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, revised July 2004

Abstract

Considering that preprocessing is an important phase in linear programming, it should be more systematically incorporated in semidefinite programming solvers. The conversion method proposed by the authors (*SIAM J. Optim.*, **11**, 647–674 (2000), and *Math. Program. (Series B)*, **95**, 303–327 (2003)) is a preprocessing method for 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 certain classes of very sparse semidefinite programs.

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

1 Introduction

Recently, Semidefinite Programming (SDP) has gained attention in several new fronts such as 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 are characterized as large-scale and most of the time it is challenging even to load the problem data in the physical memory of the computer. As a practical compromise, we often restrict ourselves to solve sparse instances of these large-scale SDPs.

Motivated by the need to solve such challenging SDPs, this article explores further the preprocessing procedure named the *conversion method* and 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 matrix completion theory. If the SDP data matrices are very sparse and the matrix sizes are large, the conversion method produces an SDP which can be solved faster and requires less memory than the original SDP when solved by a primal-dual interior-point method [16].

The conversion method is a first step towards a general preprocessing phase for sparse SDPs as 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 these procedures can be used for sparse and even for dense SDPs.

We believe that further investigations are necessary to propose efficient preprocessing to solve large-scale 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 the 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 the 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 $i \notin C_r$, and $\mathcal{S}_+^{C_r}$ denotes the set of positive semidefinite symmetric matrices with elements specified in $C_r \times C_r$. We can assume without loss of generality that $C_1, C_2, \ldots, C_\ell$ are 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 $\mathcal{E}$ 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

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

and appropriately rearrange all data matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$, and the matrices corresponding to the equalities $X_{ij}^r = X_{ij}^s$ 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 by predicting *a priori* the number of flops required to solve it by a 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 an attempt to refine it, and a flop estimation function is introduced for this purpose. The number of flops needed to compute the Schur Complement Matrix (SCM) [6] and perform other computations such as factorization of the SCM, solving triangular systems, and computing eigenvalues can be roughly estimated as a function of equality constraints m, matrix sizes n_r'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 to solutions of 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 step towards detecting SDPs which are suitable for the conversion method. We can conclude that preprocessing by the conversion method becomes more advantageous when the SDPs are sparse. In particular, it seems that sparse SDPs which have less than 5% on the extended sparsity pattern (see Section 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 to 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 as long 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 organized as follows. Section 2 reviews 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, reviews 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 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 the Introduction holds if and only if the associated graph $G(V, F^\circ)$ of F given in (i) is chordal [8, 10]. We further observe that remarkably the same fact was proved independently in graphical models in statistics [15] 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 make $G(V, F^\circ)$ 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. However, it is known that the *multiset of separators*, *i.e.*, $\{C_r \cap C_s : (C_r, C_s) \in \mathcal{E}\}$, 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 all $(C_r, C) \in \mathcal{E}$ or $(C_s, C) \in \mathcal{E}$ 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 the 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 all $(C_r,C)\in\mathcal{E}$ or $(C_s,C)\in\mathcal{E}$ 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 all $(C_r,C)\in\mathcal{E}$, or $(C_s,C)\in\mathcal{E}$, or $(C_q,C)\in\mathcal{E}$ 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 the 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) $m'_+=m_+-g(C_r\cap C_q)-g(C_s\cap C_q)$.

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 less fill-in in the aggregate sparsity matrix $\boldsymbol{A}(E)$ (e.g., Spooles 2.2 [2] and METIS 4.0.1 [11]).*

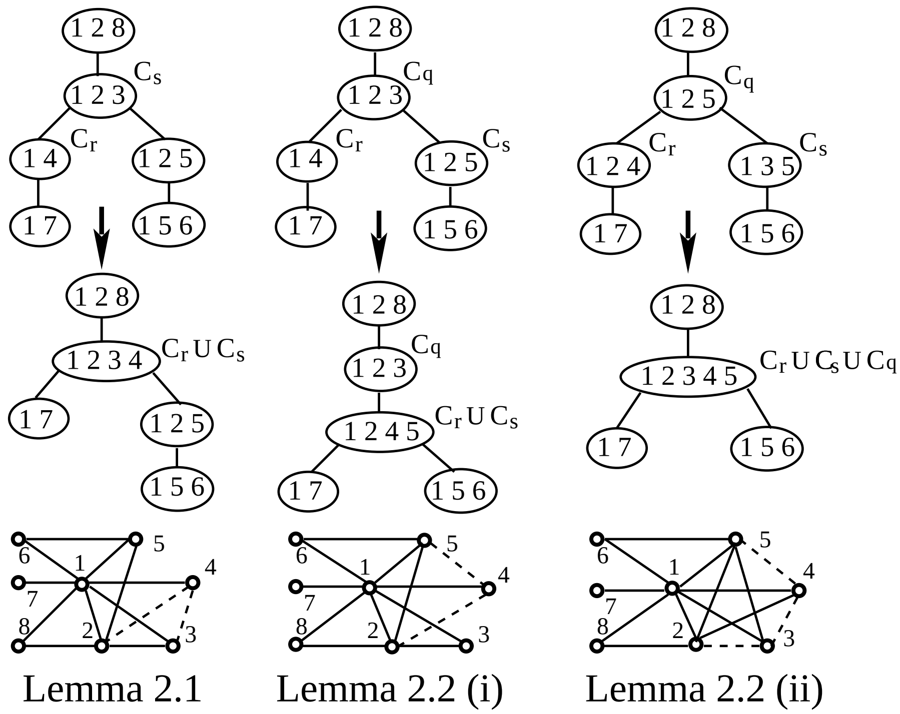

Figure 1: Illustration of Lemmas 2.1 and 2.2. Clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ before the merging (above), clique tree $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ after the merging (middle), and the associate chordal graph (bottom). Dotted lines denote the edges added to the graph due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

Step 3. *From the ordering above, perform a symbolic Cholesky factorization on $\boldsymbol{A}(E)$ 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 reduce 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 considerations in the conversion method is to obtain a suitable chordal extension $G(V, F^\circ)$ of $G(V, E^\circ)$ which allows 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 possibly gives 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 SDP completely equivalent to the original one, with smaller block matrices, but with a 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 the Introduction, we finally obtain 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 a chordal extension $G(V, F^\circ)$ that gives as little 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 crucially, a large number of equality constraints $m_+ \gg m$. One of the keys to obtaining a good conversion is to balance the factors (a') and (b') above to minimize the number of flops required by an SDP solver. Therefore, there is a necessity to use at Step 5 a heuristic procedure that directly manipulates 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 beyond reach 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 consider Algorithm 3.1 to be 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 give 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, this 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 to determine whether two maximal cliques C_r and C_s satisfying the hypothesis of Lemmas 2.1 or 2.2 should be merged so as 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 perfect, 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 clique which share a reasonable number of indices when compared with the smaller one. In particular, this criterion ignores the smaller of $|C_r|$ 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 seem 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, the best choice for the parameters given above 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 required by an SDP solver.

4.1 Estimating Flops Required by SDP Codes

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

The actual flop counts 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 the number of flops per iteration required by a primal-dual interior-point method. The main cost is computing the SCM. Other requirements are: $\mathcal{O}(m^3)$ flops for the Cholesky factorization to solve the linear system of the SCM; $\mathcal{O}(\sum_{r=1}^{\ell} n_r^3)$ flops for the multiplication of matrices of size $n_r \times n_r$ and $\mathcal{O}(m \sum_{r=1}^{\ell} n_r^2)$ flops to evaluate m inner-products between matrices of size $n_r \times n_r$ to determine the search direction; and finally $\mathcal{O}(\sum_{r=1}^{\ell} n_r^3)$ flops to compute the minimum eigenvalues of matrices of size $n_r \times n_r$ to determine the step length, in the case when all data matrices are dense. See [6] for details.

In particular, SDPA 6.00 considers the sparsity of the data matrices $\boldsymbol{A}_p$ $(p = 1, 2, \ldots, m)$, and employs the formula $\mathcal{F}_*$ [7] to compute the SCM. We assume henceforth that all data matrices $\boldsymbol{A}_p$ $(p = 0, 1, \ldots, m)$ have the same block-diagonal matrix structure consisting of ℓ block matrices with dimensions $n_r \times n_r$ $(r = 1, 2, \ldots, \ell)$ each.

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 index r. Analogously, $f_\Sigma(r)$

denotes the number of nonzero elements of $\boldsymbol{A}(E)$ for the corresponding block matrix. In the following discussion of flop estimates, there will be no loss of generality in assuming 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 of computing 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 introduce the term $n_r f_\Sigma(r)$ for each $r = 1, 2, \ldots, \ell$. In particular, $f_\Sigma(r)$ becomes equal to n_r^2 if the corresponding block matrix is dense in $\boldsymbol{A}(E)$.

We propose the following formula for the 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 reasons to be explained next.

Our goal is to replace criterion (4) which determines whether 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). Consider the most complicated case in 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, and therefore cause a substantial overhead.

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 recalculating the non-zero 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 in 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) when a decision to merge the cliques or not is almost clear from the clique tree.

As in Algorithm 3.2, it remains a critical question as to 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 believe 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 considered 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 parameters". Nevertheless, we propose the following way to determine the possibly best 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 we use a technique which combines the *analysis of variance* (ANOVA) [20] and the orthogonal arrays [19] described in [21], and 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 of computing them. Therefore, repeating ANOVA for different sets of parameter values, we can hopefully obtain the best parameters for our benchmark problems. In addition, the orthogonal arrays allow us to avoid making experiments with all possible combinations of parameter values. Details of the method are beyond the scope of this paper and are therefore omitted.

We conducted our experiments on two different computers to verify the sensitivity 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). 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.

We obtained the following parameters given in Table 1 for SDPA 6.00 [25] and SDPT3 3.02 [23].

Table 1: Parameters for Algorithms 3.2 and 4.1 on 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 Mehrotra-type 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 (see also [7]) to compute the SCM on which our newly proposed conversion flop estimation version partially relies.

Three different sets 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 parameter κ in (5) was fixed to 2.2. The parameters $\zeta_{\min}$ and $\zeta_{\max}$ in Algorithm 4.1 were empirically fixed to 0.035 and 0.98, respectively, using the benchmark problems in Subsection 5.1.

In the tables that follows, the original problem sizes are given by the number of equality constraints m, the number of rows of each block matrix n (where "d" after a number denotes a diagonal matrix), and the sparsity of problems which can be partially understood from the percentages 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 the 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 ("fe" stands for "flop estimate"). The numbers between parentheses are the time for the "conversion" and "conversion-fe". 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 the three largest block matrices after the respective conversion. Bold font numbers indicate the best timing and the ones which are at most 110% of the best timing among "standard", "conversion", and "conversion-fe" including the time for the conversion themselves. In this comparison of time, we ignored the final relative duality gaps and feasibility errors (defined next) specially in the Tables 9 and 10 on structural optimization problems.

We utilized the default parameters both for SDPA 6.00 and SDPT3 3.02 except that λ^* was occasionally changed for SDPA 6.00, and OPTIONS.gaptol was 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",

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

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

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

respectively. To save space, the negative $\log_{10}$ values of these quantities are reported. 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 in the tables.

Finally, the numerical results for SDPA 6.00 and SDPT3 3.02 show very similar behaviors. Therefore, to keep the essence of the comparison and avoid lengthy tables, we decided to not include the numerical results for SDPT3 3.02, and instead just point out the relevant differences in each of the corresponding subsections.

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 in 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 and percentages of the aggregate and extended sparsity patterns of norm minimization problems, SDP relaxation of quadratic programs with box constraints, SDP relaxation of 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 on computers 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" and "conversion-fe" are better than "standard". However, for the SDP relaxation of quadratic programs, no conversion is ideal. This result is particularly intriguing since the superiority of the "conversion" was clear when using SDPA 5.0 [16], and

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

problem	standard time (s)	conversion m_+	conversion $n_{\max}$	conversion time (s)	conversion-fe m_+	conversion-fe $n_{\max}$	conversion-fe 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)

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 on computers A and B, they have similar trends except that it is faster to solve the norm minimization problems on computer A than computer B due to its cache size.

Similar results were observed for SDPT3 3.02 on computers A and B. Details are not shown here. However, for SDPT3 3.02, "conversion" or "conversion-fe" is sometimes better than "standard" on the SDP relaxation of quadratic programs. Also, the computational time for the norm minimization problems is not faster on computer A than on computer B as was observed for SDPA 6.00.

We observe that for "conversion-fe", all of the converted problems in the corresponding tables are the same for computers A and B, and for both SDPA 6.00 and SDPT3 3.02,

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

problem	standard time (s)	conversion m_+	conversion $n_{\max}$	conversion time (s)	conversion-fe m_+	conversion-fe $n_{\max}$	conversion-fe 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)

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 about 140 times for “norm1” using SDPA 6.00, and about 14 times for SDPT3 3.02, when compared with “standard”, even including the time for the conversion itself. And in the worse case “qp4.0” the running time is only 2.4 times more than “standard”.

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 sparsity patterns including the ones after the congruent transformation [8, Section 6] like “equalG”, “gpp”, and “theta” problems. We excluded the small instances because we are interested in large-scale SDPs, and also the large ones because of the insufficiency of memory. Problem sizes and sparsity information are shown in Table 5. Observe that in several cases, the fill-in effect causes the extended sparsity patterns to become *much denser* than the corresponding aggregate sparsity patterns.

Table 5: Sizes and percentages of the aggregate and extended sparsity patterns of SDPLIB problems.

problem	m	n	aggregate (%)	extended (%)
equalG11	801	801	1.24	(A) 4.32, (B) 4.40
equalG51	1001	1001	4.59	(A) 52.99, (B) 53.37
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
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
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.

We can observe from the numerical results in Tables 6 and 7 that the conversion method is advantageous when the extended sparsity patterns are less than 5%, which

Table 6: Numerical results on SDPLIB problems for SDPA 6.00 on computer A.

problem	standard time (s)	standard rg	standard fea	conversion m_+	conversion $n_{\max}$	conversion time (s)	conversion rg	conversion fea	conversion-fe m_+	conversion-fe $n_{\max}$	conversion-fe time (s)	conversion-fe rg	conversion-fe fea
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)		
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)		
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
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**)	6		1949	200	= (**6.7**)	6	
theta5	**581.0**	6		3028	250	= (**15.0**)	6		3028	250	= (**15.7**)	6	
theta6	**1552.9**	6		4375	300	= (**31.1**)	6		4375	300	= (**32.2**)	6	
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**)	3	p5

is the case for "equalG11", "maxG11", "maxG32", "mcp250-1", "mcp500-1", "qpG11", "qpG51", and "thetaG11". The exception are "maxG32" and "mcp250-1" for SDPT3 3.02. 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 "qpG51".

Once again, the converted problems under the columns "conversion-fe" are exactly the same for the corresponding tables except "equalG11", "maxG11", "mcp250-4", "mcp500-1", "qpG11", and "ss30".

For "qpG11" we have a speed-up of 6.4 to 13.2 times when preprocessed by "conversion" or "conversion-fe" using SDPA 6.00, and 15.5 to 19.3 times for SDPT3 3.02,

Table 7: Numerical results on SDPLIB problems for SDPA 6.00 on computer B.

problem	standard time (s)	rg	fea	conversion m_+	$n_{\max}$	time (s)	rg	conversion-fe m_+	$n_{\max}$	time (s)	rg	fea
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	
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)		
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**)	p6	132	294,132d	= (**0.8**)		p6
theta2	**3.4**			498	100	= (**0.3**)		498	100	= (0.4)		
theta3	**22.5**	6		1106	150	= (**1.5**)	6	1106	150	= (**1.6**)	6	
theta4	**91.0**	6		1949	200	= (**4.6**)	6	1949	200	= (**4.9**)	6	
theta5	**264.9**	6		3028	250	= (**11.6**)	6	3028	250	= (**11.6**)	6	
theta6	**659.9**	6		4375	300	= (**24.4**)	6	4375	300	= (**25.1**)	6	
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**)	3	p5

even including the time for the conversion itself. On the other hand, the worse case is "mcp250-1" (for SDPT3 3.02) which takes only 1.6 times more than "standard" when restricting to problems with less than 5% on their extended sparsity patterns.

5.3 Structural Optimization Problems

The last set of problems are from structural optimization [12] and have sparse aggregate sparsity patterns. Problem sizes and sparsity information are shown in Table 8.

The numerical results for these four classes of problems for SDPA 6.00 on computers A and B are shown in Tables 9 and 10. Entries with "M" means out of memory.

Among these four classes, the conversion method is only advantageous for the "shmup" problems.

Table 8: Sizes and percentages of the aggregate and extended sparsity patterns of structural optimization problems.

problem	m	n	aggregate (%)	extended (%)
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
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
trto3	544	321,544d	4.22	7.50
trto4	1200	673,1200d	2.12	5.52
trto5	3280	1761,3280d	0.85	3.01
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

Table 9: Numerical results on structural optimization problems for SDPA 6.00 on 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
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
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)		
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
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

We observe that both SDPA 6.00 and SDPT3 3.02 have some difficult solving the converted problems, *i.e.*, "conversion" and "conversion-fe", to the same accuracy as "standard", suggesting that the conversion itself can sometimes cause a negative effect. In some cases like "vibra5" for "conversion-fe", SDPT3 3.02 fails to solve them. However, these structural optimization problems are difficult to solve by their nature (see "rg" and "fea" columns under "standard").

Table 10: Numerical results on structural optimization problems for SDPA 6.00 on 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
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
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)	5		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)		
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
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

Once again, the converted problems under the columns "conversion-fe" have similar sizes. In particular the converted problems are exactly the same for the corresponding tables for "buck5", "shmup3$\sim$5", "trto4$\sim$5" and "vibra5".

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 actually increases the computational cost.

In particular, SDPT3 3.02 fails to solve "buck5" and "vibra5" for "conversion" due to lack of memory while it can solve "shmup5" using "conversion-fe", which SDPA 6.00 cannot solve, again because of insufficient memory.

6 Conclusion and Further Remarks

As we stated in the 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 reducing 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 was 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 8 to 147 times faster, and “qpG11” became 6 to 19 times 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 obtaining the feasibility is difficult, the computational time takes at most 4 times more than solving without any conversion.

In practice, it is really worthwhile preprocessing by “conversion” or “conversion-fe” which has the potential to reduce the computational time by a factor of 10 to 100 for sparse SDPs. Even in those cases where solving the original problem is faster, the preprocessed SDPs take at most twice as long to solve.

Generally, when computational time is substantially reduced, so is memory utilization [16], although we did not present details on this.

We have a general impression that the performance of “conversion-fe” is better than “conversion” in the worst-case scenarios, when solving the original problem is slightly faster, for all the numerical experiments we completed. A minor remark is that “conversion-fe” produces in general similar SDPs in terms of sizes independent of computers and solvers which indicates that “conversion-fe” relies more on how we define the flop estimation function.

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

As proposed in the Introduction, the conversion method should be considered as a first step to for preprocessing in SDP solvers. An immediate project is to consider incorporating the conversion method in SDPA [25] and SDPARA [26] together with the completion method [8, 16, 17], and to further develop theoretical and practical algorithms to exploit 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 from the Institute for Mathematics and its Applications and Tokyo Institute of Technology for inviting him as a visiting researcher at their respective institutes, making this research possible. In particular, Masakazu Kojima contributed many valuable comments on this paper. The authors would like to thank Shigeru Mase from Tokyo Institute of Technology and Makoto Yamashita from Kanagawa University for suggesting the use of ANOVA. They are grateful to Takashi Tsuchiya from The Institute of Statistical Mathematics for pointing out the reference [15]. Finally, they thank to the two anonymous referees and Michael L. Overton of New York University for improving the clarity of the paper.

References

[1] E.D. Andersen and K.D. Andersen (1995). Presolving in linear programming. *Math. Program.*, **71**, 221–245.

[2] C. Ashcraft, D. Pierce, D.K. Wah, and J. Wu (1999). *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. [http://cm.bell-labs.com/netlib/linalg/spooles].

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

[4] B. Borchers (1999). SDPLIB 1.2, a library of semidefinite programming test problems. *Optim. Methods Softw.*, **11–12**, 683–690. [http://www.nmt.edu/~sdplib/].

[5] S. Burer (2003). Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM J. Optim.*, **14**, 139–172.

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

[7] K. Fujisawa, M. Kojima, and K. Nakata (1997). Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Math. Program.*, **79**, 235–253.

[8] M. Fukuda, M. Kojima, K. Murota, and K. Nakata (2000). Exploiting sparsity in semidefinite programming via matrix completion I: General framework. *SIAM J. Optim.*, **11**, 647–674.

[9] K. Gatermann, and P.A. Parrilo (2004). Symmetry groups, semidefinite programs, and sums of squares. *J. Pure Appl. Algebra*, **192**, 95–128.

[10] R. Grone, C.R. Johnson, E.M. Sá, and H. Wolkowicz (1984). Positive definite completions of partial hermitian matrices. *Linear Algebra Appl.*, **58**, 109–124.

[11] G. Karypis, and V. Kumar (1998). *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 Minnesota, Minneapolis, MN. [http://www-users.cs.umn.edu/~karypis/metis/metis].

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

[13] M. Kojima (2003). *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.

[14] J.B. Lasserre (2001). Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, **11**, 796–817.

[15] S.L. Lauritzen, T.P. Speed, and K. Vijayan (1984). Decomposable graphs and hypergraphs. *J. Aust. Math. Soc. (Series A)*, **36**, 12–29.

[16] 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. *Math. Program. (Series B)*, **95**, 303–327.

[17] K. Nakata, M. Yamashita, K. Fujisawa, and M. Kojima (2003). *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.

[18] P.A. Parrilo (2003). Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, **96**, 293–320.

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

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

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

[22] K.-C. Toh (2003). Solving large scale semidefinite programs via an iterative solver on the augmented systems. *SIAM J. Optim.*, **14**, 670–698.

[23] K.-C. Toh, M.J. Todd, and R.H. Tütüncü (1999). SDPT3 — A MATLAB software package for semidefinite programming, Version 1.3. *Optim. Methods Softw.*, **11–12**, 545-581. [http://www.math.nus.edu.sg/˜mattohkc/sdpt3.html].

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

[25] M. Yamashita, K. Fujisawa, and M. Kojima (2003). Implementation and evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0). *Optim. Methods Softw.*, **18**, 491–505.

[26] M. Yamashita, K. Fujisawa, and M. Kojima (2003). SDPARA : SemiDefinite Programming Algorithm paRAllel Version. *Parallel Comput.*, **29**, 1053–1067.

[27] Z. Zhao, B.J. Braams, M. Fukuda, M.L. Overton, and J.K. Percus (2004). The reduced density matrix method for electronic structure calculations and the role of three-index representability conditions. *J. Chem. Phys.*, **120**, 2095–2104.