

A Conversion of an SDP Having Free Variables into the Standard Form SDP

Kazuhiro Kobayashi^{*}, Kazuhide Nakata[†] and Masakazu Kojima[‡]

June 2005. Revised April 2006

Abstract. This paper deals with a semidefinite program (SDP) having free variables, which often appears in practice. To apply the primal-dual interior-point method, we usually need to convert our SDP into the standard form having no free variables. One simple way of conversion is to represent each free variable as a difference of two nonnegative variables. But this conversion not only expands the size of the SDP to be solved but also yields some numerical difficulties which are caused by the non-existence of a primal-dual pair of interior-feasible solutions in the resulting standard form SDP and its dual. This paper proposes a new conversion method that eliminates all free variables. The resulting standard form SDP is smaller in its size, and it can be more stably solved in general because the SDP and its dual have interior-feasible solutions whenever the original primal-dual pair of SDPs have interior-feasible solutions. Effectiveness of the new conversion method applied to SDPs having free variables is reported in comparison to some other existing methods.

Key words.

Semidefinite Program, Primal-Dual Interior-Point Method, Equality Constraint, Standard Form, Conversion.

^{*} Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. kazuhir2@is.titech.ac.jp

[†] Department of Industrial Engineering and Management, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. knakata@me.titech.ac.jp

[‡] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. kojima@is.titech.ac.jp

1 Introduction

Throughout the paper, we use the notation \mathbb{R} for the set of real numbers, \mathbb{R}^p for the p -dimensional Euclidean space, \mathbb{S}^n for the linear space of $n \times n$ real symmetric matrices and \mathbb{S}_+^n for the cone of positive semidefinite matrices in \mathbb{S}^n , respectively. For a pair of matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{S}^n$, the inner product is defined as $\mathbf{X} \bullet \mathbf{Y} = \sum_{i=1}^n \sum_{j=1}^n X_{ij} Y_{ij}$. Let $\mathbf{A}_i \in \mathbb{S}^n$ ($i = 1, 2, \dots, m$). We define the linear operator $\mathcal{A} : \mathbb{S}^n \rightarrow \mathbb{R}^m$ by $\mathcal{A}\mathbf{X} = (\mathbf{A}_1 \bullet \mathbf{X}, \mathbf{A}_2 \bullet \mathbf{X}, \dots, \mathbf{A}_m \bullet \mathbf{X})^T$ for every $\mathbf{X} \in \mathbb{S}^n$, and the adjoint linear operator $\mathcal{A}^* : \mathbb{R}^m \rightarrow \mathbb{S}^n$ by $\mathcal{A}^*\mathbf{y} = \sum_{i=1}^m \mathbf{A}_i y_i$ for every $\mathbf{y} \in \mathbb{R}^m$.

When we study the theory of SDPs and their computational methods, we usually deal with a standard form semidefinite program (SDP)

$$\begin{aligned} \text{(P0)} \quad & \text{minimize} \quad \mathbf{C} \bullet \mathbf{X} \\ & \text{subject to} \quad \mathcal{A}\mathbf{X} = \mathbf{b} \quad \text{and} \quad \mathbf{X} \in \mathbb{S}_+^n. \end{aligned}$$

Here $\mathbf{C} \in \mathbb{S}^n$ and $\mathbf{b} \in \mathbb{R}^m$ are given problem parameters and $\mathbf{X} \in \mathbb{S}_+^n$ is the optimization variable. In fact, the duality theory on SDPs as well as the primal-dual interior-point methods for SDPs have been presented for this standard form SDP and its dual in many articles [9, 10, 12, 16]. In applications, however, various different forms of SDPs are formulated.

One typical SDP which is different from the standard form involves free variables:

$$\begin{aligned} \text{(P1)} \quad & \text{minimize} \quad \mathbf{C} \bullet \mathbf{X} + \mathbf{f}^T \mathbf{z} \\ & \text{subject to} \quad \mathcal{A}\mathbf{X} + \mathbf{D}\mathbf{z} = \mathbf{b} \quad \text{and} \quad \mathbf{X} \in \mathbb{S}_+^n. \end{aligned}$$

Here $\mathbf{D} \in \mathbb{R}^{m \times p}$ is a constant matrix, $\mathbf{f} \in \mathbb{R}^p$ is a constant vector and $\mathbf{z} \in \mathbb{R}^p$ is a variable. Many practical problems are formulated as SDPs of this type, which include electronic structure calculation in chemical physics [6, 15] and SDP relaxations of polynomial optimization problems [11, 19]. To apply a primal-dual interior-point method [9, 10, 12, 16] to the SDP (P1), we need to convert it into the standard form SDP (P0). One simple way of conversion is to represent the free variable vector $\mathbf{z} \in \mathbb{R}^p$ as a difference of two nonnegative variable vectors such as $\mathbf{z} = \mathbf{z}_+ - \mathbf{z}_-$, $\mathbf{z}_+ \in \mathbb{R}_+^p$ and $\mathbf{z}_- \in \mathbb{R}_+^p$, where \mathbb{R}_+^p denotes the nonnegative orthant of \mathbb{R}^p . This conversion method is employed in the software package SeDuMi [17] and SDPT3 [18]. The method not only expands the size of SDP to be solved but also yields some numerical difficulties such that the converted SDP has a continuum of optimal solutions and its dual has no interior feasible solution. This demerit would often cause it difficult to solve the converted SDP stably and/or to compute a highly accurate optimal solution as reported in the paper [19]. We can also modify the primal-dual interior-point method so as to process an SDP having free variables as done in an old version of the software package SDPT3 [18]. Solving an SDP having free variables stably and accurately, however, is still an important research subject.

This paper presents a new method for converting the SDP (P1) into a standard form SDP. In order to compute an accurate optimal solution of the converted standard form SDP by a general-purpose interior-point method, it is desirable that the conversion preserves the strict feasibility. More specifically, it is desirable that the converted standard form SDP and its dual have interior feasible solutions whenever this property holds for the original primal-dual pair of SDPs. Our conversion method satisfies this property. Another important

feature of our method is that as the original SDP (P1) involves more free variables, the size of the converted standard form SDP becomes smaller than that of the original SDP (P1); hence solving the converted SDP is faster than solving the original SDP (P1) if their data matrices have similar sparsity. An important issue in our conversion method is how we maintain the sparsity [5, 7, 13, 14] of the original SDP (P1). When the original SDP (P1) is fully dense, the converted SDP is also fully dense. Hence there is no need to consider the sparsity. When the original SDP (P1) is sparse, however, the converted SDP may become denser than the original SDP (P1). Due to this worsening of the sparsity, it may be slower to solve the converted SDP than the original SDP (P1) although the size of the converted SDP is smaller than that of the original SDP (P1). Therefore we need to consider how to maintain the sparsity of the original SDP (P1) in the converted SDP.

In Section 2, we describe two methods that convert the SDP (P1) having free variables into standard form SDPs. The first one is a simple conversion method which rewrites a free variable as the difference of two non-negative variables, while the second method is our new conversion method. Section 3 is devoted to some technical details of the new conversion method including its number of arithmetic operations required and a greedy heuristic technique to make the converted standard form SDP as sparse as possible. In Section 4, we report some numerical results.

2 Conversion of an SDP having free variables into a standard form SDP

In this section, we present two methods for converting the SDP (P1) having free variables into standard form SDPs.

2.1 A simple conversion by splitting free variables

The SDP (P1) can be converted into a standard form SDP by representing the variable $\mathbf{z} \in \mathbb{R}^p$ as a difference of two nonnegative variable vectors $\mathbf{z}_+ \in \mathbb{R}_+^p$ and $\mathbf{z}_- \in \mathbb{R}_+^p$ such that $\mathbf{z} = \mathbf{z}_+ - \mathbf{z}_-$. Let $\text{diag}(\mathbf{a})$ denote the $p \times p$ diagonal matrix with the diagonal components a_1, a_2, \dots, a_p for each $\mathbf{a} = (a_1, a_2, \dots, a_p)^T \in \mathbb{R}^p$, and let \mathbf{d}_i ($i = 1, 2, \dots, m$) denote the i th row of the matrix \mathbf{D} . Then the SDP (P1) can be rewritten as a standard form SDP

$$(P2) \quad \begin{aligned} & \text{minimize} && \widehat{\mathbf{C}} \bullet \widehat{\mathbf{X}} \\ & \text{subject to} && \widehat{\mathbf{A}}\widehat{\mathbf{X}} = \mathbf{b} \quad \text{and} \quad \widehat{\mathbf{X}} \in \mathbb{S}_+^{n+2p}. \end{aligned}$$

where

$$\begin{aligned} \widehat{\mathbf{C}} &= \begin{pmatrix} \mathbf{C} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \text{diag}(\mathbf{f}) & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & -\text{diag}(\mathbf{f}) \end{pmatrix}, \\ \widehat{\mathbf{A}}_i &= \begin{pmatrix} \mathbf{A}_i & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \text{diag}(\mathbf{d}_i^T) & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & -\text{diag}(\mathbf{d}_i^T) \end{pmatrix} \quad (i = 1, 2, \dots, m), \end{aligned}$$

$$\widehat{\mathbf{X}} = \begin{pmatrix} \mathbf{X} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \text{diag}(\mathbf{z}_+) & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \text{diag}(\mathbf{z}_-) \end{pmatrix}.$$

As in the case of \mathcal{A} , $\widehat{\mathcal{A}}$ is a linear operator defined as $\widehat{\mathcal{A}}\widehat{\mathbf{X}} = (\widehat{\mathbf{A}}_1 \bullet \widehat{\mathbf{X}}, \widehat{\mathbf{A}}_2 \bullet \widehat{\mathbf{X}}, \dots, \widehat{\mathbf{A}}_m \bullet \widehat{\mathbf{X}})^T$. As we mentioned in the Introduction, disadvantages of this method are (a) the size of (P2) becomes bigger, (b) the set of optimal solutions of the SDP (P2) is unbounded if it is nonempty and (c) the dual of the SDP (P2) has no interior feasible solution.

2.2 The new conversion method

We start with the dual of (P1)

$$\begin{aligned} \text{(D1)} \quad & \text{maximize} && \mathbf{b}^T \mathbf{y} \\ & \text{subject to} && \mathbf{C} - \mathcal{A}^* \mathbf{y} = \mathbf{Z}, \mathbf{Z} \in \mathbb{S}_+^n \text{ and } \mathbf{D}^T \mathbf{y} - \mathbf{f} = \mathbf{0}. \end{aligned}$$

Without loss of generality, we assume that the $m \times p$ matrix \mathbf{D} is column full rank; hence $\text{rank}(\mathbf{D}) = p$. When \mathbf{D} is not column full rank, we remove redundant columns from \mathbf{D} so that the resulting matrix \mathbf{D}' has a smaller size and is column full rank. In this case, we use \mathbf{D}' instead of \mathbf{D} .

Since $\text{rank}(\mathbf{D}) = p$, we know that $p \leq m$ and that \mathbf{d}_i^T ($i = 1, 2, \dots, m$) contain p column vectors forming a basis of \mathbb{R}^p . Let B denote the set of indices of the column vectors in the basis and N the set of indices of the other column vectors. For simplicity of discussions, we may assume that $B = \{1, 2, \dots, p\}$ and $N = \{p+1, p+2, \dots, m\}$. We use the notation \mathbf{D}_B for the submatrix of \mathbf{D} consisting of \mathbf{d}_i ($i \in B$) and the notation \mathbf{D}_N for the submatrix of \mathbf{D} consisting of \mathbf{d}_i ($i \in N$). Similarly, we use \mathbf{y}_B , \mathbf{y}_N , \mathbf{b}_B and \mathbf{b}_N for subvectors of $\mathbf{y} \in \mathbb{R}^m$ and $\mathbf{b} \in \mathbb{R}^m$ with the basis index set B and the nonbasis index set N , respectively. Note that \mathbf{D}_B is nonsingular. Hence we can solve the equality constraint $\mathbf{D}^T \mathbf{y} - \mathbf{f} = \mathbf{0}$ in \mathbf{y}_B such that

$$\mathbf{y}_B = \mathbf{D}_B^{-T} \mathbf{f} - \mathbf{D}_B^{-T} \mathbf{D}_N^T \mathbf{y}_N.$$

Define the linear operators $\mathcal{A}_B^* : \mathbb{R}^p \rightarrow \mathbb{S}^n$ and $\mathcal{A}_N^* : \mathbb{R}^{m-p} \rightarrow \mathbb{S}^n$ by

$$\mathcal{A}_B^* \mathbf{y}_B = \sum_{i \in B} \mathbf{A}_i y_i \text{ and } \mathcal{A}_N^* \mathbf{y}_N = \sum_{i \in N} \mathbf{A}_i y_i,$$

respectively. Now, substituting $\mathbf{y}_B = \mathbf{D}_B^{-T} \mathbf{f} - \mathbf{D}_B^{-T} \mathbf{D}_N^T \mathbf{y}_N$ into the objective function and the linear matrix inequality constraint of the SDP (D1), we have

$$\begin{aligned} \mathbf{b}^T \mathbf{y} &= \mathbf{b}_B^T \mathbf{y}_B + \mathbf{b}_N^T \mathbf{y}_N \\ &= \mathbf{b}_B^T (\mathbf{D}_B^{-T} \mathbf{f} - \mathbf{D}_B^{-T} \mathbf{D}_N^T \mathbf{y}_N) + \mathbf{b}_N^T \mathbf{y}_N \\ &= \mathbf{b}_B^T \mathbf{D}_B^{-T} \mathbf{f} + (\mathbf{b}_N^T - \mathbf{b}_B^T \mathbf{D}_B^{-T} \mathbf{D}_N^T) \mathbf{y}_N \\ \mathbb{S}_+^n \ni \mathbf{Z} &= \mathbf{C} - \mathcal{A}_B^* \mathbf{y}_B - \mathcal{A}_N^* \mathbf{y}_N \\ &= \mathbf{C} - \mathcal{A}_B^* (\mathbf{D}_B^{-T} \mathbf{f} - \mathbf{D}_B^{-T} \mathbf{D}_N^T \mathbf{y}_N) - \mathcal{A}_N^* \mathbf{y}_N \\ &= (\mathbf{C} - \mathcal{A}_B^* \mathbf{D}_B^{-T} \mathbf{f}) - \sum_{i \in N} (\mathbf{A}_i - \mathcal{A}_B^* (\mathbf{D}_B^{-T} \mathbf{d}_i^T)) y_i. \end{aligned}$$

Thus we obtain the following primal-dual SDPs which are equivalent to the primal-dual pair of SDPs (P1) and (D1):

$$\begin{aligned}
\text{(P3)} \quad & \text{minimize} && \tilde{b}_0 + \tilde{\mathbf{C}} \bullet \mathbf{X} \\
& \text{subject to} && \tilde{\mathbf{A}}\mathbf{X} = \tilde{\mathbf{b}} \text{ and } \mathbf{X} \in \mathbb{S}_+^n. \\
\text{(D3)} \quad & \text{maximize} && \tilde{b}_0 + \tilde{\mathbf{b}}^T \mathbf{y}_N \\
& \text{subject to} && \tilde{\mathbf{C}} - \tilde{\mathbf{A}}^* \mathbf{y}_N = \mathbf{Z} \text{ and } \mathbf{Z} \in \mathbb{S}_+^n.
\end{aligned}$$

where

$$\begin{aligned}
\tilde{b}_0 &= \mathbf{b}_B^T \mathbf{D}_B^{-T} \mathbf{f}, \quad \tilde{\mathbf{b}} = (\mathbf{b}_N^T - \mathbf{b}_B^T \mathbf{D}_B^{-T} \mathbf{D}_N^T)^T, \quad \tilde{\mathbf{C}} = \mathbf{C} - \mathbf{A}_B^* \mathbf{D}_B^{-T} \mathbf{f}, \\
\tilde{\mathbf{A}}_i &= \mathbf{A}_i - \mathbf{A}_B^* (\mathbf{D}_B^{-T} \mathbf{d}_i^T) \quad (i \in N), \\
\tilde{\mathbf{A}}\mathbf{X} &= \left(\tilde{\mathbf{A}}_{p+1} \bullet \mathbf{X}, \tilde{\mathbf{A}}_{p+2} \bullet \mathbf{X}, \dots, \tilde{\mathbf{A}}_m \bullet \mathbf{X} \right)^T \text{ for every } \mathbf{X} \in \mathbb{S}^n, \\
\tilde{\mathbf{A}}^* \mathbf{y}_N &= \sum_{i \in N} \tilde{\mathbf{A}}_i y_i \text{ for every } \mathbf{y}_N \in \mathbb{R}^{m-p}.
\end{aligned}$$

Note that \mathbf{X}^* is a feasible (strictly feasible or optimal) solution of the SDP (P3) if and only if $(\mathbf{X}, \mathbf{z}) = (\mathbf{X}^*, \mathbf{D}_B^{-1} \mathbf{b}_B - \mathbf{D}_B^{-1} \mathbf{A}_B \mathbf{X}^*)$ is a feasible (strictly feasible or optimal) solution of the SDP (P1), and that $(\mathbf{y}_N^*, \mathbf{Z}^*)$ is a feasible (strictly feasible or optimal) solution of the SDP (D3) if and only if $(\mathbf{y}_B, \mathbf{y}_N, \mathbf{Z}) = (\mathbf{D}_B^{-T} (\mathbf{f} - \mathbf{D}_N^T \mathbf{y}_N^*), \mathbf{y}_N^*, \mathbf{Z}^*)$ is a feasible (strictly feasible or optimal) of the SDP (D1). Therefore, the SDP (P1) (or its dual) has a strictly feasible solution if and only if so does the SDP (P3) (or its dual), and there exists optimal solutions (\mathbf{X}, \mathbf{z}) of (P1) and (\mathbf{y}, \mathbf{Z}) of (D1) satisfying the strict complementarity $\mathbf{X} + \mathbf{Z} \succ \mathbf{O}$ if and only if there exists optimal solutions \mathbf{X}^* of (P3) and $(\mathbf{y}_N^*, \mathbf{Z}^*)$ of (D3) satisfying the strict complementarity $\mathbf{X}^* + \mathbf{Z}^* \succ \mathbf{O}$. These are important properties of the converted primal-dual pair of SDPs (P3) and (D3) that make it possible to efficiently compute accurate optimal solutions without numerical difficulty.

3 Some technical details

3.1 The number of arithmetic operations

Let T_{split} denote the number of arithmetic operations in dense computation for solving the SDP (P2) by SDPA [22], and T_{new} the one for solving the SDP (P3) by SDPA. These are estimated as follows:

$$\begin{aligned}
T_{\text{split}} &= O((n^3 m + n^2 m^2 + m^3)k), \\
T_{\text{new}} &= O((n^3(m-p) + n^2(m-p)^2 + (m-p)^3)k) \\
&\quad + O(p^2 m) + O((m-p)(p^2 + pn^2)),
\end{aligned}$$

where k is the number of iteration of SDPA. T_{split} and the first term in T_{new} are for all computation in SDPA to solve the SDPs (P2) and (P3), respectively. See [4]. Note that when the data matrices \mathbf{A}_i ($i = 1, 2, \dots, m$) are sparse and/or have special structures such

Table 1: Numbers of arithmetic operations when $m - p = O(1)$

case	T_{split}	T_{new}
$m = O(n^2)$	$O(n^6k)$	$O(n^3k + n^6)$
$m = O(n\sqrt{n})$	$O(n^5k)$	$O(n^3k + n^{4.5})$
$m = O(n)$	$O(n^4k)$	$O(n^3k)$

as a rank one structure [8], the number of arithmetic operations to compute the elements in the Schur complement matrix \mathbf{B} per iteration, which amounts to $O(n^3m + n^2m^2)$, can be reduced substantially. The second term $O(p^2m)$ in T_{new} is for a QR decomposition of \mathbf{D}^T to choose p linearly independent vectors \mathbf{d}_i^T ($i \in B$). The third term $O((m-p)(p^2 + pn^2))$ is for computing $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{A}}$.

We assume that $p \geq 1$, otherwise the original SDP (P1) is itself the standard form SDP. If $m - p = O(m)$ then $T_{\text{new}} = O((n^3m + n^2m^2 + m^3)k)$; hence T_{new} is of the same order as T_{split} and our conversion method does not cause serious decrease of computational time in this case.

On the other hand, when $m - p = O(1)$, we see that $T_{\text{new}} = O(n^3k + m^3 + mn^2)$. Table 1 shows the order of T_{split} and T_{new} for three cases on how m relates to n . In all cases, the order of T_{new} becomes smaller than the one of T_{split} . Hence, solving the SDP (P3) is expected to be faster than solving the SDP (P2).

3.2 Greedy heuristic for choosing a basis index set B

As we mentioned in the Introduction, maintaining the sparsity of the original SDP (P1) is important to implement our conversion method. Each data matrix $\tilde{\mathbf{A}}_i$ of the converted SDP (P3) is computed as $\tilde{\mathbf{A}}_i = \mathbf{A}_i - \mathcal{A}_B^*(\mathbf{D}_B^{-T} \mathbf{d}_i^T)$ ($i \in N$). Hence the sparsity of the matrix $\tilde{\mathbf{A}}_i$ ($i \in N$) is the same with the aggregate sparsity of the matrix \mathbf{A}_i and the p matrices \mathbf{A}_k ($k \in B$). If all of the data matrices \mathbf{A}_j ($j = 1, 2, \dots, m$) are fully dense, each matrix $\tilde{\mathbf{A}}_i$ ($i \in N$) is also fully dense. If the data matrices \mathbf{A}_j ($j = 1, 2, \dots, m$) are sparse and have different sparsity patterns, however, the sparsity of the matrix $\tilde{\mathbf{A}}_i$ becomes worse than the matrix \mathbf{A}_i ($i \in N$). Moreover, the sparsity of the matrix $\tilde{\mathbf{A}}_i$ ($i \in N$) depends on how we choose p column vectors \mathbf{d}_i^T ($i \in B$) from \mathbf{D}^T . Many SDP software packages exploit sparsity of a problem to reduce the computational time for solving the problem depending on its sparsity. More specifically, as the sparsity of the data matrices of the problem increases, the problem can be solved faster. See the papers [5, 7, 13, 14]. In order to maintain the sparsity of the original SDP (P1) as much as possible, we need to choose p column vectors \mathbf{d}_i^T ($i \in B$) from \mathbf{D}^T so that the density of the data matrices $\tilde{\mathbf{A}}_i = \mathbf{A}_i - \mathcal{A}_B^*(\mathbf{D}_B^{-T} \mathbf{d}_i^T)$ ($i \in N$) of the converted SDP (P3) is minimized. However, choosing such column vectors is very hard. Therefore we use the following heuristic method. Given $\mathbf{D}^T = [\mathbf{d}_1^T \mathbf{d}_2^T \dots \mathbf{d}_m^T]$, let $E = \{1, 2, \dots, m\}$ be the set of indices of column vectors of \mathbf{D}^T , and ϕ the set of subsets S of E such that \mathbf{d}_i^T ($i \in S$) are linearly independent. Removing columns from a set of columns \mathbf{d}_i^T ($i \in S$) produces another set of columns which are linearly independent.

Moreover, if S_p and S_{p+1} are elements of ϕ containing p and $p + 1$ indices, we always can find an index $e \in S_{p+1} \setminus S_p$, satisfying the property that $S_p \cup \{e\}$ is a subset of E such that \mathbf{d}_i^T ($i \in S_p \cup \{e\}$) are linearly independent. Hence, the system (E, ϕ) forms a matric matroid. One associates a weight w_i for each $i \in E$, which is the number of nonzero elements in the matrix \mathbf{A}_i . The weight of any element S of ϕ is the sum of the weights of its elements. The matroid optimization problem is to find the $B \in \phi$ with minimum weight, which is solved by greedy algorithm. See [1]. As a solution of this matroid optimization problem, we have the $B \in \phi$ such that $\mathbf{d}_i^T \in \mathbb{R}^p$ ($i \in B$) form a basis of \mathbb{R}^p and that \mathbf{A}_i ($i \in B$) are relatively sparse among the set of matrices \mathbf{A}_i ($i = 1, 2, \dots, m$); hence the data matrices $\tilde{\mathbf{A}}_i = \mathbf{A}_i - \mathbf{A}_B^* (\mathbf{D}_B^{-T} \mathbf{d}_i^T)$ ($i \in N$) in the resulting SDP (P3) are expected to become sparse. In order to choose linearly independent column vectors of \mathbf{D}^T , we use a QR factorization of \mathbf{D}^T . By a QR factorization of $\mathbf{D}^T \in \mathbb{R}^{p \times m}$, we have an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{p \times p}$ and an upper triangular matrix $\mathbf{R} \in \mathbb{R}^{p \times m}$ which satisfy $\mathbf{D}^T = \mathbf{Q} \times \mathbf{R}$. For each i th row of the matrix \mathbf{R} ($i = 1, 2, \dots, p$), we define an index $j(i)$ by

$$j(i) = \operatorname{argmin}_{j \in \{1, 2, \dots, m\}} \{j : R_{ij} \neq 0\}$$

where R_{ij} is the (i, j) element of \mathbf{R} . Then $\mathbf{d}_{j(i)}^T \in \mathbb{R}^p$ ($i = 1, 2, \dots, p$) are linearly independent and form a basis of \mathbb{R}^p . In our implementation of the greedy heuristic, we reorder \mathbf{d}_i^T of \mathbf{D}^T in the ascending order of weights w_i before applying a QR factorization in order to incorporate the sparsity of \mathbf{d}_i^T ($i = 1, 2, \dots, m$).

3.3 Numerical evaluation of the greedy heuristic

In order to evaluate the above greedy algorithm, we executed the following numerical experiment. We derived two SDPs of the form (P3) from the original SDP (P1). The one was generated by using the greedy heuristic algorithm, and the other as follows. We compute a QR factorization of the matrix \mathbf{D}^T with column pivoting by the *LAPACK* [2] function `dgeqp3`. As an output of this function, we have a pivoting vector. By using this pivoting information, we choose a basis index set B . This method pays no attention to the sparsity of the original SDP (P1) so that the resulting SDP (P3) may be much denser than the original SDP (P1). We implemented these two conversion methods in C++ language, and we used *LAPACK* [2] for dense computation of matrices and vectors.

In this experiment, we solved randomly generated sparse problems. We fixed the size n of the variable matrix \mathbf{X} to be 500 and the number m of constraints to be 500. The number p of free variables is 10, 30 or 100. In addition, each problem involves a sparsity parameter $q \in (0, 1]$. By using this parameter, the problem was generated so that the number of nonzero elements in the matrix \mathbf{A}_i becomes nearly $\frac{q}{2} \times n^2$ ($i = 1, 2, \dots, m$). As a result, we have a problem in which the sparsities of the data matrices \mathbf{A}_i ($i = 1, 2, \dots, m$) are different from each other depending on the parameter q . Each nonzero entry in the matrices was a randomly generated number with a uniform distribution. Moreover, the matrix \mathbf{D} was generated so that the number of nonzero elements becomes nearly $q \times mp$ and each entry was generated by a uniform random number which took value between 0 and m . The matrix \mathbf{C} was taken to be the identity matrix \mathbf{I} and each entry of the vectors \mathbf{f} and \mathbf{b} was generated by a uniform random number which took value between 0 and 1. By generating \mathbf{D} and \mathbf{f}

In this way, there surely exists a vector \mathbf{y} which satisfies $\mathbf{D}^T \mathbf{y} = \mathbf{f}$. Each entry of \mathbf{D} takes a value between 0 and m and each entry of \mathbf{f} takes a value between 0 and 1 so that the entries of \mathbf{y} are expected to be small. Thus $\mathbf{C} - \mathcal{A}^* \mathbf{y} (= \mathbf{I} - \mathcal{A}^* \mathbf{y})$ is expected to be positive semidefinite. In our numerical experiment, we ran SDPA 6.2 with the default setting. When the primal infeasibility

$$\text{p.err} = \max \{ |\mathbf{A}_i \bullet \mathbf{X}^k - b_i| : i = 1, 2, \dots, m \}, \quad (1)$$

the dual infeasibility

$$\text{d.err} = \max \left\{ \left| \left[\sum_{i=1}^m \mathbf{A}_i y_i^k + \mathbf{Z}^k - \mathbf{C} \right]_{p,q} \right| : p, q = 1, 2, \dots, n \right\}, \quad (2)$$

and the relative gap

$$\text{gap} = \frac{|\sum_{i=1}^m b_i y_i^k - \mathbf{C} \bullet \mathbf{X}^k|}{\max \{ (|\sum_{i=1}^m b_i y_i^k| + |\mathbf{C} \bullet \mathbf{X}^k|) / 2.0, 1.0 \}} \quad (3)$$

are smaller than $1.0\text{E-}7$, SDPA stops and outputs the current iterate $(\mathbf{X}^k, \mathbf{y}^k, \mathbf{Z}^k)$ as an approximate optimal solution.

Table 2 shows the computational time per iteration of SDPA applied to the converted SDP (P3) with and without the above heuristic algorithm for choosing a basis index set B . As these results show, the heuristic algorithm is effective to maintain the sparsity of the original SDP (P1) in the converted SDP (P3). Thus it is reasonable to use this heuristic algorithm from the viewpoint of computational time. We observed in some cases that the converted SDP (P3) obtained with the heuristic algorithm was difficult to be solved stably. In those cases, we tried the conversion without the heuristic algorithm so that the converted SDP could be solved stably. In our computational results in the next section, we basically use this heuristic algorithm for choosing a basis index set B . In cases that the stability is more important than the computational time, however, we do not use this heuristic algorithm.

4 Computational results

As we described in the former sections, two advantages of the new conversion method are (a) the resulting SDP has a smaller size and (b) it can be more stably solved because the resulting SDP and its dual have interior-feasible solutions whenever the original primal-dual pair of SDPs have interior-feasible solutions. In this section, we present computational results for evaluating these two advantages of the new conversion method. The SDPs having free variables used in this experiment are: randomly generated dense SDPs, norm minimization problems with free variables, sparse SDP relaxations of polynomial optimization problems [19], electronic structure calculation in chemical physics [6] and SDPLIB problems [3] with additional free variables.

The numerical experiment was done on Pentium IV (Xeon) 2.4GHz with 6GB memory. We mainly used SDPA 6.2 with its default setting. In addition to SDPA, we also used SeDuMi version 1.05 [17] for sparse SDP relaxations of polynomial optimization problems [19],

Table 2: Effect of the greedy heuristic algorithm (time per iteration in seconds)

p	q	without heuristic	with heuristic
10	0.1	1.39	1.40
10	0.3	1.54	1.56
10	1.0	50.77	1.93
30	0.1	1.41	1.41
30	0.3	1.53	1.52
30	1.0	12.47	1.95
100	0.1	1.36	1.37
100	0.3	1.53	1.49
100	1.0	5.54	1.91

$m = 500, n = 500$ for all of the problems

where the coefficient matrix \mathbf{B} in the Schur complement equation $\mathbf{B}\mathbf{d}\mathbf{y} = \mathbf{r}$ of the resulting SDP becomes sparse. The reason is that SeDuMi employs the sparse Cholesky factorization, which is essential to significantly reduce the computational time, whereas SDPA does not. We also used SDPT3 version 3.02 [18] for comparison between SDP packages.

In tables in this section, we use the following symbols and notation. A figure in ‘cpu’ column denotes the total computational time (second) of SDPA, SeDuMi or SDPT3, and a figure in a bracket on the right side of the computational time is the number of iteration. A figure in ‘pre’ column denotes the time (second) to convert the original form SDP (P1) into the converted SDP (P3). A figure in ‘cpu/it’ or ‘c/i’ column denotes the average computational time (second) per iteration. We use the notation ‘p.err’ to denote a primal infeasibility error, the notation ‘d.err’ to denote a dual infeasibility error, and the notation ‘gap’ to denote a relative gap. In the SDPA case, they are defined by (1), (2) and (3), respectively. In the SeDuMi and SDPT3 cases, they are defined by similar but slightly different formulae. See [17, 18] for details.

When we explained the SDP (P2) in Section 2.1, we included the diagonal matrix variables $\text{diag}(\mathbf{z}_+)$ and $\text{diag}(\mathbf{z}_-)$ corresponding to the two nonnegative variable vectors \mathbf{z}_+ and \mathbf{z}_- into the semidefinite cone. However, in this numerical experiment, we directly treat the SDP (P2) as a mixed conic problem over linear and semidefinite cones for efficiency of the computation. In applying SeDuMi to the original SDP (P1), we specified a block of free variables of size p via the assignment `K.f=p` [17]. Similarly, in applying SDPT3 to the original SDP (P1), we specified a block of free variables of size p via the assignments `blk{1,1}='u'` and `blk{1,2}=p` [18].

4.1 Randomly generated dense SDPs

Table 3 shows numerical results on SDPA applied to randomly generated dense SDPs of the form (P1) converted into the SDP (P2) and into the SDP (P3). For these problems, we fixed the size n of the variable matrix \mathbf{X} to be 200 and the number m of constraints to be 100. The number p of free variables is 10, 30, 50, 70 or 90, and the matrices \mathbf{A}_i ($i = 1, 2, \dots, m$)

Table 3: Fully dense problems solved by SDPA

m	n	p	SDP (P2)			SDP (P3)					
			cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
100	200	10	30(11)*	2.7	2e-2	24	36(16)	2.3	3e-8	2e-14	2e-15
100	200	30	28(11)*	2.6	7e-2	22	27(16)	1.7	3e-8	6e-14	2e-15
100	200	50	26(10)*	2.6	2e-1	18	18(16)	1.1	7e-8	4e-15	7e-16
100	200	70	26(10)*	2.6	2e-1	14	11(16)	0.69	4e-8	2e-11	1e-14
100	200	90	28(11)*	2.6	9e-2	11	5(17)	0.29	2e-8	4e-14	6e-16

(* in the ‘cpu’ column indicates a termination of SDPA before convergence)

are randomly generated with a uniform distribution. The original SDPs (P1) are fully dense so that the converted SDPs (P3) are also fully dense no matter how we choose the index set B . Thus, we did not use the greedy heuristic to derive the converted SDPs (P3). We observe that SDPA was able to solve all problems converted into the SDP (P3); it found an optimal solution of each problem whose primal infeasibility, dual infeasibility and the relative gap are smaller than $1.0e-7$. In contrast to this, SDPA was able to solve none of the problems converted into the SDP (P2); it terminated before finding an optimal solution with the same accuracy due to either (a) the coefficient matrix \mathbf{B} in the Schur complement equation $\mathbf{B}\mathbf{d}\mathbf{y} = \mathbf{r}$ is not positive definite or (b) the step length cannot be determined. We also see that as the p value becomes closer to the m value, the computational time and the computational time per iteration of SDPA applied to the converted SDP (P3) becomes smaller. Therefore the conversion into the SDP (P3) is not only numerically stable but also efficient in computational time than the conversion into the SDP (P2) in this numerical experiment. Table 4 shows results on SeDuMi, which are similar to the ones on SDPA. That is, SeDuMi was able to solve none of the original SDPs (P1), and as the p value becomes closer to the m value, the computational time and the computational time per iteration of SeDuMi applied to the converted SDP (P3) becomes smaller. Table 5 shows results on SDPT3. In contrast to SDPA and SeDuMi, SDPT3 was able to solve all of the original SDPs (P1) and the converted SDPs (P3). The computational time and the computational time per iteration of SDPT3 applied to the SDP (P3) is smaller than the one of SDPT3 applied to the SDP (P1).

4.2 Norm minimization problems with free variables

Let $\mathbf{F}_i \in \mathbb{R}^{t \times r}$ ($i = 0, 1, \dots, q$) The norm minimization problem is then defined as follows:

$$\min \|\mathbf{F}_0 + \sum_{i=1}^q \mathbf{F}_i y_i\| \text{ subject to } y_i \in \mathbb{R} \text{ (} i = 1, 2, \dots, q\text{)}.$$

where $\|\mathbf{G}\|$ is the spectral norm of \mathbf{G} ; i.e., the square root of the maximum eigenvalue of $\mathbf{G}^T \mathbf{G}$. If we define

$$m = q + 1, \mathbf{b} = (0, 0, \dots, -1)^T \in \mathbb{R}^m,$$

Table 4: Fully dense problems solved by SeDuMi

m	n	p	SDP (P1)			SDP (P3)					
			cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
100	200	10	256(12)*	21	3e-7	24	212(16)	13	3e-10	8e-10	0e+0
100	200	30	368(12)*	31	4e-7	22	126(15)	8.4	8e-10	2e-9	0e+0
100	200	50	356(11)*	32	1e-6	18	105(16)	6.6	3e-10	3e-10	0e+0
100	200	70	306(12)*	26	2e-7	14	106(14)	7.6	2e-8	4e-7	0e+0
100	200	90	590(17)*	35	9e-10	11	23(15)	1.5	8e-10	8e-10	0e+0

(* in the ‘cpu’ column indicates a termination of SeDuMi with an error termination code)

Table 5: Fully dense problems solved by SDPT3

m	n	p	SDP (P1)			SDP (P3)					
			cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
100	200	10	68(13)	5.2	8e-10	24	65(14)	4.6	9e-10	1e-7	1e-15
100	200	30	64(12)	5.3	4e-9	22	45(13)	3.5	3e-9	5e-8	2e-15
100	200	50	63(12)	5.3	3e-9	18	32(13)	2.5	3e-9	4e-9	6e-16
100	200	70	63(12)	5.3	2e-9	14	20(13)	1.5	3e-9	3e-10	8e-15
100	200	90	64(12)	5.3	6e-9	11	10(13)	0.8	5e-9	2e-9	2e-16

$$\mathbf{A}_i = - \begin{pmatrix} \mathbf{O} & \mathbf{F}_i^T \\ \mathbf{F}_i & \mathbf{O} \end{pmatrix} \quad (i = 1, 2, \dots, q),$$

$$\mathbf{A}_{q+1} = - \begin{pmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} \mathbf{O} & \mathbf{F}_0^T \\ \mathbf{F}_0 & \mathbf{O} \end{pmatrix},$$

we can transform this problem into the dual of the standard form SDP (P0).

We first generated norm minimization problems whose data matrices and vectors are randomly generated in the dual of the standard form SDP (P0) as mentioned above. Then we randomly generated a matrix $\mathbf{D} \in \mathbb{R}^{m \times p}$ and a vector $\mathbf{f} \in \mathbb{R}^p$, and added them to the dual SDP as in (D1) and to the primal SDP as in (P1). The data matrices $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_m$ of SDP (P1) defined above have a favourable sparse structure which the converted SDP (P3) could inherit, so that the original SDP (P1) and the converted SDP (P3) have similar sparsity. To derive the converted SDPs (P3), we used the greedy heuristic. Table 6 shows some critical differences in the computational time, the computational time per iteration and the numerical stability between the conversions into the SDP (P2) and the SDP (P3). In particular, SDPA solved all four problems converted into the SDP (P3) accurately, whereas it was able to solve none of the problems converted into the SDP (P2). The reason of a termination of SDPA applied to each of the converted SDP (P2) is that the coefficient matrix \mathbf{B} in the Schur complement equation $\mathbf{B}\mathbf{d}\mathbf{y} = \mathbf{r}$ was not positive definite during the iterations. Table 7 shows results on SeDuMi and Table 8 shows results on SDPT3. SeDuMi and SDPT3 were able to solve all problems in the original SDP (P1) and in the

Table 6: Norm minimization problems with free variables solved by SDPA

m	n	p	SDP (P2)			SDP (P3)					
			cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
500	500	100	890(7)*	127	2e+0	7	1334(16)	83	7e-8	7e-16	3e-15
500	500	200	1002(8)*	125	2e+0	12	936(16)	59	1e-8	7e-16	2e-15
500	500	300	1184(9)*	132	2e+0	13	819(17)	48	2e-8	1e-15	1e-15
500	500	400	1188(9)*	132	2e+0	9	526(17)	31	3e-8	2e-15	1e-15

(* in the 'cpu' column indicates a termination of SDPA before convergence)

Table 7: Norm minimization problems with free variables solved by SeDuMi

m	n	p	SDP (P1)			SDP (P3)					
			cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
500	500	100	312(12)	26	2e-12	7	105(5)	21	4e-10	1e-10	0e+0
500	500	200	278(10)	28	6e-10	12	106(5)	21	1e-12	5e-12	0e+0
500	500	300	276(10)	28	3e-10	13	99(5)	20	3e-14	4e-12	0e+0
500	500	400	339(12)	28	6e-11	9	94(5)	19	2e-14	4e-12	0e+0

converted SDP (P3). The computational time of SeDuMi/SDPT3 applied to the SDP (P3) is smaller than the one applied to the SDP (P1). The computational time per iteration of SDPT3 applied to the SDP (P3) is smaller than the one of the SDP (P1), whereas the computational time per iteration of SeDuMi applied to the SDP (P3) is similar to the one of SeDuMi applied to the SDP (P1). We note that the number of iterations of SeDuMi applied to the SDP (P3) is smaller than the one of SeDuMi applied to the SDP (P1).

4.3 Sparse SDP relaxations of polynomial optimization problems

Sparse SDP relaxations of polynomial optimization problems [19] are formulated as the dual form SDPs (D1). Test problems are chosen from [20]. An important feature of SDPs arising from sparse SDP relaxations of polynomial optimization problems is that the coefficient

Table 8: Norm minimization problems with free variables solved by SDPT3

m	n	p	SDP (P1)			SDP (P3)					
			cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
500	500	100	182(11)	17	9e-10	7	222(11)	20	1e-9	9e-9	3e-15
500	500	200	181(11)	16	2e-9	12	184(11)	17	3e-9	3e-9	3e-15
500	500	300	187(11)	17	2e-9	13	125(11)	11	4e-9	5e-9	3e-15
500	500	400	189(11)	17	3e-9	9	67(11)	6	4e-9	4e-10	3e-15

Table 9: Sizes of SDP relaxation of polynomial optimization problems

problem	m	free var.	LP var.	n (block structures)
ex5_case1	714	188	1410	[55×1, 10×20]
ex5_case2	714	188	1410	[55×1, 10×20]
ex5_case3	714	188	1410	[55×1, 10×20]
ex9_1_1	2379	955	4732	[105×1, 14×26]
ex9_1_2	1000	468	1980	[66×1, 11×20]
ex9_1_4	1000	468	1980	[66×1, 11×20]
ex9_1_5	2379	956	4732	[105×1, 14×26]
ex9_1_8	1819	705	3614	[91×1, 13×25]
ex9_2_1	1000	468	1980	[66×1, 11×20]
ex9_2_2	494	250	972	[45×1, 9×18]
ex9_2_4	494	224	972	[45×1, 9×16]
ex9_2_5	494	250	972	[45×1, 9×16]
ex9_2_6	1819	705	3614	[91×1, 13×24]
ex9_2_7	1000	468	1980	[66×1, 11×20]
ex9_2_8	209	115	406	[28×1, 7×12]

matrix in the Schur complement equation becomes sparse. Therefore we used SeDuMi for solving these problems, which employs the sparse Cholesky factorization to solve the Schur complement equation. We did not use the greedy heuristic algorithm for choosing a basis index set B because the numerical stability was more critical than the computational time for these SDPs. Sizes of the problems are shown in Table 9. An entry $[55 \times 1, 10 \times 20]$ in the ' n (block structures)' column means that the problem has 21 SDP blocks, the size of the first block is 55 and the sizes of the remaining 20 blocks are 10. Table 10 shows results on SeDuMi applied to the original SDP (P1) and the converted SDP (P3) for solving these problems. SeDuMi was able to find more accurate optimal solutions of all problems converted into the SDP (P3) than the original form (P1) except ex9_1.8. We also see that the computational time per iteration of the SDP (P3) is smaller than the one of the SDP (P1) for all problems in Table 10. In these SDPs, the number p of free variables is large so that the number of constraints in the converted SDP (P3) becomes smaller than the one of the original SDP (P1). This contributed to the reduction of the computational time.

4.4 Electronic structure calculation in chemical physics

SDPs with equality constraints arise in electronic structure calculation in chemical physics [6], which are formulated as the dual form SDPs (D1). Their sizes of Table 11 can be so huge that parallel computation is often inevitable to solve them [6]. Here we show some numerical results for solving small size problems. Because of the nature of these SDPs, we need to adjust a parameter of SDPA in order to solve them stably. In particular, we adjusted a parameter 'lambdaStar' of SDPA, which controls the magnitude of the initial point, from the default value 1.0e2 to 1.0e0. To derive the converted SDPs (P3), we used the greedy heuristic. Table 12 shows results on SDPA applied to the SDP (P2) and the

Table 10: SDP relaxation of polynomial optimization problems solved by SeDuMi

problem	SDP (P1)			SDP (P3)					
	cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
ex5_case1	29(37)*	0.78	1e-7	0.43	18(37)*	0.49	5e-8	2e-8	5e-10
ex5_case2	27(35)*	0.77	2e-8	0.45	17(35)*	0.49	2e-8	4e-9	1e-10
ex5_case3	25(32)*	0.78	3e-8	0.44	16(32)*	0.5	2e-8	2e-9	7e-11
ex9_1_1	536(22)*	24	2e-9	31	116(21)	5.5	4e-11	1e-12	3e-14
ex9_1_2	47(23)*	2.0	3e-8	3.3	11(26)	0.42	8e-10	4e-11	2e-12
ex9_1_4	60(30)*	2.0	1e-9	3.4	14(36)	0.39	5e-11	8e-12	3e-13
ex9_1_5	543(22)*	25	2e-7	31	130(21)*	6.2	1e-7	5e-9	7e-11
ex9_1_8	326(30)*	11	1e-7	13	74(24)*	3.1	8e-6	4e-7	7e-9
ex9_2_1	50(24)*	2.1	5e-9	3.4	17(26)*	0.65	2e-10	1e-11	5e-13
ex9_2_2	46(56)*	0.82	8e-11	1.1	12(56)*	0.21	6e-10	3e-9	2e-10
ex9_2_4	5.0(17)*	0.29	4e-9	0.35	1.9(22)	0.09	2e-12	2e-13	2e-15
ex9_2_5	5.2(15)*	0.35	1e-8	0.45	1.7(18)	0.09	9e-12	7e-13	4e-14
ex9_2_6	371(34)*	11	9e-12	13	97(35)*	2.8	6e-12	2e-13	4e-15
ex9_2_7	47(23)*	2.0	5e-8	3.4	15(25)	0.6	8e-10	5e-11	2e-12
ex9_2_8	0.92(15)*	0.06	2e-9	0.04	0.37(14)	0.03	2e-12	3e-13	1e-14

(* in the 'cpu' column indicates a termination of SeDuMi before convergence)

SDP (P3). We note that the 'd.err' values, which are not presented in the table, are smaller than $1.0e-7$ for all problems in both cases of the SDP (P2) and the SDP (P3). However, the 'gap' values and the 'p.err' values of SDPA applied to the SDP (P3) are smaller than or of almost the same magnitude as those of SDPA applied to the SDP (P2) for all problems except H3.2A1.DZ.r12. The computational time per iteration of SDPA applied to the SDP (P3) is larger than the one of SDPA applied to the SDP (P2). In these SDPs, the number of constraints m in the original SDP (P1) is small so that the number of constraints in the converted SDP (P3) is nearly the same with the one of SDP (P1). Moreover, a few of the data matrices $\mathbf{A}_i (i = 1, 2, \dots, m)$ are dense and the other data matrices are very sparse. Thus, when an index i of a dense matrix \mathbf{A}_i happens to be in the basis index set B , the data matrices in the converted SDP (P3) become denser than those in the original SDP (P1). This resulted in an increase of the computational time of SDPA applied to the SDP (P3). Table 13 shows results on SeDuMi. As in the case of SDPA, the 'd.err' values, which are not presented in the table, are smaller than $1.0e-7$ for all problems in both cases of SDP (P1) and the SDP (P3). The 'p.err' values of SeDuMi applied to the SDP (P3) are smaller than those of SeDuMi applied to the SDP (P1) for all problems, and the 'gap' values of SeDuMi applied to the SDP (P3) are smaller than or of almost the same magnitude as those of SeDuMi applied to the SDP (P1) for all problems except FH2+.1A1.STO6G.r14.

4.5 SDPLIB problems with additional free variables

We modified benchmark test problems from SDPLIB [3]. Each problem in SDPLIB is formulated as a standard form SDP (P0). To generate an SDP having free variables, we

Table 11: Sizes of electronic structure calculation problems in chemical physics

problem	m	free var.	n (block structures)
Be.1S.STO6G.r10	465	34	[5×4, 10×6, 25×4, 50×5, 175×2]
BeH+.1Sigma+.STO6G.r12	948	46	[6×4, 15×4, 20×2, 36×4, 72×1, 90×4, 306×2]
CH3.2A2.STO6G.r16	2964	76	[8×4, 28×4, 56×2, 64×4, 128×1, 224×2]
FH2+.1A1.STO6G.r14	1743	60	[7×4, 21×4, 35×2, 49×4, 98×1, 147×4, 490×2]
H3.2A1.DZ.r12	948	46	[6×4, 15×4, 20×2, 36×4, 72×1, 90×4, 306×2]
Li.2S.STO6G.r10	465	34	[5×4, 10×6, 25×4, 50×5, 175×2]
NH2-.1A1.STO6G.r14	1743	60	[7×4, 21×4, 35×2, 49×4, 98×1, 147×4, 490×2]
NH4+.1A1.STO6GN.r18	4743	94	[9×4, 36×4, 81×4, 84×2, 162×1, 324×2]

Table 12: Electronic structure calculation problems in chemical physics solved by SDPA

problem	SDP (P1)				SDP (P3)				
	cpu	c/i	gap	p.err	pre	cpu	c/i	gap	p.err
Be.1S.STO6G.r10	53(20)	2.7	6e-8	2e-9	1.1	90(19)	4.7	9e-8	1e-10
BeH+.1Sigma+.STO6G.r12	618(24)	26	7e-8	3e-9	2.7	2952(25)	118	5e-8	2e-11
CH3.2A2.STO6G.r16	637(16)*	40	6e-3	1e-3	2	904(24)*	38	2e-7	4e-7
FH2+.1A1.STO6G.r14	2895(20)*	145	3e-4	1e-2	8	9545(28)	341	4e-8	2e-8
H3.2A1.DZ.r12	570(22)	26	9e-10	6e-10	2.7	2717(22)*	124	3e-7	4e-11
Li.2S.STO6G.r10	54(20)	2.7	6e-8	6e-10	1.1	88(19)	4.6	9e-8	1e-12
NH2-.1A1.STO6G.r14	2895(20)*	145	4e-4	2e-3	7.4	6098(18)*	339	1e-4	2e-4
NH4+.1A1.STO6GN.r18	3230(25)	129	4e-8	1e-7	4.0	2874(22)	131	1e-7	9e-8

(* in the 'cpu' column indicates a termination of SDPA before convergence)

Table 13: Electronic structure calculation problems in chemical physics solved by SeDuMi

problem	SDP (P1)				SDP (P3)				
	cpu	c/i	gap	p.err	pre	cpu	c/i	gap	p.err
Be.1S.STO6G.r10	49(24)	2.0	3e-8	7e-9	1.1	51(22)	2.3	9e-8	6e-10
BeH+.1Sigma+.STO6G.r12	362(29)	12	5e-8	1e-8	2.7	428(29)	15	2e-8	2e-10
CH3.2A2.STO6G.r16	3240(55)*	59	8e-9	6e-8	2	2689(51)	53	7e-8	4e-9
FH2+.1A1.STO6G.r14	2225(33)	67	2e-7	5e-8	8	2515(32)	79	1e-6	7e-9
H3.2A1.DZ.r12	588(37)*	16	9e-9	3e-9	2.7	967(47)*	21	9e-10	8e-11
Li.2S.STO6G.r10	42(21)	2.0	2e-9	6e-10	1.1	43(19)	2.3	4e-8	2e-10
NH2-.1A1.STO6G.r14	2331(35)	67	9e-8	3e-8	7.4	2560(33)	78	7e-7	8e-9
NH4+.1A1.STO6GN.r18	10949(44)*	249	3e-8	8e-8	4.0	10027(44)	228	4e-7	1e-8

(* in the 'cpu' column indicates a termination of SeDuMi with an error termination code)

Table 14: Sizes of problems from SDPLIB with free variables

problem	m	free var.	n
theta3	1106	365	150
theta4	1949	649	200
qap7	358	119	50
qap10	1021	340	101
mcp124-3	124	41	124
mcp124-4	124	41	124
mcp250-3	250	83	250
mcp250-4	250	83	250
gpp124-3	125	41	124
gpp124-4	125	41	124
gpp250-2	251	83	250
gpp250-3	251	83	250

randomly generated a matrix $\mathbf{D} \in \mathbb{R}^{m \times p}$ and a vector $\mathbf{f} \in \mathbb{R}^p$ and add them to this SDP (P0), where p was taken to be $m/3$. In this way, we had a set of SDPs having free variables. To derive the converted SDPs (P3), we used the greedy heuristic. Tables 15, 16 and 17 show results on SDPA applied to the standard form SDP (P2) and the converted SDP (P3), results on SeDuMi applied to the original SDP (P1) and the converted SDP (P3), results on SDPT3 applied to the original SDP (P1) and the converted SDP (P3), respectively. SDPA solved 7 of the problems converted into the SDP (P3) accurately, while it was able to solve none of the problems converted into the SDP (P2) because of some numerical difficulties. SeDuMi also found more accurate solutions of the SDP (P3) than the original SDP (P1) in all cases except mcp124-3, mcp124-4, mcp250-3 and mcp250-4. Therefore, the conversion into the SDP (P3) is effective in SDPA and SeDuMi in those cases. For mcp124-4, mcp250-3 and mcp250-4, SeDuMi found solutions of the SDP (P1) whose ‘gap’ are smaller than those of solutions of the SDP (P3), whereas the ‘p.err’ of solutions of the SDP (P1) are larger than those of solutions of the SDP (P3); hence we cannot say which solutions are better than the others. For mcp124-3, SeDuMi found solutions of the original SDP (P1) whose ‘gap’ and ‘p.err’ are smaller than those of the solutions of the SDP (P3). On the other hand, SDPT3 was able to find accurate optimal solutions for all problems in the SDP (P1) and the SDP (P3) forms except gpp124-4 in the SDP (P1) form. The computational time of SDPA/SDPT3 applied to the converted SDP (P3) is larger than the one of SDPA/SDPT3 applied to the SDP (P2)/SDP (P1). The reason is that the data matrices in the converted problems become denser than the data matrices in the original SDP (P1). It is interesting, however, to observe that the computational time of SeDuMi applied to the converted SDP (P3) is smaller than the one of SeDuMi applied to the original SDP (P1) for 8 problems. This difference may be caused by the fact that the sparsity of data matrices are exploited differently in these software packages.

Table 15: SDPLIB with free variables solved by SDPA

problem	SDP (P2)			SDP (P3)					
	cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
theta3	15(4)*	3.8	2e+0	50	116(20)	5.8	6e-10	1e-11	1e-12
theta4	57(3)*	19	2e+0	211	486(20)	24	1e-9	4e-13	3e-12
qap7	0.60(5)*	0.12	5e-1	3.0	2.5(17)	0.15	5e-10	9e-12	2e-12
qap10	14(5)*	2.8	1e+0	56	52(19)	2.7	2e-10	9e-9	3e-11
mcp124-3	0.36(9)*	0.040	2e+0	0.26	2.2(24)*	0.092	6e-8	4e-8	1e-11
mcp124-4	0.38(9)*	0.042	2e+0	0.51	2.0(22)*	0.091	5e-8	3e-9	3e-12
mcp250-3	2.3(9)*	0.26	2e+0	1.1	56(21)*	2.7	2e-7	3e-7	1e-11
mcp250-4	2.5(10)*	0.25	2e+0	2.0	68(24)*	2.8	4e-7	2e-8	7e-12
gpp124-3	0.51(10)*	0.051	2e-2	11	2.0(21)*	0.095	8e-9	6e-10	2e-12
gpp124-4	0.40(8)*	0.050	7e-2	11	2.1(22)	0.095	3e-10	2e-8	2e-12
gpp250-2	2.4(8)*	0.30	2e-1	200	55(19)	2.9	2e-10	6e-10	2e-12
gpp250-3	2.3(8)*	0.29	1e-1	198	49(19)	2.6	2e-10	1e-9	8e-12

(* in the 'cpu' column indicates a termination of SDPA before convergence)

Table 16: SDPLIB with free variables solved by SeDuMi

problem	SDP (P1)				SDP (P3)				
	cpu	cpu/it	gap	p.err	pre	cpu	cpu/it	gap	p.err
theta3	121(3)*	61	7e-3	2e-1	50	52(19)	2.7	4e-12	2e-11
theta4	332(2)*	166	2e-2	5e-1	211	223(19)	12	3e-12	4e-11
qap7	10(8)*	1.3	9e-5	3e-2	3.0	2(15)	0.13	2e-9	8e-8
qap10	207(7)*	30	9e-4	7e+0	56	40(16)	2.5	2e-10	6e-8
mcp124-3	6.7(14)*	0.48	2e-11	5e-8	0.26	4.8(14)	0.34	8e-8	7e-8
mcp124-4	6.0(14)*	0.43	5e-12	2e-8	0.51	4.6(14)	0.33	3e-8	4e-9
mcp250-3	46(13)*	3.5	2e-10	1e-6	1.1	39(15)	2.6	4e-7	4e-7
mcp250-4	51(14)*	3.6	1e-10	1e-6	2.0	41(14)	2.9	2e-7	5e-8
gpp124-3	4.4(13)*	0.34	8e-7	5e-7	11	5.7(25)	0.24	2e-12	2e-9
gpp124-4	5.1(13)*	0.39	1e-6	3e-7	11	5.5(25)	0.22	3e-13	1e-8
gpp250-2	34(15)*	2.3	1e-5	2e-5	200	41(18)	2.3	2e-10	6e-8
gpp250-3	29(12)*	2.4	1e-5	8e-6	198	51(19)	2.7	1e-11	5e-7

(* in the 'cpu' column indicates a termination of SeDuMi with an error termination code)

Table 17: SDPLIB with free variables solved by SDPT3

problem	SDP (P1)			SDP (P3)					
	cpu	cpu/it	gap	pre	cpu	cpu/it	gap	p.err	d.err
theta3	53(13)	4.1	7e-9	50	117(15)	7.8	7e-9	5e-8	3e-17
theta4	253(14)	18	2e-9	211	429(16)	27	2e-9	2e-7	3e-17
qap7	4.2(14)	0.30	2e-9	3.0	7.6(12)	0.63	2e-9	1e-9	4e-15
qap10	48(14)	3.4	1e-8	56	75(14)	5.4	2e-9	7e-10	3e-14
mcp124-3	1.9(13)	0.15	3e-9	0.26	2.7(14)	0.19	1e-8	5e-7	9e-17
mcp124-4	2.1(13)	0.16	3e-9	0.51	3.4(17)	0.20	4e-8	7e-7	9e-17
mcp250-3	8.0(14)	0.57	9e-9	1.1	27(15)	1.8	4e-8	6e-7	8e-17
mcp250-4	9.1(16)	0.57	8e-10	2.0	32(18)	1.8	5e-8	8e-7	8e-17
gpp124-3	2.6(16)	0.18	5e-9	11	3.2(16)	0.23	8e-9	3e-10	1e-14
gpp124-4	2.2(14)*	0.16	3e-8	11	3.2(16)	0.20	5e-9	6e-11	9e-15
gpp250-2	12(17)	0.71	8e-10	200	25(16)	1.6	8e-10	2e-10	1e-14
gpp250-3	12(17)	1.2	7e-9	198	26(16)	1.6	4e-10	3e-10	4e-14

(* in the 'cpu' column indicates a termination of SDPT3 with an error termination code)

5 Concluding Remarks

In this paper, we have proposed a new method to convert an SDP having free variables into the standard form SDP. The new conversion method has two advantages (a) the resulting SDP has a smaller size, and (b) it can be more stably solved. Through the numerical experiments, we have shown these two advantages. The conversion can be used as a preprocessing phase within any SDP software package. As a topic of further study, it would be worthwhile developing more effective algorithm to choose a basis index set B than the greedy heuristic algorithm described in section 3.2 so that the resulting SDP (P3) becomes sparser.

Many of the discussions on the primal-dual SDP pairs (P1) - (D1), (P2) - (D2) and (P3) - (D3) can be easily modified so as to deal with corresponding primal-dual pairs of LPs and SOCPs having free variables. How to handle free variables has been studied extensively in LPs (for example, [21]), and many software packages based on the primal-dual interior point method can solve LPs having free variables without any difficulty. But how difficult SOCPs having free variables are is not clear, at least, to the authors. If free variables in SOCPs cause numerical difficulty, the method proposed in this paper may be helpful in solving such SOCPs. However, the authors need to do extensive numerical experiments on such SOCPs to judge the practical usefulness of the proposed method to SOCPs.

Acknowledgment: The authors are grateful to the referees for their many valuable comments and suggestions that considerably improved the paper. In particular, the last remark in Section 5 is added to respond to one of the referees.

References

- [1] R. Ahuja, T. Magnanti and J. Orlin, *Network Flows – theory, algorithms, and applications*, (Prentice Hall, 1993).
- [2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenney and D. Sorensen, *LAPACK Users' Guide Third*, (SIAM, 1999).
- [3] B. Borchers, SDPLIB 1.2, a library of semidefinite programming test problems, *Optimization Methods and Software*, 11 & 12 (1999) 683-690.
- [4] K. Fujisawa, M. Fukuda, M. Kojima and K. Nakata, Numerical evaluation of SDPA (SemiDefinite Programming Algorithm), in: H. Frenk, K. Roos, T. Terlaky and S. Zhang eds., *High Performance Optimization*, (Kluwer Academic Press, 2000) 267–301.
- [5] K. Fujisawa, M. Kojima and K. Nakata, Exploiting sparsity in primal-dual interior-point methods for semidefinite programming, *Mathematical Programming*, 79 (1997) 235–253.
- [6] M. Fukuda, B.J. Braams, M. Nakata, M.L. Overton, J.K. Percus, M. Yamashita and Z. Zhao, Large-scale semidefinite programs in electronic structure calculation, Research Report B-413, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo 152-8552, February 2005.
- [7] M. Fukuda, M. Kojima, K. Murota and K. Nakata, Exploiting sparsity in semidefinite programming via matrix completion I: General framework, *SIAM Journal on Optimization*, 11 (2000) 647–674.
- [8] C. Helmberg, Semidefinite programming for combinatorial optimization, ZIP report ZR-0034, TU Berlin, Konrad-Zuse-Zentrum, Berlin, 2000, Habilitationsschrift. Available at <http://www.zip.de/PaperWeb/abstracts/ZR-0034/>.
- [9] C. Helmberg, F. Rendl, R. J. Vanderbei and H. Wolkowicz, An interior-point method for semidefinite programming, *SIAM Journal on Optimization*, 6 (1996) 342–361.
- [10] M. Kojima, S. Shindoh and S. Hara, Interior-point methods for the monotone semidefinite linear complementarity problem in symmetric matrices, *SIAM Journal on Optimization*, 7 (1997) 86–125.
- [11] J. B. Lasserre, Global optimization with polynomials and the problems of moments, *SIAM Journal on Optimization*, 11 (2001) 796–817.
- [12] R. D. C. Monteiro, Primal-dual path-following algorithms for semidefinite programming, *SIAM Journal on Optimization*, 7 (1997) 663–678.
- [13] 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*, 95 (2003) 303–327.

- [14] 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, Meguro, Tokyo 152-8552, November 2003.
- [15] M. Nakata, H. Nakatsuji, M. Ehara, M. Fukuda, K. Nakata and K. Fujisawa, Variational calculations of fermion second-order reduced density matrices by semidefinite programming algorithm, *Journal of Chemical Physics*, 114 (2001) 8282-8292.
- [16] Yu. E. Nesterov and M. J. Todd, Primal-dual interior-point methods for self-scaled cones, *SIAM Journal on Optimization*, 8 (1998) 324-364.
- [17] J. F. Sturm, Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones, *Optimization Methods and Software*, 11 & 12 (1999) 625-653.
- [18] R. H. Tütüncü, K. C. Toh and M. J. Todd, Solving semidefinite-quadratic-linear programs using SDPT3, *Mathematical Programming*, 95 (2003) 189-217.
- [19] H. Waki, S. Kim, M. Kojima and M. Muramatsu, Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity, to appear in *SIAM Journal on Optimization*, September 2005.
- [20] GLOBAL Library, <http://www.gamsworld.org/global/globallib.htm>
- [21] R. J. Vanderbei and T. J. Carpenter, Symmetric indefinite systems for interior point methods, *Mathematical Programming*, 58 (1993) 1-32.
- [22] M. Yamashita, K. Fujisawa and M. Kojima, Implementation and Evaluation of SDPA6.0 (SemiDefinite Programming Algorithm 6.0), *Optimization Methods and Software*, 18 (2003) 491-505.