal{K}^*|$ matrices of size $n_r \times n_r$, $|C_r| \equiv n_r \leq n$ $(r = 1, 2, \ldots, \ell^*)$, and

(b') $m_+ = m + \displaystyle\sum_{(C_r, C_s) \in \mathcal{E}^*} g(C_r \cap C_s)$ equality constraints.

If we opt for an chordal extension $G(V, F^\circ)$ that gives less fill-in as possible at Step 3 (and therefore an ordering at Step 2), we obtain an SDP (3) with ℓ^* smallest block matrices as possible of size n_r $(r = 1, 2, \ldots, \ell^*)$, and more crucial, a large number of equality constraints $m_+ \gg m$. One of the keys to obtain a good conversion is to balance the factors (a') and (b') above to have few flop operations as possible when solving by an SDP solver. Therefore, there is a necessity to use a heuristic procedure at Step 5 which manipulates directly the clique trees, which in practice means that we are adding new edges to the chordal graph

$G(V, F^\circ)$ to create a new chordal graph $G(V, (F^*)^\circ)$ with $F^* \supseteq F$ and a corresponding clique tree $\mathcal{T}^*(\mathcal{K}^*, \mathcal{E}^*)$, which has less overlaps between adjacent cliques.

One can also consider an algorithm that avoids all of these manipulations and finds an ordering at Step 2 which gives the best chordal extension (and a clique tree), and therefore, makes Step 5 unnecessary. However, this alternative seems too beyond of our perspectives due to the complexity of the problem: predicting flops of a sophisticated optimization solver from the structure of the feeding data, and producing the best ordering of rows/columns of the aggregate sparsity matrix $\boldsymbol{A}(E)$.

Therefore, we assume Algorithm 3.1 as a pragmatic strategy for the conversion method.

The details of Step 5 are given in the next subsection.

3.2 A Simple Heuristic Algorithm to Balance the Sizes of SDPs

We will make use of Lemmas 2.1 and 2.2 here. These lemmas tell us a sufficient condition for merging the cliques in the clique tree without losing the CIP. Once we merge two cliques C_r and C_s, it will reduce the total number of block matrices by one, the number of equality constraints by $g(C_r \cap C_s)$, and increase the size of one of block matrices in (3) in the simplest case (Lemma 2.1). Also, observe that these operations add extra edges to the chordal graph $G(V, F^\circ)$ to produce a chordal graph $G(V, (F^*)^\circ)$ which is associated with the clique tree $\mathcal{T}^*(\mathcal{K}^*, \mathcal{E}^*)$.

As we have mentioned before, it seems very difficult to find an exact criterion which determines when two maximal cliques C_r and C_s satisfying the hypothesis of Lemmas 2.1 or 2.2 should be merged or not to balance the factors (a') and (b') in terms of flops. Therefore, we have adopted one simple criterion [16].

Let $\zeta \in (0, 1)$. We decide to merge the cliques C_r and C_s in $\mathcal{T}(\mathcal{K}, \mathcal{E})$ if

$$h(C_r, C_s) \equiv \min \left\{ \frac{|C_r \cap C_s|}{|C_r|}, \frac{|C_r \cap C_s|}{|C_s|} \right\} \geq \zeta. \tag{4}$$

Although criterion (4) is not complete, it takes into account the sizes of the cliques C_r and C_s involved, and compares them with the size of common indices $|C_r \cap C_s|$. Also, the minimization among the two quantities avoids the merging of a large and a small cliques which share a reasonable number of indices if compared with the smaller one. In particular, this criterion ignores the smallest part between $|C_r|$ and $|C_s|$.

Again, we opt for a specific order to merge the cliques in the clique tree as given in Algorithm 3.2 [16]. Variations are possible but is seems too demanding for our final purpose.

Here we introduce a new parameter η, which was not considered in the previous version [16], since we think that the number of equality constraints in (3) must diminish considerably when merging cliques to reduce the overall computational time after the conversion.

Algorithm 3.2 Diminishing the number of maximal cliques in the clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$.

Choose a maximal clique in $\mathcal{K}$ *to be the root for* $\mathcal{T}(\mathcal{K}, \mathcal{E})$*, and let* $\zeta, \eta \in (0, 1)$
for *each maximal clique* C *which was visited for the last time in* $\mathcal{T}(\mathcal{K}, \mathcal{E})$ *in a depth-first search*
  Set $C_q = C$
  for *each pair of descendents* C_r *and* C_s *of* C_q *in* $\mathcal{T}(\mathcal{K}, \mathcal{E})$
    if *criterion (4) is satisfied (and* $m'_+ < \eta m_+$ *if Lemma 2.2 (i) applies)*
      then *merge* C_r *and* C_s *(or* C_q*,* C_r *and* C_s*), and let* $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$
  end(for)
  Set $C_r = C$
  for *each descendent* C_s *of* C_r *in* $\mathcal{T}(\mathcal{K}, \mathcal{E})$
    if *criterion (4) is satisfied*
      then *merge* C_r *and* C_s *and let* $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$
  end(for)
end(for)

Unfortunately in practice, each of the above parameters depends on the SDP. In Subsection 4.2 we will try to estimate the "best parameter" using a statistical method. The inclusion of the new parameter η, and the estimation of good values for both parameters ζ and η in (4) makes Algorithm 3.2 an improvement over the previous one [16].

4 Conversion Method with Flop Estimation

So far, we have presented the algorithms implemented in [16] with a slight modification. Our proposal here is to replace criterion (4) with a new one which approximately predicts the flops of an SDP solver.

4.1 Estimating Flops of SDP Codes

At first, we will restrict our discussion on a particular solver: SDPA 6.00 [25] which is an implementation of the Mehrotra-type primal-dual predictor-corrector interior-point method using the HRVW/KSH/M direction.

The actual flops of sophisticated solvers are very complex and difficult to predict. They depend not only on the sizes and the structure of a particular SDP, but also on the actual data, sparsity, degeneracy of the problem, *etc.* However, we know a rough estimation of flops per iteration of a primal-dual interior-point method. It consists on the flops to compute the SCM [6], plus other parts which depend on the actual size and eventually on the number of nonzero elements of data matrices of a given SDP which is $\mathcal{O}(m^3 + \sum_{r=1}^{\ell} n_r^3 + m \sum_{r=1}^{\ell} n_r^2)$ [6].

In particular, SDPA 6.00 considers the sparsity of data matrices $\boldsymbol{A}_p$ $(p = 1, 2, \ldots, m)$, and employs the formula $\mathcal{F}_*$ [7] to compute the SCM. For each $p = 1, 2, \ldots, m$ and $r = 1, 2, \ldots, \ell$, let $f_p(r)$ denote the number of nonzero elements of $\boldsymbol{A}_p$ for the corresponding block matrix with dimension $n_r \times n_r$. Analogously, $f_\Sigma(r)$ denotes the number of nonzero elements of $\boldsymbol{A}(E)$ for the corresponding block matrix. In the following discussion about flop estimate,

there will be no loss of generality in considering that $f_p(r)$ $(p = 1, 2, \ldots, m)$ are sorted in non-increasing order for each $r = 1, 2, \ldots, \ell$ fixed.

Given a constant $\kappa \geq 1$, the cost to compute the SCM is given by

$$\sum_{r=1}^{\ell} S(f_1(r), f_2(r), \ldots, f_m(r), n_r)$$

where

$$\begin{aligned} S(f_1(r), f_2(r), \ldots, f_m(r), n_r) = \sum_{p=1}^{m} \min\{&\kappa n_r f_p(r) + n_r^3 + \kappa \sum_{q=p}^{m} f_q(r), \\ &\kappa n_r f_p(r) + \kappa(n_r + 1) \sum_{q=p}^{m} f_q(r), \\ &\kappa(2\kappa f_p(r) + 1) \sum_{q=p}^{m} f_q(r)\}. \end{aligned} \tag{5}$$

Considering also the sparsity of block matrices, we propose the following formula for flop estimate of each iteration of the primal-dual interior-point method. Let $\alpha, \beta, \gamma > 0$,

$$\begin{aligned} &C_{\alpha,\beta,\gamma}(f_1, f_2, \ldots, f_m, m, n_1, n_2, \ldots, n_\ell) \\ &= \sum_{r=1}^{\ell} S(f_1(r), f_2(r), \ldots, f_m(r), n_r) + \alpha m^3 + \beta \sum_{r=1}^{\ell} n_r^3 + \gamma \sum_{r=1}^{\ell} n_r f_\Sigma(r). \end{aligned} \tag{6}$$

Observe that the term $\mathcal{O}(m \sum_{r=1}^{\ell} n_r^2)$ was not include in the proposed formula for the reasons explained next.

Our goal is to replace criterion (4) which defines when we should merge the cliques C_r and C_s in $\mathcal{T}(\mathcal{K}, \mathcal{E})$. Therefore we just need to consider the difference of (6) before and after merging C_r and C_s to determine if it is advantageous or not to execute this operation (see Step 5 of Algorithm 3.1). Considering the most complicate case Lemma 2.2 (ii), we decide to *merge* if

$$\begin{aligned} &C^{before}_{\alpha,\beta,\gamma}(f_1, f_2, \ldots, f_m, m, n_1, n_2, \ldots, n_r, n_s, n_q, \ldots, n_\ell) \\ &-C^{after}_{\alpha,\beta,\gamma}(f_1, f_2, \ldots, f_{m_+}, m_+, n_1, n_2, \ldots, n_t, \ldots, n_{\ell'}) \\ &= S(f_1(r), f_2(r), \ldots, f_m(r), n_r) + S(f_1(s), f_2(s), \ldots, f_m(s), n_s) + S(f_1(q), f_2(q), \ldots, f_m(q), n_q) \\ &-S(f_1(t), f_2(t), \ldots, f_{m_+}(t), n_t) + \alpha(m^3 - m_+^3) + \beta(n_r^3 + n_s^3 + n_q^3 - n_t^3) \\ &+\gamma(n_r f_\Sigma(r) + n_s f_\Sigma(s) + n_q f_\Sigma(q) - n_t f_\Sigma(t)) > 0, \end{aligned} \tag{7}$$

where t denotes a new index of a block matrix (clique) after merging C_r, C_s and C_q, $n_t = |C_r \cup C_s \cup C_q| = |C_r \cup C_s|$, and $\ell' = \ell - 2$. Criterion (7) has the advantage of just carrying out the computation of corresponding block matrices (cliques). The inclusion of $\mathcal{O}(m \sum_{r=1}^{\ell} n_r^2)$ in (6) would complicate the evaluation of (7) since it would involve information on all block matrices.

Another simplification we imposed in the actual implementation was to replace $f_\Sigma(\cdot)$ by $f'_\Sigma(\cdot)$,

$$f_\Sigma(n_t) \geq f'_\Sigma(n_t) \equiv \max\{f_\Sigma(r), f_\Sigma(s), f_\Sigma(r) + f_\Sigma(s) - |C_r \cap C_s|^2\}$$

which avoids us to recalculate the non-zeros elements of each corresponding block matrix at every evaluation of (7). We observe however that $f_p(r)$ $(p = 1, 2, \ldots, m_+,\ r = 1, 2, \ldots, \ell')$ can be always retrieved exactly.

The remaining cases Lemma 2.1 and Lemma 2.2 (i) follow analogously.

Preliminary numerical experiments using criterion (7) showed that its computation is still very expensive even after several simplifications. Therefore, we opted to implement Algorithm 4.1 which is similar to Algorithm 3.2 and utilizes a hybrid criterion with (4).

Algorithm 4.1 Diminishing the number of maximal cliques in the clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$.

Choose a maximal clique in $\mathcal{K}$ to be the root for $\mathcal{T}(\mathcal{K}, \mathcal{E})$, and let $0 < \zeta_{\min} < \zeta_{\max} < 1$, and $\alpha, \beta, \gamma > 0$.
for *each maximal clique C which was visited for the last time in $\mathcal{T}(\mathcal{K}, \mathcal{E})$ in a depth-first search*
  Set $C_q = C$
  for *each pair of descendents C_r and C_s of C_q in $\mathcal{T}(\mathcal{K}, \mathcal{E})$*
    if *criterion (4) is satisfied for $\zeta_{\max}$*
      then *merge C_r and C_s (or C_q, C_r and C_s), and let $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$*
    elseif *criterion (4) is satisfied, for $\zeta_{\min}$*
      if *criterion (7) is satisfied*
        then *merge C_r and C_s (or C_q, C_r and C_s), and let $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$*
  end(for)
  Set $C_r = C$
  for *each descendent C_s of C_r in $\mathcal{T}(\mathcal{K}, \mathcal{E})$*
    if *criterion (4) is satisfied for $\zeta_{\max}$*
      then *merge C_r and C_s and let $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$*
    elseif *criterion (4) is satisfied, for $\zeta_{\min}$*
      if *criterion (7) is satisfied with the terms corresponding to n_q and C_q removed*
        then *merge C_r and C_s and let $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$*
  end(for)
end(for)

Algorithm 4.1 utilizes the new criterion (7) if $\zeta_{\min} \leq h(C_r, C_s) < \zeta_{\max}$. If $h(C_r, C_s) \geq \zeta_{\max}$, we automatically decide to merge. If $h(C_r, C_s) < \zeta_{\min}$, we do not merge. This strategy avoids excessive evaluation of (7).

As in Algorithm 3.2, it remains a critical question of how we choose the parameters; $\alpha, \beta, \gamma > 0$ in Algorithm 4.1, and $\zeta, \eta \in (0, 1)$ in Algorithm 3.2.

Although we are mainly focusing on SDPA 6.00, we suppose that the same algorithm and criterion can be adopted for SDPT3 3.02 [23] with the HRVW/KSH/M direction since it also utilizes the formula $\mathcal{F}_*$ [7] to compute the SCM, and it has the same complexity order at each iteration of the primal-dual interior-point method. Therefore, we have also included it in our numerical experiments.

4.2 Estimating Parameters for the Best Performance

Estimating parameters $\zeta, \eta \in (0,1)$ in Algorithm 3.2, and $\alpha, \beta, \gamma > 0$ in Algorithm 4.1 is not an easy task. In fact, our experience tell us that each SDP has its "best parameter". Nevertheless, we propose the following way to determine the possible best and universal parameters $\zeta, \eta, \alpha, \beta$, and γ.

We consider four classes of SDP from [16] as our benchmark problems, *i.e.*, norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of max-cut problems over lattice graphs, and SDP relaxation of graph partitioning problems (see Subsection 5.1).

Then using a technic which combines the *analysis of variance* (ANOVA) [20] and the orthogonal arrays [19] described in [21], we try to estimate the universal parameters. The ANOVA is in fact a well-known method to detect the most significant factors (parameters). However, it is possible to determine the best values for the parameters in the process to compute them. Therefore, repeating ANOVA for different set of parameter values, we can obtain the most possible best parameters for our benchmark problems. In addition, the orthogonal arrays allow us to avoid the experimentations of all possible combinations of parameter values, and even though, obtain the desired result. Details of the method are beyond the objective of this paper and therefore omitted here.

We conducted our experiments at two different computers to verify the sensibility of the parameters: computer A (Pentium III 700MHz with a level 1 data cache of 16KB, level 2 cache of 1MB, and main memory of 2GB) and computer B (Athlon 1.2GHz with a level 1 data cache of 64KB, level 2 cache of 256KB, and main memory of 2GB). We obtained the following parameters given at Table 1 for SDPA 6.00 [25] and SDPT3 3.02 [23].

Table 1: Parameters for Algorithms 3.2 and 4.1 at the computers A and B when using SDPA 6.00 and SDPT3 3.02.

	computer A					computer B				
code	ζ	η	α	β	γ	ζ	η	α	β	γ
SDPA 6.00	0.065	0.963	0.50	36	11	0.055	0.963	0.72	16	9
SDPT3 3.02	0.095	1.075	0.70	20	46	0.085	0.925	0.58	12	50

5 Numerical Experiments

We report in this section the numerical experiments on the performance of proposed versions of the conversion method, *i.e.*, the original version with a new parameter in its heuristic procedure (Subsection 3.2) and the newly proposed one which estimates the flops of each iteration of SDP solvers (Subsection 4.1).

Among the major codes to solve SDPs, we chose the SDPA 6.00 [25] and the SDPT3 3.02 [23]. Both codes are implementations of the primal-dual predictor-corrector infeasible interior-point method. In addition, they use the HRVW/KSH/M search direction and the

subroutines described in Subsection 4.1 (also [7]) to compute the SCM in which our newly proposed conversion flop estimation version partially relies.

Two different computers were used in the numerical experiments: computer A (Pentium III 700MHz with a level 1 data cache of 16KB, level 2 cache of 1MB, and main memory of 2GB) and computer B (Athlon 1.2GHz with a level 1 data cache of 64KB, level 2 cache of 256KB, and main memory of 2GB). Observe that they have different CPU chips and foremost, different cache sizes which have a relative effect on the performance of numerical linear algebra subroutines used in each of the codes.

Three different set of SDPs were tested, and they are reported in the next subsections. Subsection 5.1 reports results on the SDPs we used to estimate the parameters (see Subsection 4.2), and the same parameters were used for the SDPs in Subsections 5.2, and 5.3. The other parameters κ in (5) was fixed to 2.2, $\zeta_{\min}$ and $\zeta_{\max}$ of Algorithm 4.1 were fixed to 0.035 and 0.98, respectively.

At the tables that follows, the original problem sizes are given by the number of equality constraints m, the number of rows of each block matrices n (where "d" after a number denotes a diagonal matrix), and the sparsity of problems which can be partially understood by the percentage of the aggregate and extended sparsity patterns (Section 2).

In each of numerical result tables, "standard" means the time to solve the corresponding problem by the SDPA 6.00 or SDPT3 3.02 only, "conversion" is the time to solve the equivalent SDP after its conversion by the original version with a new parameter proposed in Subsection 3.2, and "conversion-fe" is the time to solve the equivalent SDP after its conversion by the version proposed in Subsection 4.1. The numbers between parenthesis are the time for the "conversion" and "conversion-fe" themselves. Entries with "=" mean that the converted SDPs became exactly the same as before the conversion. m_+ is the number of equality constraints and $n_{\max}$ gives the sizes of three largest block matrices after the respective conversion.

We utilized the default parameters both for SDPA 6.00 and SDPT3 3.02 excepting λ^* which was occasionally changed for SDPA 6.00, and OPTIONS.gaptol set to 10^{-7} and OPTIONS.cachesize was set according to the computer for SDPT3 3.02. When the solvers *fail* to solve an instance, we report the relative duality gap denoted by "rg",

$$\begin{array}{cc} \text{(for SDPA)} & \text{(for SDPT3)} \\ \dfrac{|\boldsymbol{A}_0\bullet\boldsymbol{X}-\sum_{p=1}^m b_p y_p|}{\max\{1.0,(|\boldsymbol{A}_0\bullet\boldsymbol{X}|+|\sum_{p=1}^m b_p y_p|)/2\}}, & \dfrac{\boldsymbol{X}\bullet\boldsymbol{S}}{\max\{1.0,(|\boldsymbol{A}_0\bullet\boldsymbol{X}|+|\sum_{p=1}^m b_p y_p|)/2\}}, \end{array} \tag{8}$$

and/or the feasibility error, denoted by "fea",

$$\begin{array}{cc} \text{(primal feasilibity error for SDPA)} & \text{(dual feasilibity error for SDPA)} \\ \max\{|\boldsymbol{A}_p\bullet\boldsymbol{X}-b_p| : p=1,2,\ldots,m\}, & \max\left\{\left|\left[\sum_{p=1}^m \boldsymbol{A}_p y_p+\boldsymbol{S}-\boldsymbol{C}\right]_{ij}\right| : i,j=1,2,\ldots,n\right\}, \\ \text{(primal feasilibity error for SDPT3)} & \text{(dual feasilibity error for SDPT3)} \\ \dfrac{\sqrt{\sum_{p=1}^m(\boldsymbol{A}_p\bullet\boldsymbol{X}-b_p)^2}}{\max\{1.0,\|\boldsymbol{b}\|_2\}}, & \dfrac{\|\sum_{p=1}^m \boldsymbol{A}_p y_p+\boldsymbol{S}-\boldsymbol{C}\|_2}{\max\{1.0,\|\boldsymbol{C}\|_2\}}, \end{array} \tag{9}$$

respectively. For instance "rg"=2 means that the relative duality gap is less than 10^{-2} and "fea"=p6 means that the primal feasibility error is less than 10^{-6}. When "rg" and "fea" are less than the required accuracy 10^{-7}, they are omitted at the tables.

5.1 Benchmark Problems

The sizes of our benchmark SDPs, *i.e.*, norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of max-cut problems over lattice graphs, and SDP relaxation of graph partitioning problems, are shown at Table 2. The original formulation of graph partitioning problems gives a dense aggregate sparsity pattern not allowing us to use the conversion method, and therefore we previously applied an appropriate congruent transformation [8, Section 6] to them.

Table 2: Sizes of norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of the maximum cut problems, and SDP relaxation of graph partition problems.

problem	m	n	aggregate (%)	extended (%)
norm1	11	1000	0.30	0.30
norm2	11	1000	0.50	0.50
norm5	11	1000	1.10	1.10
norm10	11	1000	2.08	2.09
norm20	11	1000	4.02	4.06
norm50	11	1000	9.60	9.84
qp3.0	1001	1001,1000d	0.50	2.83
qp3.5	1001	1001,1000d	0.55	4.56
qp4.0	1001	1001,1000d	0.60	6.43
qp4.5	1001	1001,1000d	0.66	8.55
qp5.0	1001	1001,1000d	0.70	10.41
mcp2×500	1000	1000	0.40	0.50
mcp4×250	1000	1000	0.45	0.86
mcp5×200	1000	1000	0.46	1.03
mcp8×125	1000	1000	0.47	1.38
mcp10×100	1000	1000	0.48	1.57
mcp20×50	1000	1000	0.49	2.12
mcp25×40	1000	1000	0.49	2.25
gpp2×500	1001	1000	0.70	0.70
gpp4×250	1001	1000	1.05	1.10
gpp5×200	1001	1000	1.06	1.30
gpp8×125	1001	1000	1.07	2.39
gpp10×100	1001	1000	1.07	2.94
gpp20×50	1001	1000	1.08	4.97
gpp25×40	1001	1000	1.08	5.31

The discussion henceforth considers the advantages in terms of the computational time.

Tables 3 and 4 give the results for SDPA 6.00 at computer A and B, respectively. For the norm minimization problems, it is advantageous to apply the "conversion". For the SDP relaxations of maximum cut problems and graph partitioning problems, it seems that "conversion-fe" is mostly better than "standard" or "conversion". However, for the SDP

Table 3: Numerical results on norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of the maximum cut problems, and SDP relaxation of graph partition problems for SDPA 6.00 at computer A.

	standard	conversion			conversion-fe		
problem	time (s)	m_+	$n_{\max}$	time (s)	m_+	$n_{\max}$	time (s)
norm1	691.1	77	16,16,16	2.3 (4.3)	58	29,29,29	2.9 (1.8)
norm2	820.2	113	31,31,31	4.7 (4.4)	71	58,58,58	7.4 (2.7)
norm5	1047.4	206	77,77,77	15.4 (4.5)	116	143,143,143	33.6 (9.0)
norm10	1268.8	341	154,154,154	50.4 (5.8)	231	286,286,286	138.1 (34.2)
norm20	1631.7	641	308,308,308	192.6 (8.9)	221	572,448	514.3 (182.9)
norm50	2195.5	1286	770,280	1093.0 (20.2)	11	1000	= (20.0)
qp3.0	895.4	1373	816,22,19	916.1 (34.3)	1219	864,21,18	847.4 (41.8)
qp3.5	891.5	1444	844,20,18	1041.5 (39.3)	1249	875,18,12	890.2 (45.2)
qp4.0	891.0	1636	856,26,20	1294.5 (48.2)	1420	883,20,13	1084.5 (53.4)
qp4.5	891.4	1431	905,15,10	1163.6 (63.2)	1284	930,10,9	1028.3 (68.4)
qp5.0	892.7	1515	909,12,11	1206.3 (74.4)	1381	922,12,11	1045.9 (79.4)
mcp2×500	822.6	1102	31,31,31	91.8 (1.1)	1051	58,58,58	53.0 (4.4)
mcp4×250	719.0	1236	64,63,63	96.4 (2.1)	1204	118,116,115	97.5 (5.7)
mcp5×200	764.9	1395	91,82,82	153.5 (2.5)	1317	156,149,146	125.5 (6.3)
mcp8×125	662.8	1343	236,155,134	100.7 (5.5)	1202	240,236,233	72.3 (12.3)
mcp10×100	701.1	1547	204,172,161	129.0 (7.8)	1196	301,296,227	88.7 (14.7)
mcp20×50	653.1	1657	367,312,307	149.8 (19.3)	1552	570,367,75	221.5 (25.2)
mcp25×40	690.7	1361	622,403,5	246.1 (30.3)	1325	584,441	225.2 (46.1)
gpp2×500	806.1	1133	47,47,47	77.7 (1.7)	1073	86,86,86	53.4 (5.7)
gpp4×250	814.3	1181	136,77,77	64.4 (2.8)	1106	143,143,143	56.5 (8.1)
gpp5×200	807.5	1211	130,93,93	67.7 (3.5)	1106	172,172,172	63.8 (10.4)
gpp8×125	806.1	1472	170,166,159	119.5 (6.8)	1392	319,290,272	144.6 (14.1)
gpp10×100	798.9	1809	236,208,203	195.1 (10.6)	1263	396,339,296	151.2 (23.4)
gpp20×50	799.6	1679	573,443,35	314.5 (18.2)	1379	566,461	293.7 (21.1)
gpp25×40	808.3	1352	526,500	268.3 (39.2)	1904	684,353	436.3 (82.2)

relaxation of quadratic programs, any conversion is not ideal. This result is particularly intriguing since the superiority of the "conversion" was clear when using SDPA 5.0 [16], and SDPA 6.00 mainly differs from SDPA 5.0 in the numerical linear algebra library where Meschach was replaced by ATLAS/LAPACK in the latest version.

Comparing the results at computers A and B, they have similar tendencies excepting that it is faster to solve the norm minimization problems at computer A than computer B due to its cache size.

Similar results can be seen at Tables 5 and 6 for SDPT3 3.02 at computers A and B. However, in this case, "conversion" or "conversion-fe" is better than "standard" on the SDP relaxation of quadratic programs. Also, the computational time for the norm minimization problems at computer A is not faster than computer B as it was observed for SDPA 6.00.

We observe that for "conversion-fe", all of converted problems at Tables 3, 4, 5, and 6 become the same, respectively, excepting for "norm1", "norm2", "norm10", and "mcp25×40".

Summing up, preprocessing by "conversion" or "conversion-fe" produces in the best case a speed up of 8∼147 times for "norm1" when compared with "standard" even considering the time for the conversion itself. And in the worse case "qp4.0", they take only 2.4 times more than "standard".

Table 4: Numerical results on norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of the maximum cut problems, and SDP relaxation of graph partition problems for SDPA 6.00 at computer B.

	standard	conversion			conversion-fe		
problem	time (s)	m_+	$n_{\max}$	time (s)	m_+	$n_{\max}$	time (s)
norm1	303.9	66	19,19,19	1.2 (1.9)	51	29,29,29	1.4 (0.7)
norm2	462.0	95	37,37,37	2.5 (1.8)	68	58,58,58	3.6 (1.3)
norm5	1181.3	176	91,91,91	9.3 (2.3)	116	143,143,143	17.4 (4.2)
norm10	2410.4	286	182,182,182	43.2 (3.4)	176	286,286,286	77.1 (14.9)
norm20	2664.5	431	364,364,312	251.9 (6.0)	221	572,448	1350.2 (110.3)
norm50	2970.3	1286	910,140	2632.2 (15.4)	11	1000	= (15.2)
qp3.0	349.4	1287	843,22,19	493.3 (18.5)	1219	864,21,18	447.9 (20.7)
qp3.5	348.1	1391	853,20,18	580.7 (22.4)	1249	875,18,12	474.6 (24.7)
qp4.0	347.8	1601	861,26,20	789.2 (32.4)	1420	883,20,13	631.9 (34.6)
qp4.5	349.0	1399	915,15,10	626.9 (46.9)	1284	930,10,9	540.3 (49.1)
qp5.0	347.5	1514	910,12,11	709.5 (59.0)	1381	922,12,11	574.6 (60.3)
mcp2×500	268.3	1084	37,37,37	49.2 (0.7)	1051	58,58,58	35.1 (2.2)
mcp4×250	234.2	1204	85,75,75	56.0 (1.2)	1204	118,116,115	67.2 (2.9)
mcp5×200	250.7	1295	106,96,94	73.0 (1.6)	1317	156,149,146	95.2 (3.2)
mcp8×125	221.5	1340	160,155,154	52.7 (3.2)	1202	240,236,233	36.2 (6.8)
mcp10×100	233.1	1615	233,201,187	101.5 (4.0)	1196	301,296,227	42.1 (8.2)
mcp20×50	215.6	1330	581,392,62	84.2 (8.3)	1552	570,367,75	102.8 (14.6)
mcp25×40	228.4	1325	567,458	87.2 (18.3)	1406	650,378	100.5 (33.5)
gpp2×500	265.5	1109	55,55,55	44.7 (1.0)	1073	86,86,86	31.9 (2.8)
gpp4×250	267.1	1151	140,91,91	34.7 (1.8)	1106	143,143,143	29.9 (4.4)
gpp5×200	265.3	1169	168,110,110	34.0 (2.3)	1106	172,172,172	32.5 (5.5)
gpp8×125	265.0	1337	202,177,176	49.2 (4.7)	1392	319,290,272	93.3 (7.6)
gpp10×100	253.4	1536	277,277,242	107.9 (7.0)	1263	396,339,296	62.0 (13.5)
gpp20×50	262.4	2030	512,491,39	179.7 (9.4)	1379	566,461	112.3 (10.9)
gpp25×40	264.1	1352	526,500	103.2 (22.1)	1904	689,353	183.7 (47.6)

5.2 SDPLIB Problems

The next set of problems are from the SDPLIB 1.2 collection [4]. We selected the problems which have sparse aggregate patterns including the ones after the congruent transformation [8, Section 6] like "equalG", "gpp", and "theta" problems. We excluded the largest instances. Problem sizes and sparsity information are shown at Table 7. Observe that in several cases, the fill-in effect causes the extended sparsity patterns to become *much denser* than the corresponding aggregate sparsity patterns.

We can observe from the numerical results at Tables 8, 9, 10, and 11 that the conversion method is advantageous when the extended sparsity patterns are less than 5%, which is the case for "equalG11", "maxG11", "maxG32", "mcp124-1", "mcp250-1", "mcp500-1", "qpG11", "qpG51", and "thetaG11". The exception are "mcp124-1" and "mcp250-1" at Table 10, and "mcp124-1" at Table 11. In particular, it is difficult to say which version of the conversion method is ideal in general, but it seems that "conversion" is particularly better for "maxG32", and "conversion-fe" is better for "equalG11", "maxG11", and "qpG11".

Once again, the converted problems under the columns "conversion-fe" are exactly the same for the Tables 8, 9, 10, and 11 respectively, excepting "arch" problems, "equalG11", "gpp124-1", "gpp124-2", "maxG11", "mcp250-4", "mcp500-1", "qpG11", and "ss30".

For "qpG11" we have a speed up of 6.4∼ 19.3 times when preprocessed by "conversion" or "conversion-fe" even considering the time for the conversion itself. On the other hand,

Table 5: Numerical results on norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of the maximum cut problems, and SDP relaxation of graph partition problems for SDPT3 3.02 at computer A.

	standard		conversion				conversion-fe			
problem	time (s)	rg fea	m_+	$n_{\max}$	time (s)	fea	m_+	$n_{\max}$	time (s)	fea
norm1	234.7		110	11,11,11	10.5 (6.0)		53	29,29,29	20.9 (1.5)	p6
norm2	330.7		158	22,22,22	23.9 (5.1)	p6	68	58,58,58	26.2 (2.6)	
norm5	418.5		311	53,53,53	51.5 (4.9)	p6	116	143,143,143	59.3 (8.8)	p6
norm10	507.7		561	106,106,106	85.4 (6.1)	p6	176	286,286,286	108.3 (33.7)	
norm20	686.2		1061	211,211,211	178.4 (9.3)		221	572,448	325.2 (181.1)	
norm50	1295.5		1286	527,523	730.2 (20.9)		11	1000	= (20.3)	
qp3.0	466.3		1717	769,57,22	333.6 (31.0)		1219	864,21,18	336.8 (41.8)	
qp3.5	470.3		1616	810,20,18	324.8 (36.7)		1249	875,18,12	328.6 (44.9)	
qp4.0	500.6		1913	835,26,22	436.0 (46.6)		1420	883,13,10	397.7 (53.5)	
qp4.5	492.9		1681	875,15,14	449.0 (61.3)		1284	930,10,9	469.0 (68.4)	
qp5.0	487.6		1929	878,20,20	460.6 (71.7)		1381	922,12,11	419.4 (79.4)	
mcp2×500	264.0	2 p2	1144	22,22,22	24.9 (0.9)		1051	58,58,58	66.7 (4.5)	
mcp4×250	297.7		1334	72,45,45	108.5 (1.6)		1204	118,116,115	67.6 (5.7)	
mcp5×200	307.5		1315	92,68,58	91.5 (2.0)		1317	156,149,146	79.1 (6.2)	
mcp8×125	272.4		1494	95,94,94	89.6 (3.5)		1202	240,236,233	66.5 (12.3)	
mcp10×100	295.4		1616	150,126,114	92.4 (4.2)		1196	301,296,227	66.7 (14.7)	
mcp20×50	270.4		1630	349,252,247	116.8 (10.9)		1552	570,367,75	143.7 (25.2)	
mcp25×40	270.2		2031	328,307,280	179.0 (18.0)		1406	650,378	151.0 (58.9)	
gpp2×500	432.5		1205	32,32,32	136.1 (1.3)		1073	86,86,86	75.9 (5.6)	
gpp4×250	436.8		1301	53,53,53	105.4 (2.1)		1106	143,143,143	78.1 (8.1)	
gpp5×200	414.8		1337	72,64,64	98.4 (2.6)		1106	172,172,172	87.7 (10.4)	
gpp8×125	430.0		1790	183,126,121	137.4 (5.2)		1392	319,290,272	127.2 (14.1)	
gpp10×100	402.9		1824	188,158,150	139.3 (7.0)		1263	396,339,296	129.6 (23.3)	
gpp20×50	383.4		2599	338,298,287	419.7 (15.7)		1379	566,461	184.7 (21.2)	
gpp25×40	388.0		1703	411,380,261	167.8 (21.4)		1904	689,353	307.8 (82.2)	

the worse case goes to "mcp250-1" which takes only 1.6 times more than "standard" when restricting problems with less than 5% on their extended sparsity patterns.

5.3 Structural Optimization Problems

The last set of problems are from the structural optimization [12] which have sparse aggregate sparsity patterns. Problem sizes and sparsity information are shown at Table 12.

The numerical results for these four classes of problems for SDPA 6.00 and SDPT3 3.02 at computers A and B are shown at Tables 13, 14, 15, and 16. Entries with "M" means out of memory and "*" means fail to solve.

Among these four classes, the conversion method is only advantageous for the "shmup" problems. In particular, "conversion" and "conversion-fe" have similar performances over "standard" for SDPA 6.00 at computers A and B (Tables 13 and 14), while "conversion" is better than "conversion-fe" and "standard" (excepting "shmup5") for SDPT3 3.02 at computers A and B (Tables 15 and 16).

We observe here that both SDPA 6.00 and SDPT3 3.02 have some difficult to solve the converted problems, *i.e.*, "conversion" and "conversion-fe", and get the same accuracy as "standard", suggesting that the conversion itself can sometimes cause some negative effect when solving by interior-point methods. In some cases like "vibra5" for "conversion-fe", SDPT3 3.02 fails to solve them (Tables 15 and 16). However, these structural optimal

Table 6: Numerical results on norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of the maximum cut problems, and SDP relaxation of graph partition problems for SDPT3 3.02 at computer B.

	standard		conversion				conversion-fe			
problem	time (s)	rg fea	m_+	$n_{\max}$	time (s)	fea	m_+	$n_{\max}$	time (s)	rg fea
norm1	115.2		101	12,12,12	7.2 (2.5)	p5	53	29,29,29	14.3 (0.7)	p6
norm2	162.1		146	24,24,24	18.1 (2.1)		68	58,58,58	15.8 (1.3)	6
norm5	209.0		281	59,59,59	29.1 (2.5)	p6	116	143,143,143	36.9 (4.2)	p6
norm10	259.4		506	118,118,118	51.0 (3.5)	p6	176	286,286,286	69.5 (15.0)	
norm20	414.3		851	236,236,236	112.1 (6.1)		221	572,448	180.9 (110.5)	
norm50	746.6		1286	589,461	446.0 (15.6)		11	1000	= (15.4)	
qp3.0	260.1		1397	807,22,19	177.9 (17.1)		1219	864,21,18	191.4 (20.8)	
qp3.5	258.9		1591	806,20,18	180.4 (20.7)		1249	875,18,12	188.4 (24.8)	
qp4.0	275.0		1947	823,26,22	242.5 (31.0)		1420	883,20,13	227.3 (34.4)	
qp4.5	267.8		1613	886,15,14	251.5 (45.4)		1284	930,10,9	260.0 (48.5)	
qp5.0	265.6		1856	887,20,20	256.5 (57.9)		1381	922,12,11	233.3 (60.9)	
mcp2×500	140.4	2 p2	1132	32,24,24	14.5 (0.5)		1051	58,58,58	36.4 (2.2)	
mcp4×250	160.8		1398	51,51,49	59.6 (0.9)		1204	118,116,115	37.3 (2.9)	
mcp5×200	165.2		1476	67,66,65	52.6 (1.1)		1317	156,149,146	44.8 (3.2)	
mcp8×125	148.7		1421	138,114,111	44.3 (2.3)		1202	240,236,233	40.1 (6.8)	
mcp10×100	160.9		1764	171,167,141	53.8 (2.8)		1196	301,296,227	42.4 (8.2)	
mcp20×50	147.6		1801	297,280,252	73.7 (5.9)		1552	570,367,75	81.7 (14.6)	
mcp25×40	147.9		2112	346,328,307	97.6 (11.7)		1406	650,378	86.8 (33.4)	
gpp2×500	233.8		1181	36,36,36	67.8 (0.8)		1073	86,86,86	42.3 (2.9)	
gpp4×250	235.8		1271	59,59,59	51.3 (1.3)		1106	143,143,143	43.7 (4.4)	
gpp5×200	225.9		1295	90,71,71	49.7 (1.6)		1106	172,172,172	51.0 (5.5)	
gpp8×125	232.7		1645	132,131,130	70.5 (3.2)		1392	319,290,272	84.1 (7.6)	
gpp10×100	218.1		1949	186,183,165	98.8 (4.8)		1263	396,339,296	84.5 (13.5)	
gpp20×50	209.9		2347	370,352,319	166.3 (15.0)		1379	566,461	106.6 (10.9)	
gpp25×40	216.0		1703	411,380,261	96.2 (12.5)		1904	689,353	176.2 (47.4)	

problems are difficult to solve by their nature (see "rg" and "fea" columns under "standard").

Once again, the converted problems under the columns "conversion-fe" have similar sizes. In particular the converted problems are exactly the same for Tables 13, 14, 15, and 16 for "buck5", "shmup3∼5", "trto4∼5" and "vibra5", respectively.

Although it is difficult to make direct comparisons due to the difference in accuracies, it seems in general that "conversion-fe" is better than "conversion" for worse case scenarios when preprocessing *fails*.

6 Concluding and Further Remarks

As we stated in Introduction, the conversion method is a preprocessing phase of SDP solvers for sparse and large-scale SDPs. We slightly improved the original version [16] here, and proposed a new version of the conversion method which attempts to produce the best equivalent SDP in terms of shortening the computational time. A flop estimation function was introduced to be used in the heuristic routine in the new version. Extensive numerical computation using SDPA 6.00 and SDPT3 3.02 were conducted comparing the computational time for different sets of SDPs on different computers using the original version of the conversion, "conversion", the flop estimation version, "conversion-fe", and SDPs without preprocessing, "standard". In some cases, the results were astonishing: "norm1" became

Table 7: Sizes of SDPLIB problems.

problem	m	n	aggregate (%)	extended (%)
arch0	174	161,174d	11.44	24.18
arch2	174	161,174d	12.93	23.34
arch4	174	161,174d	12.93	23.34
arch8	174	161,174d	12.93	23.34
equalG11	801	801	1.24	(A) 4.32, (B) 4.40
equalG51	1001	1001	4.59	(A) 52.99, (B) 53.37
gpp100	101	100	20.34	61.66
gpp124-1	125	124	9.53	33.18
gpp124-2	125	124	16.70	56.26
gpp124-3	125	124	28.12	74.26
gpp124-4	125	124	44.61	89.19
gpp250-1	251	250	5.27	31.02
gpp250-2	251	250	8.51	52.00
gpp250-3	251	250	16.09	72.31
gpp250-4	251	250	26.84	84.98
gpp500-1	501	500	2.56	28.45
gpp500-2	501	500	4.42	46.54
gpp500-3	501	500	7.72	66.25
gpp500-4	501	500	15.32	83.06
maxG11	800	800	0.62	2.52
maxG32	2000	2000	0.25	1.62
maxG51	1000	1000	1.28	13.39
mcp100	100	100	6.38	19.88
mcp124-1	124	124	2.74	4.75
mcp124-2	124	124	4.94	16.99
mcp124-3	124	124	8.87	38.20
mcp124-4	124	124	17.34	64.58
mcp250-1	250	250	1.46	3.65
mcp250-2	250	250	2.36	14.04
mcp250-3	250	250	4.51	34.09
mcp250-4	250	250	8.15	57.10
mcp500-1	500	500	0.70	2.13
mcp500-2	500	500	1.18	10.78
mcp500-3	500	500	2.08	27.94
mcp500-4	500	500	4.30	52.89
qpG11	800	1600	0.19	0.68
qpG51	1000	2000	0.35	3.36
ss30	132	294,132d	8.81	18.71
theta1	104	50	32.24	57.68
theta2	498	100	36.08	77.52
theta3	1106	150	35.27	85.40
theta4	1949	200	34.71	85.89
theta5	3028	250	34.09	89.12
theta6	4375	300	34.42	90.36
thetaG11	2401	801	1.62	4.92
thetaG51	6910	1001	4.93	53.65

(A): computer A, (B): computer B.

Table 8: Numerical results on SDPLIB problems for SDPA 6.00 at computer A.

	standard	conversion			conversion-fe		
problem	time (s) rg fea	m_+	$n_{\max}$	time (s) rg fea	m_+	$n_{\max}$	time (s) rg fea
arch0	13.7	174	161,174d	= (0.4)	552	124,64,174d	21.5 (0.6) 6 p6
arch2	15.6	174	161,174d	= (0.4)	405	100,82,174d	12.6 (0.5) 5 p5
arch4	15.6	174	161,174d	= (0.4)	405	100,82,174d	12.4 (0.4) 5
arch8	17.4 6	174	161,174d	= (0.4)	405	100,82,174d	12.5 (0.5) 4 p4
equalG11	455.2	1064	408,408,9	158.1 (14.3) 5 p6	1314	219,218,209	82.9 (7.7) 5
equalG51	865.1	2682	998,52,29	1234.6 (679.6) 6	1407	1000,29	900.5 (681.3)
gpp100	1.2	101	100	= (0.2)	101	100	= (0.2)
gpp124-1	2.2	125	124	= (0.2)	161	122,10	2.2 (0.2) 6
gpp124-2	2.1	146	123,7	2.3 (0.4)	146	123,7	2.1 (0.4)
gpp124-3	2.0	125	124	= (0.5)	125	124	= (0.5)
gpp124-4	2.1	125	124	= (0.7)	125	124	= (0.7)
gpp250-1	14.9	272	249,7	16.9 (1.6)	251	250	= (1.8)
gpp250-2	14.6	251	250	= (3.3)	251	250	= (3.4)
gpp250-3	13.5	251	250	= (5.0)	251	250	= (5.0)
gpp250-4	13.3	251	250	= (6.9)	251	250	= (6.9)
gpp500-1	125.5	830	493,17,13	114.0 (16.1) 5	537	499,9	124.0 (16.7) 5
gpp500-2	115.1	1207	496,30,26	132.4 (41.4) 5	501	500	= (37.2)
gpp500-3	108.4	1005	498,27,18	127.4 (56.7) 5	654	499,18	111.0 (57.1) 6
gpp500-4	99.0	501	500	= (88.7)	501	500	= (88.8)
maxG11	343.7	972	408,403,13	113.3 (10.7)	1208	216,216,208	60.8 (8.2)
maxG32	5526.4	2840	1021,1017,5	1704.3 (274.5)	2000	2000	= (358.5)
maxG51	651.8	2033	957,16,15	867.6 (84.4)	1677	971,15,15	756.8 (88.5)
mcp100	0.8	123	93,5,5	0.9 (0.1)	107	95,5,3	0.8 (0.1)
mcp124-1	1.6	129	108,4,2	1.2 (0.1)	184	60,48,13	0.9 (0.1)
mcp124-2	1.5	163	117,5,5	1.6 (0.1)	144	120,5,4	1.5 (0.2)
mcp124-3	1.5	160	122,7,6	1.6 (0.2)	124	124	= (0.3)
mcp124-4	1.5	124	124	= (0.4)	124	124	= (0.4)
mcp250-1	10.9	268	217,7,3	8.5 (0.5)	401	176,58,7	8.3 (0.5)
mcp250-2	10.6	342	233,12,5	11.9 (0.9)	327	237,12,5	11.2 (1.0)
mcp250-3	10.6	444	243,11,11	14.1 (1.9)	277	247,6,4	10.6 (2.1)
mcp250-4	10.5	305	249,11	11.3 (3.6)	305	249,11	11.6 (3.6)
mcp500-1	87.6	545	403,10,10	63.1 (3.6)	1271	324,124,10	115.2 (3.3)
mcp500-2	87.9	899	436,14,12	127.7 (6.4)	784	449,15,9	109.2 (7.3)
mcp500-3	82.4	1211	477,18,16	155.6 (17.5)	1019	482,16,13	128.2 (18.2)
mcp500-4	82.6	2013	491,20,20	246.8 (46.1)	758	498,18,15	89.5 (46.8)
qpG11	2612.5	946	419,396,5	181.6 (15.6)	1208	219,216,213	187.6 (12.3)
qpG51	5977.3	2243	947,16,15	1830.4 (88.4)	1838	965,16,15	1357.1 (99.8)
ss30	99.1	132	294,132d	= (1.1)	1035	171,165,132d	67.0 (1.5) 4 p6
theta1	0.4	104	50	= (0.0)	104	50	= (0.0)
theta2	7.8	498	100	= (0.5)	498	100	= (0.5)
theta3	43.8	1106	150	= (2.1)	1106	150	= (2.2)
theta4	184.9 6	1949	200	= (6.2)	1949	200	= (6.7)
theta5	581.0 6	3028	250	= (15.0)	3028	250	= (15.7)
theta6	1552.9 6	4375	300	= (31.1)	4375	300	= (32.2)
thetaG11	1571.8	2743	315,280,242	852.8 (31.7)	2572	417,402	937.9 (51.1)
thetaG51	24784.7 3 p5	7210	1000,25	* (817.8)	6910	1001	= (855.9)

8∼147 times faster, and "qpG11" became 6∼19 faster when compared with "standard".

Unfortunately, it seems that each SDP prefers one of the versions of the conversion method. However, we can say in general that both conversions are advantageous to use when the extended sparsity pattern is less than 5%, and even in abnormal cases, like in structural optimization problems where getting the feasibility is difficult, the computational time takes at most 4 times more than solving without any conversion.

In practical matters, it is really worth to preprocess by "conversion" or "conversion-fe"

Table 9: Numerical results on SDPLIB problems for SDPA 6.00 at computer B.

	standard		conversion				conversion-fe			
problem	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg	m_+	$n_{\max}$	time (s)	rg
arch0	6.8		174	161,174d	= (0.2)		174	161,174d	= (0.3)	
arch2	7.7		174	161,174d	= (0.2)		174	161,174d	= (0.3)	
arch4	7.6		174	161,174d	= (0.2)		174	161,174d	= (0.3)	
arch8	8.2	6	174	161,174d	= (0.2)	6	174	161,174d	= (0.3)	6
equalG11	153.5		1008	409,409,9	57.0 (8.0)	5	972	410,409	51.7 (8.5)	5
equalG51	284.3		2682	998,52,29	498.8 (762.4)	6	1407	1000,29	318.9 (763.6)	6
gpp100	0.5		101	100	= (0.2)		101	100	= (0.2)	
gpp124-1	0.9		125	124	= (0.1)		125	124	= (0.2)	
gpp124-2	0.9		146	123,7	1.0 (0.3)		125	124	= (0.3)	
gpp124-3	0.9		125	124	= (0.4)		125	124	= (0.4)	
gpp124-4	0.9		125	124	= (0.6)		125	124	= (0.6)	
gpp250-1	6.4		272	249,7	6.5 (1.3)		251	250	= (1.4)	
gpp250-2	6.1		251	250	= (3.2)		251	250	= (3.2)	
gpp250-3	5.9		251	250	= (4.9)		251	250	= (4.9)	
gpp250-4	5.7		251	250	= (7.0)		251	250	= (7.1)	
gpp500-1	45.6		752	494,17,11	54.4 (14.9)	5	537	499,9	45.6 (15.2)	5
gpp500-2	42.0		801	498,26	47.7 (41.6)	6	501	500	= (39.2)	
gpp500-3	39.5		1005	498,27,18	57.0 (61.8)	6	654	499,18	42.0 (62.1)	6
gpp500-4	36.1		501	500	= (98.7)		501	500	= (99.0)	
maxG11	118.6		936	408,408	44.0 (6.0)		1072	408,216,208	37.6 (5.8)	
maxG32	1652.1		2820	1022,1018	618.9 (150.4)		2000	2000	= (182.2)	
maxG51	216.2		1868	964,16,15	432.6 (73.9)		1677	971,15,15	345.9 (75.6)	
mcp100	0.4		107	95,5,3	0.4 (0.0)		107	95,5,3	0.4 (0.1)	
mcp124-1	0.7		129	108,4,2	0.6 (0.0)		184	60,48,13	0.5 (0.0)	
mcp124-2	0.7		157	118,5,5	0.8 (0.1)		144	120,5,4	0.7 (0.1)	
mcp124-3	0.7		139	123,6	0.7 (0.2)		124	124	= (0.2)	
mcp124-4	0.7		124	124	= (0.3)		124	124	= (0.3)	
mcp250-1	4.8		268	217,7,3	4.1 (0.3)		401	176,58,7	4.4 (0.3)	
mcp250-2	4.6		336	235,12,5	5.7 (0.6)		327	237,12,5	5.5 (0.6)	
mcp250-3	4.7		444	243,11,11	6.8 (1.6)		277	247,6,4	4.8 (1.7)	
mcp250-4	4.6		305	249,11	5.2 (3.2)		250	250	= (3.1)	
mcp500-1	32.8		539	405,10,10	29.2 (1.9)		530	410,10,10	28.7 (2.2)	
mcp500-2	32.9		862	439,14,12	70.7 (4.5)		784	449,15,9	56.8 (4.9)	
mcp500-3	30.9		1175	478,18,16	114.0 (17.0)		1019	482,16,13	63.8 (17.3)	
mcp500-4	30.9		1823	492,20,20	127.2 (50.2)		758	498,18,15	35.0 (50.5)	
qpG11	810.4		946	419,396,5	103.7 (8.7)		1072	397,219,216	118.2 (9.1)	
qpG51	1735.8		2180	950,16,15	1389.6 (77.2)		1838	965,16,15	904.0 (81.0)	
ss30	52.9	p6	132	294,132d	= (0.7)		132	294,132d	= (0.8)	
theta1	0.2		104	50	= (0.0)		104	50	= (0.0)	
theta2	3.4		498	100	= (0.3)		498	100	= (0.4)	
theta3	22.5	6	1106	150	= (1.5)	6	1106	150	= (1.6)	
theta4	91.0	6	1949	200	= (4.6)	6	1949	200	= (4.9)	
theta5	264.9	6	3028	250	= (11.6)	6	3028	250	= (11.6)	
theta6	659.9	6	4375	300	= (24.4)	6	4375	300	= (25.1)	
thetaG11	684.4		2743	362,330,145	508.2 (20.5)		2572	417,402	501.4 (28.9)	
thetaG51	11120.2	3 p5	7210	1000,25	* (754.4)		6910	1001	= (776.0)	

which provides an enormous shortening of time for sparse SDPs, that can be 10∼100 times faster in some cases, even having the risk to face *bad* problems, that can take 2 times more to solve in most of cases.

Here we omitted the discussion on used memory, but its advantage is clear in general when the computational time is good as well [16].

We have a general impression that the performance of "conversion-fe" is better than "conversion" in the worse case scenarios, when solving the original problem is slightly faster, for all the numerical experiments we completed. A minor remark is that "conversion-fe"

Table 10: Numerical results on SDPLIB problems for SDPT3 3.02 at computer A.

	standard			conversion					conversion-fe				
problem	time (s)	rg	fea	m_+	$n_{\max}$	time (s)	rg	fea	m_+	$n_{\max}$	time (s)	rg	fea
arch0	22.6	6	p5	174	161,174d	= (0.4)			174	161,174d	= (0.5)		
arch2	21.5	6	p6	174	161,174d	= (0.4)			174	161,174d	= (0.5)		
arch4	21.8			174	161,174d	= (0.4)			174	161,174d	= (0.5)		
arch8	20.8			174	161,174d	= (0.4)			174	161,174d	= (0.5)		
equalG11	234.2	6	p6	2511	203,202,202	286.8 (6.8)	5	p4	1143	410,218,209	103.6 (12.2)	5	p5
equalG51	861.8			2682	998,52,29	1425.7 (679.5)	5	p5	1407	1000,29	1103.2 (681.0)		p6
gpp100	3.1			101	100	= (0.2)			101	100	= (0.2)		
gpp124-1	5.4			161	122,10	6.5 (0.2)	5	p5	125	124	= (0.2)		
gpp124-2	5.7		p6	146	123,7	6.0 (0.4)			146	123,7	6.0 (0.4)		
gpp124-3	4.3			125	124	= (0.5)			125	124	= (0.5)		
gpp124-4	5.0		p6	125	124	= (0.7)			125	124	= (0.7)		
gpp250-1	15.8	5	p5	282	248,7,5	21.1 (1.7)	6	p5	251	250	= (1.8)		
gpp250-2	16.8			251	250	= (3.3)			251	250	= (3.4)		
gpp250-3	17.7		p6	251	250	= (4.9)			251	250	= (5.0)		
gpp250-4	14.9			251	250	= (6.9)			251	250	= (6.9)		
gpp500-1	108.2	6		908	492,17,13	163.3 (16.1)	5	p4	537	499,9	120.6 (16.7)	5	
gpp500-2	102.0	6		1312	495,30,26	172.4 (41.1)	5	p6	501	500	= (37.3)		
gpp500-3	122.7	6	p6	1356	496,28,27	198.0 (56.6)		p4	654	499,18	129.9 (57.1)	5	p6
gpp500-4	119.5	6	p6	501	500	= (88.5)	6	p6	501	500	= (88.7)		
maxG11	178.9			2228	178,178,178	162.1 (5.3)			1208	216,216,208	62.2 (8.2)		
maxG32	2017.6			5310	528,526,513	1908.2 (161.1)			2000	2000	= (357.0)		
maxG51	531.5			2501	937,16,15	658.5 (82.3)			1677	971,15,15	550.7 (88.7)		
mcp100	3.0			141	90,7,5	3.6 (0.1)			107	95,5,3	3.3 (0.1)		
mcp124-1	3.3			167	85,18,11	3.2 (0.1)			184	60,48,13	3.4 (0.1)		
mcp124-2	3.6			157	118,5,5	4.0 (0.2)			144	120,5,4	4.3 (0.2)		
mcp124-3	4.2			188	121,8,7	5.0 (0.2)			124	124	= (0.3)		
mcp124-4	4.6			124	124	= (0.4)			124	124	= (0.4)		
mcp250-1	9.3			436	170,56,11	14.7 (0.4)			401	176,58,7	13.8 (0.5)		
mcp250-2	10.4			342	233,12,5	11.3 (0.9)			327	237,12,5	11.4 (1.1)		
mcp250-3	12.9			472	242,11,11	17.1 (1.9)			277	247,6,4	13.5 (2.1)		
mcp250-4	13.4			736	246,20,17	22.9 (3.5)			250	250	= (3.4)		
mcp500-1	51.5			582	388,11,10	35.2 (3.3)			530	410,10,10	37.3 (4.4)		
mcp500-2	64.3			987	430,15,14	71.3 (6.3)			784	449,15,9	64.9 (7.3)		
mcp500-3	81.6			1566	467,18,18	142.4 (17.2)			1019	482,16,13	96.8 (18.2)		
mcp500-4	83.9			2800	486,34,20	370.0 (46.1)			758	498,18,15	89.7 (46.8)		
qpG11	1835.3			1364	218,208,204	84.4 (10.6)			1072	397,219,216	79.7 (16.0)		
qpG51	3876.9			2472	937,16,15	652.9 (87.6)			1838	965,16,15	527.1 (99.9)		
ss30	50.0	6	p6	132	294,132d	= (1.1)			132	294,132d	= (1.3)		
theta1	1.2	5		104	50	= (0.0)			104	50	= (0.0)		
theta2	9.4	5	p5	498	100	= (0.5)			498	100	= (0.5)		
theta3	50.4	4	p4	1106	150	= (2.1)			1106	150	= (2.3)		
theta4	183.2	6	p5	1949	200	= (6.2)			1949	200	= (6.7)		
theta5	579.2	6	p5	3028	250	= (15.0)			3028	250	= (15.7)		
theta6	1562.5	5	p5	4375	300	= (31.1)			4375	300	= (32.2)		
thetaG11	1426.1			2914	224,224,211	1579.4 (26.0)	6	p5	2572	417,402	1106.7 (51.1)		p6
thetaG51	*			7210	1000,25	* (817.8)			6910	1001	* (856.0)		

produces in general similar SDPs in terms of sizes independent of computers and solvers which indicates that "conversion-fe" relies on how we define the flop estimation function.

It also remains the difficult question if it is possible to obtain homogeneous matrix sizes for the converted SDPs (see columns $n_{\max}$).

As proposed in Introduction, the conversion method should be considered as a first step to include preprocessing in SDP solvers. A very further project considers incorporating the conversion method in SDPA [25] and SDPARA [26] together with the completion method

Table 11: Numerical results on SDPLIB problems for SDPT3 3.02 at computer B.

	standard		conversion				conversion-fe			
problem	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg fea
arch0	13.1	6 p6	174	161,174d	= (0.2)		174	161,174d	= (0.3)	
arch2	12.9	6 p6	174	161,174d	= (0.2)		405	100,82,174d	10.6 (0.3)	4 p5
arch4	13.1		174	161,174d	= (0.2)		405	100,82,174d	10.6 (0.3)	5 p6
arch8	12.6		174	161,174d	= (0.2)		405	100,82,174d	11.1 (0.3)	1 p4
equalG11	132.2	6	1979	214,213,205	161.0 (3.8)	5 p4	972	410,409	77.3 (8.7)	6 p5
equalG51	477.9		2682	998,52,29	750.1 (762.4)	5 p6	1407	1000,29	589.8 (762.6)	6 p5
gpp100	2.0		101	100	= (0.2)		101	100	= (0.2)	
gpp124-1	3.0	6 p6	161	122,10	4.1 (0.2)	5 p4	125	124	= (0.2)	
gpp124-2	3.4	6 p6	146	123,7	3.6 (0.3)	6	146	123,7	3.6 (0.3)	p6
gpp124-3	2.7		125	124	= (0.4)		125	124	= (0.4)	
gpp124-4	3.1		125	124	= (0.6)		125	124	= (0.6)	
gpp250-1	12.4	5 p5	282	248,7,5	16.6 (1.3)	6 p6	251	250	= (1.4)	
gpp250-2	12.3	6 p6	251	250	= (3.1)		251	250	= (3.2)	
gpp250-3	13.7	6 p6	251	250	= (4.9)		251	250	= (5.0)	
gpp250-4	11.2		251	250	= (7.0)		251	250	= (7.1)	
gpp500-1	64.0	5 p5	908	492,17,13	97.0 (15.0)	4	537	499,9	80.5 (15.2)	6 p6
gpp500-2	68.5		1312	495,30,26	105.5 (41.7)	5 p5	501	500	= (39.2)	
gpp500-3	71.1	6 p6	1005	498,27,18	97.3 (61.9)	5 p5	654	499,18	84.7 (62.0)	5 p5
gpp500-4	78.4		501	500	= (98.9)		501	500	= (98.8)	
maxG11	98.5		1808	195,192,190	85.4 (3.2)		1072	408,216,208	48.9 (5.8)	
maxG32	1054.3		5310	528,526,513	1034.0 (91.7)		2000	2000	= (182.2)	
maxG51	287.7		2414	941,16,15	347.5 (72.2)		1677	971,15,15	306.8 (75.7)	
mcp100	2.1		138	91,7,5	2.3 (0.0)		107	95,5,3	2.1 (0.1)	
mcp124-1	2.2		145	97,13,5	2.3 (0.0)		184	60,48,13	2.1 (0.1)	
mcp124-2	2.3		163	117,5,5	2.4 (0.1)		144	120,5,4	2.7 (0.1)	
mcp124-3	2.7		188	121,8,7	3.1 (0.2)		124	124	= (0.2)	
mcp124-4	3.0		124	124	= (0.3)		124	124	= (0.3)	
mcp250-1	8.1		305	209,9,7	6.6 (0.3)		401	176,58,7	8.5 (0.3)	
mcp250-2	8.8		420	228,17,12	9.1 (0.5)		327	237,12,5	8.8 (0.6)	
mcp250-3	11.6		597	241,19,11	13.6 (1.6)		277	247,6,4	10.7 (1.7)	
mcp250-4	12.0		495	248,20,11	12.5 (3.2)		250	250	= (3.1)	
mcp500-1	35.7		564	395,11,10	22.9 (1.7)		530	410,10,10	24.7 (2.2)	
mcp500-2	45.7		903	435,14,12	42.0 (4.5)		784	449,15,9	41.5 (4.9)	
mcp500-3	59.1		1455	471,18,18	80.3 (16.9)		1019	482,16,13	60.6 (17.3)	
mcp500-4	60.4		2800	486,34,20	209.5 (50.0)		758	498,18,15	58.5 (50.6)	
qpG11	966.6		1354	219,208,204	56.5 (6.0)		1072	397,219,216	52.8 (9.1)	
qpG51	1901.4		2372	941,16,15	342.5 (77.1)		1838	965,16,15	292.5 (80.6)	
ss30	33.5	6 p6	132	294,132d	= (0.7)		132	294,132d	= (0.8)	
theta1	0.8	5	104	50	= (0.0)		104	50	= (0.0)	
theta2	5.6	6 p6	498	100	= (0.3)		498	100	= (0.4)	
theta3	29.4	5 p6	1106	150	= (1.5)		1106	150	= (1.6)	
theta4	107.7	5 p5	1949	200	= (4.6)		1949	200	= (4.8)	
theta5	333.0	6 p6	3028	250	= (11.7)		3028	250	= (11.9)	
theta6	895.3	5 p5	4375	300	= (24.4)		4375	300	= (25.2)	
thetaG11	823.3		2743	389,224,224	626.2 (15.8)	6	2572	417,402	593.4 (29.0)	p6
thetaG51	*		7210	1000,25	* (756.4)		6910	1001	* (776.1)	

[8, 16, 17], and further develop theoretical and practical algorithms to explore sparsity and eliminate degeneracies.

Acknowledgment

The second author is grateful for his funding from the National Science Foundation (NSF) via Grant No. ITR-DMS 0113852, and also to Douglas N. Arnold and Masakazu Kojima

Table 12: Sizes of structural optimization problems.

problem	m	n	aggregate (%)	extended (%)
buck1	36	24,25,36d	35.65	42.44
buck2	144	96,97,144d	10.60	13.94
buck3	544	320,321,544d	3.67	7.40
buck4	1200	672,673,1200d	1.85	5.14
buck5	3280	1760,1761,3280d	0.74	2.98
shmup1	16	41,40,32d	34.80	40.48
shmup2	200	441,440,400d	4.03	10.26
shmup3	420	901,900,840d	2.03	6.24
shmup4	800	1681,1680,1600d	1.11	4.27
shmup5	1800	3721,3720,3600d	0.51	2.49
trto1	36	25,36d	39.79	42.81
trto2	144	97,144d	12.17	14.12
trto3	544	321,544d	4.22	7.50
trto4	1200	673,1200d	2.12	5.52
trto5	3280	1761,3280d	0.85	3.01
vibra1	36	24,25,36d	35.65	42.44
vibra2	144	96,97,144d	10.60	13.94
vibra3	544	320,321,544d	3.67	7.40
vibra4	1200	672,673,1200d	1.85	5.14
vibra5	3280	1760,1761,3280d	0.74	2.98

from Institute for Mathematics and its Applications and Tokyo Institute of Technology for inviting him as a visitor researcher at their respective institutes making this research possible. In particular, Masakazu Kojima contributed with valuable comments on this paper. The authors would like to thank Shigeru Mase and Makoto Yamashita from Tokyo Institute of Technology for suggesting them of using ANOVA. Finally, they are grateful for the kindness of Takashi Tsuchiya from The Institute of Statistical Mathematics for point out the reference [15].

References

[1] E.D. Andersen and K.D. Andersen, "Presolving in linear programming," Mathematical Programming, vol. 71, pp. 221–245, 1995.

[2] C. Ashcraft, D. Pierce, D.K. Wah, and J. Wu, "The reference manual for SPOOLES, release 2.2: An object oriented software library for solving sparse linear systems of equations," Boeing Shared Services Group, Seattle, WA, January 1999. Available at http://cm.bell-labs.com/netlib/linalg/spooles.

Table 13: Numerical results on structural optimization problems for SDPA 6.00 at computer A.

	standard		conversion				conversion-fe			
problem	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg fea
buck1	0.2		36	25,24,36d	= (0.0)		67	22,16,15	0.3 (0.0)	
buck2	5.2	6	155	96,94,6	3.8 (0.2)		202	91,51,51	3.7 (0.3)	4
buck3	142.6	5 p6	774	314,201,131	153.8 (6.0)	4 p5	722	318,180,154	185.6 (6.9)	5 p4
buck4	1437.8		2580	400,369,297	3026.7 (34.6)	5 p5	2691	637,458,235	4004.2 (63.7)	4 p6
buck5	33373.8	5 p6	7614	608,530,459	68446.4 (430.4)	2 p4	5254	1043,976,789	33730.9 (732.7)	2 p4
shmup1	0.4		19	40,40,3	0.4 (0.1)		74	40,32,18	0.4 (0.0)	5
shmup2	354.8		203	440,440,3	288.2 (4.4)	5 p6	709	242,242,220	192.2 (4.5)	4 p5
shmup3	2620.9	5	885	900,480,451	1544.3 (29.1)	4	885	900,480,451	1544.1 (33.5)	4
shmup4	21598.4	5	2609	882,882,840	6155.5 (174.8)	3	1706	1680,882,840	8962.2 (216.5)	3
shmup5	M		10171	1922,1861,1860	M (1874.0)		5706	1922,1922,1861	M (2359.4)	
trto1	0.1		36	25,36d	= (0.0)		57	16,15,36d	0.1 (0.0)	6
trto2	2.1	6	155	94,6,2	2.0 (0.1)	5 p6	165	88,15,144d	1.6 (0.2)	5 p5
trto3	71.2	6 p6	652	183,147,7	59.3 (1.7)	5 p4	760	223,112,7	87.6 (3.2)	5 p4
trto4	762.7	5 p5	1542	392,293,12	798.4 (16.5)	4 p3	1539	405,284,12	764.5 (23.1)	4 p3
trto5	12036.5	4 p5	4111	905,498,406	13814.0 (230.0)	3 p4	4235	934,844,32	14864.7 (262.4)	3 p4
vibra1	0.2		36	25,24,36d	= (0.0)		67	22,16,15	0.2 (0.0)	6
vibra2	5.3		155	96,94,6	5.0 (0.2)		202	91,51,51	3.8 (0.3)	5 p5
vibra3	170.7	5 p6	774	314,201,131	183.4 (6.1)	4 p5	722	318,180,154	169.1 (6.8)	4 p5
vibra4	1596.9	5 p6	2580	400,369,297	2717.5 (35.3)	4 p3	2691	637,458,235	3436.2 (63.7)	4 p4
vibra5	32946.8	5 p5	7614	608,530,459	61118.3 (430.4)	3 p4	5254	1043,976,789	29979.9 (732.6)	3 p4

Table 14: Numerical results on structural optimization problems for SDPA 6.00 at computer B.

	standard		conversion				conversion-fe			
problem	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg fea	m_+	$n_{\max}$	time (s)	rg fea
buck1	0.1		36	25,24,36d	= (0.0)		46	25,22,6	0.1 (0.0)	
buck2	2.0		160	96,92,9	2.1 (0.1)	5	166	96,51,51	1.9 (0.2)	6
buck3	76.7	5 p6	794	314,205,125	108.2 (3.5)	4 p5	686	318,317,14	95.6 (4.4)	4 p5
buck4	714.4		2286	371,346,338	2110.9 (19.9)	6 p5	2391	669,637,30	2488.2 (30.8)	5 p6
buck5	16667.3	5 p6	6692	639,609,599	46381.0 (367.7)	3 p4	5254	1043,976,789	24194.2 (383.7)	2 p5
shmup1	0.2		19	40,40,3	0.2 (0.0)		19	40,40,3	0.2 (0.0)	
shmup2	148.3	5	203	440,440,3	115.0 (2.5)	5	456	440,242,220	102.5 (2.8)	4
shmup3	968.1	5	420	901,900,840d	= (17.4)		885	900,480,451	635.2 (18.0)	4
shmup4	5536.6	5	2609	882,882,840	3151.7 (97.5)	3	1706	1680,882,840	3407.9 (112.1)	3
shmup5	M		5706	1922,1922,1861	M (1156.9)		5706	1922,1922,1861	M (1237.3)	
trto1	0.0		36	25,36d	= (0.0)		36	25,36d	= (0.0)	
trto2	1.0	6	160	92,9,2	1.0 (0.1)	5 p5	181	91,13,2	1.2 (0.1)	4 p5
trto3	40.4	5 p6	779	313,14,12	77.0 (1.7)	3 p4	607	318,7,7	61.1 (1.7)	5 p4
trto4	424.1	5 p5	1734	401,283,12	735.6 (12.5)	4 p4	1539	405,284,12	497.9 (13.6)	3 p3
trto5	7281.3	4 p5	4117	725,588,498	9382.1 (147.6)	3 p5	4235	934,844,32	10734.3 (133.6)	3 p4
vibra1	0.1		36	25,24,36d	= (0.0)		46	25,22,6	0.1 (0.0)	
vibra2	2.5		160	96,92,9	2.6 (0.1)	6	166	96,51,51	1.8 (0.2)	5 p6
vibra3	91.2	5 p6	794	314,205,125	131.7 (3.5)	4 p5	686	318,317,14	115.2 (4.4)	4 p5
vibra4	793.4	5 p6	2286	371,346,338	2212.1 (19.5)	5 p3	2391	669,637,30	2334.1 (31.1)	4 p3
vibra5	14737.5	5 p5	6692	639,609,599	42047.7 (368.1)	2 p3	5254	1043,976,789	22015.9 (383.9)	3 p4

[3] J.R.S. Blair, and B. Peyton, "An introduction to chordal graphs and clique tress," in Graph Theory and Sparse Matrix Computations, A. George, J.R. Gilbert, J.W.H. Liu (Eds.), Springer-Verlag, pp. 1–29, 1993.

Table 15: Numerical results on structural optimization problems for SDPT3 3.02 at computer A.

	standard				conversion					conversion-fe				
problem	time (s)	rg	fea		m_+	$n_{\max}$	time (s)	rg	fea	m_+	$n_{\max}$	time (s)	rg	fea
buck1	2.5		p6		36	25,24,36d	= (0.0)			46	25,22,6	2.6 (0.0)		
buck2	8.6	6			166	96,64,38	10.3 (0.2)	6	p5	166	96,51,51	9.7 (0.3)	5	p6
buck3	123.0	4	p4		1141	190,142,140	181.1 (4.2)	3	p4	722	318,180,154	136.2 (6.9)	4	p4
buck4	780.7	5	p5		3113	282,266,220	1325.6 (26.6)	3	p3	2391	669,637,30	1423.0 (60.8)	3	p5
buck5	9634.4	3	p5		9066	450,423,392	M (381.2)			5254	1043,976,789	9558.8 (732.8)	2	p3
shmup1	3.2	6	p6		19	40,40,3	4.2 (0.0)	6		19	40,40,3	3.5 (0.1)	6	
shmup2	280.0	6	p6		709	242,242,220	161.8 (3.6)	4	p3	709	242,242,220	162.3 (4.5)	4	p3
shmup3	1444.8	5	p5		1350	480,480,451	821.8 (25.7)	5	p3	885	900,480,451	1193.7 (33.5)	5	p5
shmup4	8272.6	6	p6		3512	882,840,840	5110.2 (170.8)	6	p5	1706	1680,882,840	6726.0 (216.5)	6	p5
shmup5	73395.4	5	p5		30363	1050,1020,990	M (1888.7)			5706	1922,1922,1861	30199.9 (2358.1)	5	p4
trto1	1.2	6	p6		36	25,36d	= (0.0)			57	16,15,36d	1.3 (0.0)	5	p5
trto2	5.2	6	p6		166	64,38,2	6.0 (0.1)	4	p4	181	91,13,2	7.5 (0.2)	4	p3
trto3	53.5	5	p4		986	184,132,18	126.4 (2.0)	5	p2	607	318,7,7	97.1 (3.2)	4	p3
trto4	414.9	4	p3		1797	245,235,227	490.2 (15.0)	3	p2	1539	405,284,12	466.4 (23.1)	2	p2
trto5	4934.6	3	p4		7128	452,390,382	22572.0 (164.6)	4	p1	4235	934,844,32	6129.6 (262.4)	1	p2
vibra1	1.6	6	p6		36	25,24,36d	= (0.0)			46	25,22,6	1.5 (0.0)	6	p6
vibra2	10.5	6	p6		166	96,64,38	11.2 (0.2)	5	p5	166	96,51,51	10.9 (0.3)	5	p5
vibra3	129.1	4	p4		1141	190,142,140	216.3 (4.2)	4	p3	722	318,180,154	164.3 (6.8)	5	p4
vibra4	888.0	5	p4		3113	282,266,220	1613.7 (26.6)	0	p2	2391	669,637,30	1558.7 (60.8)	0	p2
vibra5	16327.1	4	p4		9066	450,423,392	M (381.1)			5254	1043,976,789	* (732.6)		

Table 16: Numerical results on structural optimization problems for SDPT3 3.02 at computer B.

	standard				conversion					conversion-fe				
problem	time (s)	rg	fea		m_+	$n_{\max}$	time (s)	rg	fea	m_+	$n_{\max}$	time (s)	rg	fea
buck1	1.6		p6		36	25,24,36d	= (0.0)			46	25,22,6	1.7 (0.0)		
buck2	5.4		p5		166	96,74,28	6.2 (0.1)	5	p5	166	96,51,51	6.2 (0.2)	6	p4
buck3	77.3	3	p5		968	184,145,145	91.5 (2.4)	3	p4	686	318,317,14	91.7 (4.3)	4	p3
buck4	453.6	5	p5		3358	283,245,220	1042.4 (14.8)	3	p3	2391	669,637,30	804.7 (31.3)	3	p5
buck5	5118.3	3	p5		7762	470,459,423	M (241.5)			5254	1043,976,789	5189.5 (384.0)	2	p3
shmup1	2.0	6	p6		19	40,40,3	2.6 (0.0)	6		19	40,40,3	2.2 (0.0)	6	
shmup2	164.3	6	p6		203	440,440,3	161.2 (2.5)	5	p5	456	440,242,220	143.6 (2.8)	5	p5
shmup3	771.6	5	p5		1353	480,480,450	514.6 (14.7)	5	p3	885	900,480,451	638.1 (18.0)	5	p4
shmup4	4334.4	6	p6		3512	882,840,840	2803.8 (95.6)	6	p5	1706	1680,882,840	3585.2 (112.6)	6	p5
shmup5	32807.7	2			11575	1922,1860,1054	M (1099.4)			5706	1922,1922,1861	15617.9 (1236.9)	5	p4
trto1	0.8	6	p6		36	25,36d	= (0.0)			57	16,15,36d	0.8 (0.0)	5	p5
trto2	3.1	6			166	74,28,2	4.3 (0.1)	5	p5	181	91,13,2	4.5 (0.1)	4	p3
trto3	33.5	5	p4		806	184,139,14	70.0 (1.2)	2	p2	607	318,7,7	61.0 (1.7)	4	p3
trto4	234.8	4	p3		1863	273,225,206	349.2 (9.6)	3	p2	1539	405,284,12	279.3 (13.5)	3	p2
trto5	2598.4	3	p4		7068	444,405,372	8028.3 (111.0)	4	p2	4235	934,844,32	3338.9 (134.2)	1	p2
vibra1	1.0	6	p6		36	25,24,36d	= (0.0)			46	25,22,6	0.9 (0.0)	6	p6
vibra2	6.1	6	p6		166	96,74,28	6.7 (0.1)	5	p5	166	96,51,51	6.4 (0.2)	5	p5
vibra3	80.8	4	p4		968	184,145,145	96.1 (2.5)	4	p3	686	318,317,14	109.2 (4.3)	4	p4
vibra4	508.7	5	p4		3358	283,245,220	1143.6 (15.0)	2	p3	2391	669,637,30	878.2 (31.2)	0	p2
vibra5	7174.2	2	p5		7762	470,459,423	M (239.0)			5254	1043,976,789	* (384.3)		

[4] B. Borchers, "SDPLIB 1.2, a library of semidefinite programming test problems", Optimization Methods & Software, vol. 11–12, pp. 683–690, 1999. Available at http://www.nmt.edu/~sdplib/.

[5] S. Burer, "Semidefinite programming in the space of partial positive semidefinte matrices", SIAM Journal on Optimization, vol. 14, pp. 139–172, 2003.

[6] K. Fujisawa, M. Fukuda, M. Kojima, and K. Nakata, "Numerical evaluations of SDPA (SemiDefinite Programming Algorithm)," in High Performance Optimization, H. Frenk, K. Roos, T. Terlaky, and S. Zhang (Eds.), Kluwer Academic Publishers, pp. 267–301, 2000.

[7] K. Fujisawa, M. Kojima, and K. Nakata, "Exploiting sparsity in primal-dual interior-point methods for semidefinite programming," Mathematical Programming, vol. 79, pp. 235–253, 1997.

[8] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, "Exploiting sparsity in semidefinite programming via matrix completion I: General framework," SIAM Journal on Optimization, vol. 11, pp. 647–674, 2000.

[9] K. Gatermann, and P.A. Parrilo, "Symmetry groups, semidefinite programs, and sums of squares," arXiv:math.AC/0211450, 2002.

[10] R. Grone, C.R. Johnson, E.M. Sá, H. Wolkowicz, "Positive definite completions of partial hermitian matrices," Linear Algebra and its Applications, vol. 58, pp. 109–124, 1984.

[11] G. Karypis, and V. Kumar, "METIS — A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0 —," Department of Computer Science/Army HPC Research Center, University of Minneasota, Minneapolis, MN, September 1998. Available at http://www-users.cs.umn.edu/~karypis/metis/metis.

[12] M. Kočvara and M. Stingl. Available at http://www2.am.uni-erlangen.de/~kocvara/pennon/problems.html.

[13] M. Kojima, "Sums of squares relaxations of polynomial semidefinite programs," Research Report B-397, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan, November 2003.

[14] J.B. Lasserre, "Global optimization with polynomials and the problem of moments," SIAM Journal on Optimization, vol. 11, pp. 796–817, 2001.

[15] S.L. Lauritzen, T.P. Speed, and K. Vijayan, "Decomposable graphs and hypergraphs," The Journal of the Australian Mathematical Society, Series A, vol. 36, pp. 12–29, 1984.

[16] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima, and K. Murota, "Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results," Mathematical Programming, Series B, vol. 95, pp. 303–327, 2003.

[17] K. Nakata, M. Yamashita, K. Fujisawa, and M. Kojima, "A parallel primal-dual interior-point method for semidefinite programs using positive definite matrix completion," Research Report B-398, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan, November 2003.

[18] P.A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," Mathematical Programming, vol. 96, pp. 293–320, 2003.

[19] G.S. Peace, Taguchi Methods: A Hands-on Approach, Addison-Wesley Publishing Company: Reading, Massachusetts, 1993.

[20] H. Sahai, and M.I. Ageel, The Analysis of Variance: Fixed, Random and Mixed Models, Birkhäuser: Boston, 2000.

[21] G. Taguchi, and Y. Yokoyama, Jikken Keikakuho, 2nd ed. (in Japanese), Nihon Kikaku Kyokai: Tokyo, 1987.

[22] K.-C. Toh, "Solving large scale semidefinite programs via an iterative solver on the augmented systems," SIAM Journal on Optimization, vol. 14, pp. 670–698, 2003.

[23] K.-C. Toh, M.J. Todd, and R.H. Tütüncü, "SDPT3 — A MATLAB software package for semidefinite programming, Version 1.3", Optimization Methods & Software, vol. 11–12, pp. 545-581, 1999. Available at http://www.math.nus.edu.sg/~mattohkc/sdpt3.html.

[24] J. Whittaker, Graphical Models in Applied Multivariate Statistics, John Wiley & Sons: New York, 1990.

[25] M. Yamashita, K. Fujisawa, and M. Kojima, "Implementation and evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0)", Optimization Methods & Software, vol. 18, pp. 491–505, 2003.

[26] M. Yamashita, K. Fujisawa, and M. Kojima, "SDPARA : SemiDefiniteProgramming Algorithm paRAllel Version", Parallel Computing, vol. 29, pp. 1053–1067, 2003.

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