ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

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

Kazuhiro Kobayashi, Kazuhide Nakata and Masakazu Kojima

June 2005, B–416

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-416 A Conversion of an SDP Having Free Variables into the Standard Form SDP
Kazuhiro Kobayashi[⋆], Kazuhide Nakata[†] and Masakazu Kojima[‡], June 2005

Abstract.
This paper deals with a semidefinite program (SDP) having free variables, a minimization of a linear objective function in a positive semidefinite matrix variable and free real variables subject to linear equality constraints in those variables. This type of SDP often appears in practice. To apply the primal-dual interior-point method developed for the standard form SDP having no free variables, we need to convert our SDP into the standard from. 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 SDP to be solved but also yields some degeneracy in the resulting standard form SDP. We can also modify the primal-dual interior-point method so as to adapt it to an SDP having free variables. This paper proposes a new conversion method that eliminates all free variables. The resulting standard form SDP is smaller in its size, and it could be more stably solved in general because the conversion yields no degeneracy. Effectiveness of the new conversion method applied to SDPs having free variables is reported in comparison to some other existing methods; SDPA, SeDuMi and SDPT3 with the new conversion methods are compared to SDPA with the simple conversion mentioned above, SeDuMi without any conversion and SDPT3 without any conversion, respectively.

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. Research supported by Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 14750049, *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. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234. *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 $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{S}^n$, the inner product is defined as $\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^n \sum_{j=1}^n X_{ij} Y_{ij}$. Let $\boldsymbol{A}_i \in \mathbb{S}^n$ $(i = 1, 2, \cdots, m)$. We define the linear operator $\mathcal{A} : \mathbb{S}^n \to \mathbb{R}^m$ by $\mathcal{A}\boldsymbol{X} = (\boldsymbol{A}_1 \bullet \boldsymbol{X}, \boldsymbol{A}_2 \bullet \boldsymbol{X}, \cdots, \boldsymbol{A}_m \bullet \boldsymbol{X})^T$ for every $\boldsymbol{X} \in \mathbb{S}^n$, and the adjoint linear operator $\mathcal{A}^* : \mathbb{R}^m \to \mathbb{S}^n$ by $\mathcal{A}^*\boldsymbol{y} = \sum_{i=1}^m \boldsymbol{A}_i y_i$ for every $\boldsymbol{y} \in \mathbb{R}^m$.

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

$$\begin{array}{lll} \text{(P0)} & \text{minimize} & \boldsymbol{C} \bullet \boldsymbol{X} \\ & \text{subject to} & \mathcal{A}\boldsymbol{X} = \boldsymbol{b} \quad \text{and } \boldsymbol{X} \in \mathbb{S}^n_+. \end{array}$$

Here $\boldsymbol{C} \in \mathbb{S}^n$ and $\boldsymbol{b} \in \mathbb{R}^m$ are constant, and $\boldsymbol{X} \in \mathbb{S}^n$ is a 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 [7, 8, 9, 12]. In applications, however, various different forms of SDPs are formulated.

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

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

Here $\boldsymbol{D} \in \mathbb{R}^{m \times p}$ is a constant matrix, $\boldsymbol{f} \in \mathbb{R}^p$ is a constant vector and $\boldsymbol{z} \in \mathbb{R}^p$ is a variable. Many practical problems are formulated as SDPs of this type, which include quantum chemistry [11, 17] and polynomial optimization problems [15]. To apply the primal-dual interior-point method [7, 8, 9, 12] 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 $\boldsymbol{z} \in \mathbb{R}^p$ as a difference of two nonnegative variable vectors such as $\boldsymbol{z} = \boldsymbol{z}_+ - \boldsymbol{z}_-$, $\boldsymbol{z}_+ \in \mathbb{R}^p_+$ and $\boldsymbol{z}_- \in \mathbb{R}^p_+$, where $\mathbb{R}^p_+$ denotes the nonnegative orthant of $\mathbb{R}^p$. This conversion method is employed in the software package SeDuMi [13]. The method not only expands the size of SDP to be solved but also yield some degeneracy such that the converted SDP has continuum number 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 [15]. We can also modify the primal-dual interior-point method so as to process an SDP having free variables as done in the software package SDPT3 [14]. Solving an SDP having free variable 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 yields no degeneracy. More specifically, it is desirable that the converted standard form SDP and its dual would have interior-feasible solutions whenever the original primal-dual pair of SDPs have interior-feasible solutions. 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 ([4, 6, 10]) 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 of the problem. 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 the one 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 one is a simple conversion method referred above, and the other is the new conversion method. Section 3 is devoted to some technical details of the new conversion method including its required number of arithmetic operations 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 $\boldsymbol{z} \in \mathbb{R}^p$ as a difference of two nonnegative variable vectors $\boldsymbol{z}_+ \in \mathbb{R}^p_+$ and $\boldsymbol{z}_- \in \mathbb{R}^p_-$ such that $\boldsymbol{z} = \boldsymbol{z}_+ - \boldsymbol{z}_-$. Let $\mathrm{diag}(\boldsymbol{a})$ denote the $p \times p$ diagonal matrix with the diagonal components $a_1, a_2, \ldots, a_p$ for each $\boldsymbol{a} = (a_1, a_2, \ldots, a_p)^T \in \mathbb{R}^p$, and let $\boldsymbol{d}_i$ $(i = 1, 2, \ldots, m)$ denote the ith row of the matrix $\boldsymbol{D}$. Then the SDP (P1) can be rewritten as a standard form SDP

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

where

$$\widehat{\boldsymbol{C}} = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \mathrm{diag}(\boldsymbol{f}) & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & -\mathrm{diag}(\boldsymbol{f}) \end{pmatrix},$$

$$\widehat{\boldsymbol{A}}_i = \begin{pmatrix} \boldsymbol{A}_i & \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \mathrm{diag}(\boldsymbol{d}_i^T) & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & -\mathrm{diag}(\boldsymbol{d}_i^T) \end{pmatrix} \ (i = 1, 2, \ldots, m),$$

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

As in the case of $\mathcal{A}$, $\widehat{\mathcal{A}}$ is a linear operator defined as $\widehat{\mathcal{A}}\widehat{\boldsymbol{X}} = (\widehat{\boldsymbol{A}}_1 \bullet \widehat{\boldsymbol{X}}, \widehat{\boldsymbol{A}}_2 \bullet \widehat{\boldsymbol{X}}, \cdots, \widehat{\boldsymbol{A}}_m \bullet \widehat{\boldsymbol{X}})^T$. 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)

$$\begin{array}{lll} \text{(D1)} & \text{maximize} & \boldsymbol{b}^T\boldsymbol{y} \\ & \text{subject to} & \boldsymbol{C} - \mathcal{A}^*\boldsymbol{y} = \boldsymbol{Z},\ \boldsymbol{Z} \in \mathbb{S}^n_+ \text{ and } \boldsymbol{D}^T\boldsymbol{y} - \boldsymbol{f} = \boldsymbol{0}. \end{array}$$

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

Since $\mathrm{rank}(\boldsymbol{D}) = p$, we know that $p \le m$ and that $\boldsymbol{d}_i^T$ $(i = 1, 2, \ldots, m)$ 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 $B = \{1, 2, \ldots, p\}$ and $N = \{p+1, p+2, \ldots, m\}$. We use the notation $\boldsymbol{D}_B$ for the submatrix of $\boldsymbol{D}$ consisting of $\boldsymbol{d}_i$ $(i \in B)$ and the notation $\boldsymbol{D}_N$ for the submatrix of $\boldsymbol{D}$ consisting of $\boldsymbol{d}_i$ $(i \in N)$. Similarly, we use $\boldsymbol{y}_B$, $\boldsymbol{y}_N$, $\boldsymbol{b}_B$ and $\boldsymbol{b}_N$ for subvectors of $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{b} \in \mathbb{R}^m$ with the basis indices B and the nonbasis indices N, respectively. Note that $\boldsymbol{D}_B$ is nonsingular. Hence we can solve the equality constraint $\boldsymbol{D}^T\boldsymbol{y} - \boldsymbol{f} = \boldsymbol{0}$ in $\boldsymbol{y}_B$ such that

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

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

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

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

$$\begin{array}{rcl} \boldsymbol{b}^T\boldsymbol{y} &=& \boldsymbol{b}_B^T\boldsymbol{y}_B + \boldsymbol{b}_N^T\boldsymbol{y}_N \\ &=& \boldsymbol{b}_B^T(\boldsymbol{D}_B^{-T}\boldsymbol{f} - \boldsymbol{D}_B^{-T}\boldsymbol{D}_N^T\boldsymbol{y}_N) + \boldsymbol{b}_N^T\boldsymbol{y}_N \\ &=& \boldsymbol{b}_B^T\boldsymbol{D}_B^{-T}\boldsymbol{f} + (\boldsymbol{b}_N^T - \boldsymbol{b}_B^T\boldsymbol{D}_B^{-T}\boldsymbol{D}_N^T)\boldsymbol{y}_N \\ \mathbb{S}^n_+ &\ni& \boldsymbol{Z} \\ &=& \boldsymbol{C} - \mathcal{A}_B^*\boldsymbol{y}_B - \mathcal{A}_N^*\boldsymbol{y}_N \\ &=& \boldsymbol{C} - \mathcal{A}_B^*(\boldsymbol{D}_B^{-T}\boldsymbol{f} - \boldsymbol{D}_B^{-T}\boldsymbol{D}_N^T\boldsymbol{y}_N) - \mathcal{A}_N^*\boldsymbol{y}_N \\ &=& (\boldsymbol{C} - \mathcal{A}_B^*\boldsymbol{D}_B^{-T}\boldsymbol{f}) - \sum_{i \in N} \left(\boldsymbol{A}_i - \mathcal{A}_B^*(\boldsymbol{D}_B^{-T}\boldsymbol{d}_i^T)\right) y_i. \end{array}$$

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

$$\begin{array}{lll} \text{(P3)} & \text{minimize} & \widetilde{b}_0 + \widetilde{\boldsymbol{C}} \bullet \boldsymbol{X} \\ & \text{subject to} & \widetilde{\mathcal{A}}\boldsymbol{X} = \tilde{\boldsymbol{b}} \ \text{ and } \boldsymbol{X} \in \mathbb{S}^n_+. \\ \text{(D3)} & \text{maximize} & \widetilde{b}_0 + \tilde{\boldsymbol{b}}^T \boldsymbol{y}_N \\ & \text{subject to} & \widetilde{\boldsymbol{C}} - \widetilde{\mathcal{A}}^* \boldsymbol{y}_N = \boldsymbol{Z} \text{ and } \boldsymbol{Z} \in \mathbb{S}^n_+. \end{array}$$

where

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

Let $\boldsymbol{X}^*$ be an optimal solution of (P3), and $(\boldsymbol{y}_N^*, \boldsymbol{Z}^*)$ an optimal solution of (D3).

Then optimal solutions of (P1) and (D1) are obtained by $(\boldsymbol{X}, \boldsymbol{z}) = (\boldsymbol{X}^*, \boldsymbol{D}_B^{-1}\boldsymbol{b}_B - \boldsymbol{D}_B^{-1}\mathcal{A}_B\boldsymbol{X}^*)$ and $(\boldsymbol{y}_B, \boldsymbol{y}_N, \boldsymbol{Z}) = (\boldsymbol{D}_B^{-T}(\boldsymbol{f} - \boldsymbol{D}_N^T\boldsymbol{y}_N^*), \boldsymbol{y}_N^*, \boldsymbol{Z}^*)$, respectively.

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 [16], and T_{new} that for solving the SDP (P3) by SDPA. These are estimated as follows:

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

where k is the number of iterations of SDPA. T_{split} and the first term in T_{new} are for all computation of SDPA to solve the SDPs (P2) and (P3), respectively. See [5]. The second term $O(p^2m)$ in T_{new} is for the QR decomposition of $\boldsymbol{D}^T$ to choose p linearly independent vectors $\boldsymbol{d}_i^T$ $(i \in B)$. The third term $O((m-p)(p^2 + pn^2))$ is for computing $\widetilde{\boldsymbol{C}}$ and $\widetilde{\mathcal{A}}$.

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 increase of computation 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 faster than solving the SDP (P2).

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

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

3.2 Greedy heuristic for choosing a basis 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 $\widetilde{\boldsymbol{A}}_i$ of the converted SDP (P3) is computed as $\widetilde{\boldsymbol{A}}_i = \boldsymbol{A}_i - \mathcal{A}_B^*(\boldsymbol{D}_B^{-T}\boldsymbol{d}_i^T)$ $(i \in N)$. Hence the sparsity of the matrix $\widetilde{\boldsymbol{A}}_i$ $(i \in N)$ is the same with the aggregate sparsity of the matrix $\boldsymbol{A}_i$ and the p matrices $\boldsymbol{A}_k$ $(k \in B)$. If all of the data matrices $\boldsymbol{A}_j$ $(j = 1, 2, \ldots, m)$ are fully dense, each matrix $\widetilde{\boldsymbol{A}}_i$ $(i \in N)$ is also fully dense. If the data matrices $\boldsymbol{A}_j$ $(j = 1, 2, \ldots, m)$ are sparse and have different sparsity patterns, however, the sparsity of the matrix $\widetilde{\boldsymbol{A}}_i$ becomes worse than the matrix $\boldsymbol{A}_i$ $(i \in N)$. Moreover, the sparsity of the matrix $\widetilde{\boldsymbol{A}}_i$ $(i \in N)$ depends on how we choose p column vectors $\boldsymbol{d}_i^T$ $(i \in B)$ from $\boldsymbol{D}^T$. Many SDP software packages exploit sparsity of a problem to reduce the computation 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 [4, 6, 10]. In order to maintain the sparsity of the original SDP (P1) as much as possible, we need to choose p column vectors $\boldsymbol{d}_i^T$ $(i \in B)$ from $\boldsymbol{D}^T$ so that the density of the data matrices $\widetilde{\boldsymbol{A}}_i = \boldsymbol{A}_i - \mathcal{A}_B^*(\boldsymbol{D}_B^{-T}\boldsymbol{d}_i^T)$ $(i \in N)$ of the converted SDP (P3) is minimized. However, choosing such column vectors is very hard. Therefore we use the following heuristic method. Let $E = \{1, 2, \ldots, m\}$, the set of indices of row vectors $\boldsymbol{D}$ or the set of indices of column vectors of $\boldsymbol{D}^T$, and $\psi \subset 2^E$ the family of subsets I of E such that $\boldsymbol{d}_i^T$ $(i \in I)$ are linearly independent. Then the system (E, ψ) forms a matric matroid [1]. We call each $I \in \psi$ an independent set. We associate a weight $w_i =$ "the number of nonzero elements of the matrix $\boldsymbol{A}_i$" to each element $i \in E$, and we define the weight $w(S)$ of each subset S of E as the sum of the weights of its elements $i \in S$; $w(S) = \sum_{i \in S} w_i$. Then, we consider the matroid optimization problem: Find a maximal independent set of the system (E, ψ) with the minimum weight. This problem is known to be solved by a greedy algorithm. See [1]. As a solution of this matroid optimization problem, we have a $B \in \psi$ such that $\boldsymbol{d}_i^T \in \mathbb{R}^p$ $(i \in B)$ forms a basis of $\mathbb{R}^p$ and that $\boldsymbol{A}_i$ $(i \in B)$ are relatively sparse among the set of matrices $\boldsymbol{A}_i$ $(i = 1, 2, \ldots, m)$; hence the data matrices $\widetilde{\boldsymbol{A}}_i = \boldsymbol{A}_i - \mathcal{A}_B^*(\boldsymbol{D}_B^{-T}\boldsymbol{d}_i^T)$ $(i \in N)$ in the resulting SDP (P3) become sparse.

3.3 Numerical evaluation of the greedy heuristic

In order to evaluate the above greedy heuristic algorithm, we executed numerical experiments. 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 $\boldsymbol{D}^T$ with column pivoting by the *LAPACK* [2] function

`dgeqp3`. As an output of this function, we have a pivoting vector P. By using this pivoting information, we choose the basis indices 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 $\boldsymbol{X}$ 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 has 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 $\boldsymbol{A}_i$ $(i = 1, 2, \ldots, m)$ becomes nearly $\frac{q}{i^2} \times n^2$. As a result, we have a problem in which the sparsities of the data matrices $\boldsymbol{A}_i$ $(i = 1, 2, \cdots, m)$ are different each other. Each nonzero entry in the matrices was generated by a uniform random number. Moreover, the matrix $\boldsymbol{D}$ was generated so that the number of nonzero elements becomes nearly $q \times n^2$ and each entry was generated by a uniform random number which took value between 0 and m.

The matrix $\boldsymbol{C}$ was taken to be the identity matrix $\boldsymbol{I}$ and each entry of the vectors $\boldsymbol{f}$ and $\boldsymbol{b}$ was generated by a uniform random number which took a value between 0 and 1. By generating $\boldsymbol{D}$, $\boldsymbol{C}$ and $\boldsymbol{f}$ in this way, there exists a vector $\boldsymbol{y}$ with which $\boldsymbol{C} - \mathcal{A}^*\boldsymbol{y}$ becomes positive semidefinite and the relation $\boldsymbol{D}^T\boldsymbol{y} = \boldsymbol{f}$ holds in the dual problem (D1). That is, the dual problem (D1) surely has a feasible solution.

In our numerical experiments, we ran SDPA with the default setting. When the primal infeasibility

$$\max\left\{\left|\boldsymbol{A}_i \bullet \boldsymbol{X}^k - b_i\right| : i = 1, 2, \ldots, m\right\}, \tag{1}$$

the dual infeasibility

$$\max\left\{\left|[\sum_{i=1}^{m} \boldsymbol{A}_i y_i^k + \boldsymbol{Z}^k - \boldsymbol{C}]_{p,q}\right| : p, q = 1, 2, \ldots, n\right\}, \tag{2}$$

and the relative gap

$$\frac{\left|\sum_{i=1}^{m} b_i y_i^k - \boldsymbol{C} \bullet \boldsymbol{X}^k\right|}{\max\left\{(\left|\sum_{i=1}^{m} b_i y_i^k\right| + \left|\boldsymbol{C} \bullet \boldsymbol{X}^k\right|)/2.0, 1.0\right\}} \tag{3}$$

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

Table 2 shows the computation time per iteration of SDPA applied to the converted SDP (P3) with and without the above heuristic algorithm for choosing the basis indices B. As these results show, the heuristic algorithm is effective to maintain the sparsity of the original SDP (P1) in the converted SDP (P3). In computational results reported in the next section, we used 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 could be more stably solved because

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

the conversion yields no degeneracy. In this section, we present computational results for evaluating these two advantages of the new conversion method. The numerical experiment was done on Pentium IV (Xeon) 2.4GHz with 6GB memory. In tables in this section, we use following notations. An "Yes" in "opt." column denotes that the problem was solved; that is, SDPA found an approximate optimal solution whose primal infeasibility, dual infeasibility and relative gap are smaller than 1.0E–7. A "No" in "opt." column denotes that the problem was not solved. A figure in "CPU" column denotes the total computation time (second) of the primal-dual interior-point method, a figure in "iter." denotes the number of iterations of the primal-dual interior-point method, a figure in "C/i" or "CPU/iter." column denotes the computation time (second) per iteration of the primal-dual interior-point method, and a figure in "conv." denotes the time (second) to convert the original SDP (P1) into the converted SDP (P3). In case that the problem was not solved, we show the figures in "CPU" column, "iter." column and "C/i" column in the brackets. Moreover, a figure in "p.err" column denotes the primal infeasibility defined by (1), a figure in "d.err" column the dual infeasibility defined by (2), and a figure in "rel.gap" column the relative gap defined by (3) in the last iteration of SDPA.

4.1 Stability

In order to evaluate the stability of the new conversion method, 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 randomly generated a matrix $\boldsymbol{D} \in \mathbb{R}^{m \times p}$ and a vector $\boldsymbol{f} \in \mathbb{R}^{p}$ and add them to this SDP (P0), where p was taken to be $m/2$. In this way, we have a set of SDPs having free variables. Table 3 and 4 show the results on SDPA applied to the standard form SDP (P2) and the converted SDP (P3) for solving these problems. SDPA applied to the SDP (P2) was not able to solve any of the problems except `qap5_eq`, `qap6_eq` and `qap7_eq` because of some numerical difficulties. In contrast to this, SDPA applied to the converted SDP (P3) was able to solve all of the problems; it found an approximate optimal solution for each problem whose primal infeasibility, dual

infeasibility and relative gap are smaller than 1.0E–7. That is, the new conversion method with SDPA shows a good performance for solving SDPs having free variables stably.

4.2 Reduction of running time

The other advantage of the new conversion method is that the resulting standard form SDP (P3) has a smaller size than the original SDP (P1). Hence solving the converted SDP (P3) is faster than solving the original SDP (P1) if their data matrices have similar sparsity. In order to evaluate this effect, we used two sets of test problems: randomly generated fully dense problems and norm minimization problems with free variables.

For fully dense problems, we fixed the size n of the variable matrix $\boldsymbol{X}$ 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. Moreover, the matrices $\boldsymbol{A}_i$ $(i = 1, 2, \ldots, m)$ are generated by uniform random numbers. Note that both of the original SDP (P1) and the converted SDP (P3) are fully dense. Table 5 shows the results of SDPA applied to the SDP (P2) and the results of SDPA applied to the converted SDP (P3). Note that SDPA applied to the SDP (P3) was able to solve all of the problems, whereas SDPA applied to the SDP (P2) was not able to solve any of the problems. For these problems, as the p value becomes closer to the the m value, the computation time of SDPA applied to the converted SDP (P3) becomes shorter.

In addition to the fully dense problems, we generated norm minimization problems with free variables. The norm minimization problem is formulated as a standard form SDP (P0) [7]. We firstly generated the norm minimization problem in the standard form SDP (P0) whose data matrices and vectors are randomly generated. Secondly we randomly generated a matrix $\boldsymbol{D}$ and a vector $\boldsymbol{f}$ and add them to the standard form SDP(P0). Then we have an SDP (P1) having free variables. This problem has a favourable sparsity pattern in the sense that the original SDP (P1) and the converted SDP (P3) have similar sparsity. Table 6 shows the results for solving these problems. As the p value becomes closer to the m value, the computation time of SDPA applied to the converted SDP (P3) becomes shorter.

4.3 Conversion as a preprocessing of the primal-dual interior-point method

We can consider the new conversion method as a preprocessing for an SDP software package which implements the primal-dual interior-point method. That is, we convert an SDP having free variables into the standard form SDP (P3) before we apply the SDP software package. The resulting SDP (P3) is in the standard form so that it can be solved by any SDP software package. In the previous section, we applied SDPA to solve the standard form SDP (P3). Here we show numerical results on SeDuMi and SDPT3 applied to the converted SDP (P3). We ran SeDuMi version 1.05 and SDPT3 version 3.02 with the default settings. An "Yes" in "opt." column in Table 7 and 8 denotes that SeDuMi/SDPT3 terminated with the termination code which indicates that the problem was solved. A "No" in "opt." column in Table 7 and 8 denotes that SeDuMi/SDPT3 terminated with the termination code which indicates that the problem was not solved because of some numerical difficulties.

SeDuMi can deal with free variables so that it can directly solve the original SDP (P1). Table 7 shows numerical results on SeDuMi applied to the original SDP (P1) and applied to

Table 3: Results of SDPA, SDPLIB with free var.

	SDPA (P2)				SDPA (P3)				
name	opt.	CPU	iter.	C/i	opt	CPU	iter.	C/i	conv.
theta1_eq	No.	(0.10)	(11)	(0.01)	Yes	0.26	15	0.02	0.08
theta2_eq	No	(4.22)	(10)	(0.42)	Yes	8.51	16	0.53	2.70
theta3_eq	No	(38.36)	(9)	(4.26)	Yes	84.38	17	4.96	22.75
theta4_eq	No	(185.81)	(9)	(20.65)	Yes	409.89	18	22.77	120.77
theta5_eq	No	(576.28)	(8)	(72.04)	Yes	2209.10	19	116.27	449.29
qap5_eq	Yes	0.14	14	0.01	Yes	0.27	14	0.02	0.150
qap6_eq	Yes	1.18	15	0.08	Yes	0.47	15	0.03	0.970
qap7_eq	Yes	5.65	15	0.38	Yes	2.25	15	0.15	3.10
qap8_eq	No	(8.77)	(12)	(0.73)	Yes	7.94	16	0.5	4.72
qap9_eq	No	(19.07)	(11)	(1.73)	Yes	18.67	16	1.17	19.89
qap10_eq	No	(57.50)	(11)	(5.23)	Yes	85.83	16	5.36	33.40
mcp124-1_eq	No	(0.56)	(12)	(0.05)	Yes	1.45	17	0.09	0.11
mcp124-2_eq	No	(0.55)	(12)	(0.05)	Yes	1.42	17	0.08	0.12
mcp124-3_eq	No	(0.55)	(12)	(0.05)	Yes	1.32	16	0.08	0.11
mcp124-4_eq	No	(0.49)	(11)	(0.04)	Yes	1.29	15	0.09	0.13
mcp250-1_eq	No	(3.33)	(13)	(0.26)	Yes	59.06	21	2.81	0.44
mcp250-2_eq	No	(3.17)	(12)	(0.26)	Yes	56.25	19	2.96	0.50
mcp250-3_eq	No	(3.12)	(12)	(0.26)	Yes	49.66	18	2.76	0.44
mcp250-4_eq	No	(2.99)	(11)	(0.27)	Yes	53.50	19	2.82	0.50
gpp124-1_eq	No	(0.62)	(12)	(0.05)	Yes	1.75	20	0.09	0.53
gpp124-2_eq	No	(0.63)	(12)	(0.05)	Yes	1.67	19	0.09	0.50
gpp124-3_eq	No	(0.59)	(12)	(0.05)	Yes	1.64	18	0.09	0.56
gpp124-4_eq	No	(0.56)	(11)	(0.05)	Yes	1.52	17	0.09	0.54
gpp250-1_eq	No	(3.56)	(12)	(0.30)	Yes	64.73	21	3.08	4.93
gpp250-2_eq	No	(3.58)	(12)	(0.30)	Yes	49.80	17	2.93	4.74
gpp250-3_eq	No	(3.23)	(11)	(0.29)	Yes	51.39	17	3.02	4.94
gpp250-4_eq	No	(3.17)	(11)	(0.29)	Yes	50.22	17	2.95	5.05

Table 4: Results of SDPA, SDPLIB with free var.

	SDPA (P2)			SDPA (P3)		
name	p.err	d.err	rel. gap	p.err	d.err	rel. gap
theta1_eq	6.76e-5	2.42e-8	1.12e-3	4.97e-9	5.50e-11	3.19e-8
theta2_eq	3.79e-4	4.98e-7	1.74e-2	2.69e-13	5.17e-13	2.43e-8
theta3_eq	2.89e-4	4.90e-6	1.38e-1	1.16e-12	2.17e-12	4.15e-8
theta4_eq	2.10e-4	3.66e-5	7.19e-2	1.73e-11	1.14e-11	4.22e-8
theta5_eq	5.12e-5	9.19e-5	7.41e-1	8.44e-12	1.13e-11	1.06e-8
qap5_eq	6.18e-10	1.82e-8	3.59e-8	1.40e-11	3.09e-12	4.36e-8
qap6_eq	1.99e-10	5.62e-8	1.66e-8	4.52e-11	1.95e-12	5.57e-8
qap7_eq	2.14e-10	5.70e-8	1.30e-8	4.69e-11	4.30e-12	5.01e-8
qap8_eq	8.91e-5	2.87e-8	4.70e-5	1.16e-10	1.21e-11	2.37e-8
qap9_eq	9.86e-6	3.03e-7	2.06e-4	1.54e-10	2.45e-11	3.14e-8
qap10_eq	1.06e-4	2.24e-7	1.66e-4	5.38e-10	3.50e-11	8.26e-8
mcp124-1_eq	3.01e-3	1.37e-5	2.01e-3	3.81e-10	2.98e-12	5.83e-8
mcp124-2_eq	2.99e-3	4.43e-6	2.23e-3	1.06e-10	5.74e-12	2.41e-8
mcp124-3_eq	2.97e-3	4.78e-6	7.75e-4	3.06e-10	5.93e-12	4.20e-8
mcp124-4_eq	1.42e-3	1.93e-5	1.53e-3	2.15e-10	1.00e-11	7.19e-8
mcp250-1_eq	2.58e-3	3.42e-5	5.30e-3	9.85e-11	9.98e-13	5.05e-8
mcp250-2_eq	2.83e-3	2.57e-5	7.09e-3	1.27e-11	1.05e-12	9.30e-8
mcp250-3_eq	1.05e-2	1.36e-5	3.19e-3	6.29e-12	1.26e-12	8.82e-8
mcp250-4_eq	2.11e-3	6.60e-5	8.36e-3	1.51e-11	1.03e-12	1.28e-8
gpp124-1_eq	1.40e-2	1.02e-5	5.10e-3	3.35e-12	1.70e-12	6.40e-8
gpp124-2_eq	9.05e-3	4.66e-6	5.88e-3	5.99e-12	2.19e-12	1.21e-8
gpp124-3_eq	1.41e-2	3.25e-6	7.48e-3	1.49e-11	2.15e-12	3.62e-8
gpp124-4_eq	2.05e-3	2.27e-5	2.46e-2	3.05e-12	1.43e-12	5.93e-8
gpp250-1_eq	7.45e-3	1.71e-5	7.47e-3	5.06e-8	3.74e-8	1.54e-8
gpp250-2_eq	4.23e-3	1.40e-5	8.46e-3	1.48e-9	2.56e-13	7.44e-8
gpp250-3_eq	2.55e-3	3.34e-5	4.73e-2	2.75e-9	3.34e-13	3.73e-8
gpp250-4_eq	3.71e-3	2.86e-5	3.69e-2	2.15e-9	4.38e-13	5.66e-8

Table 5: Reduction of the time (fully dense problems)

			SDPA (P2)			SDPA (P3)		
m	n	p	CPU	iter.	CPU/iter.	CPU	iter.	CPU/iter.
100	200	10	(30.43)	(11)	(2.77)	36.36	16	2.27
100	200	30	(28.12)	(11)	(2.56)	26.53	16	1.66
100	200	50	(25.64)	(10)	(2.56)	18.11	16	1.13
100	200	70	(25.51)	(10)	(2.55)	10.67	16	0.67
100	200	90	(28.09)	(11)	(2.55)	4.82	17	0.28

Table 6: SDPA norm min. problems with free var.

			SDPA (P2)			SDPA (P3)		
m	n	p	CPU	iter.	CPU/iter.	CPU	iter.	CPU/iter.
500	500	100	(890.13)	(7)	(127.16)	1334.51	16	83.41
500	500	200	(1002.27)	(8)	(125.28)	936.48	16	58.53
500	500	300	(1183.63)	(9)	(131.51)	819.40	17	48.20
500	500	400	(1188.12)	(9)	(132.01)	526.19	17	30.95

the converted SDP (P3). From this table, we observe that SeDuMi applied to the converted SDP (P3) was able to solve 11 problems that SeDuMi applied to the original SDP (P1) was not able to solve. That is, SeDuMi applied to the converted SDP (P3) is more stable than SeDuMi applied to the original SDP (P1). Moreover, the computation time per iteration of SeDuMi applied to the original SDP (P1) is similar to that of SeDuMi applied to the converted SDP (P3) for all of the problems.

SDPT3 can also deal with free variables so that it can directly solve the original SDP (P1). Table 8 shows numerical results on SDPT3 applied to the original SDP (P1) and the converted SDP (P3). SDPT3 applied to both the original SDP (P1) and applied to the converted SDP (P3) were able to solve all of the problems stably. The computation time per iteration of SDPT3 applied to the converted SDP (P3) is larger than the computation time per iteration of SDPT3 applied to the original SDP (P1). The reason of this is that the sparsity of the converted SDP (P3) is worse than the original SDP (P1). The worsening of sparsity increased the computation time of SDPT3. However, this increase of computation time is not so large. More specifically, the computation time per iteration of SDPT3 applied to the converted SDP (P3) is at most four times larger than that of SDPT3 applied to the original SDP (P1).

5 Concluding Remarks

In this paper, we described 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 could be more stably solved. Through the numerical experiments, we showed these two advantages. The conversion can be used as a preprocessing so that it can be used with any SDP software package. As a topic of further study, it would be worthwhile developing more effective algorithm to choose a basis B than the greedy heuristic algorithm described in section 3.2 so that the resulting SDP (P3) becomes sparser.

References

[1] R. Ahuja, T. Magnanti and J. Orlin, *Network Flows – theory, algorithms, and applications*, (Prentice Hall, 1993).

Table 7: Results of SeDuMi, SDPLIB with free var.

	SeDuMi (P1)				SeDuMi (P3)				
name	opt.	CPU	iter.	C/i	opt.	CPU	iter.	C/i	conv.
theta1_eq	No	(0.8)	(13)	(0.06)	Yes	0.9	15	0.60	0.08
theta2_eq	No	(6.2)	(11)	(0.56)	Yes	5.9	15	0.39	2.70
theta3_eq	No	(54.0)	(11)	(4.91)	Yes	57.0	15	3.8	22.75
theta4_eq	No	(231.1)	(10)	(23.11)	Yes	292.7	15	19.51	120.77
theta5_eq	No	(960.7)	(11)	(87.34)	Yes	726.8	14	51.91	449.29
qap5_eq	No	(0.6)	(13)	(0.05)	Yes	0.4	15	0.03	0.150
qap6_eq	No	(1.6)	(14)	(0.11)	Yes	0.9	15	0.06	0.970
qap7_eq	No	(2.6)	(13)	(0.20)	Yes	3.8	15	0.25	3.10
qap8_eq	No	(6.3)	(12)	(0.53)	Yes	7.1	15	0.47	4.72
qap9_eq	No	(16.9)	(11)	(1.54)	Yes	19.9	15	1.33	19.89
qap10_eq	No	(41.9)	(11)	(3.81)	No	(85.83)	(16)	(5.36)	33.40
mcp124-1_eq	Yes	3.1	15	0.21	Yes	3.3	17	0.19	0.11
mcp124-2_eq	Yes	3.1	15	0.21	Yes	4.6	18	0.26	0.12
mcp124-3_eq	Yes	3.2	15	0.21	Yes	3.3	17	0.19	0.13
mcp124-4_eq	Yes	3.2	15	0.21	Yes	3.8	18	0.21	0.12
mcp250-1_eq	Yes	24.3	16	1.52	Yes	31.4	18	1.74	0.47
mcp250-2_eq	Yes	23.5	15	1.57	Yes	29.6	18	1.64	0.42
mcp250-3_eq	Yes	23.7	15	1.58	Yes	26.9	17	1.58	0.48
mcp250-4_eq	Yes	23.7	15	1.58	Yes	25.1	16	1.57	0.54
gpp124-1_eq	Yes	5.0	17	0.29	Yes	4.5	19	0.24	0.53
gpp124-2_eq	Yes	4.9	18	0.27	Yes	4.3	19	0.23	0.50
gpp124-3_eq	Yes	4.5	17	0.26	Yes	3.6	17	0.21	0.56
gpp124-4_eq	Yes	5.0	17	0.29	Yes	4.1	18	0.23	0.54
gpp250-1_eq	No	(47.3)	(21)	(2.25)	No	(63.6)	(20)	(3.18)	4.93
gpp250-2_eq	No	(59.1)	(21)	(2.81)	No	(41.5)	(18)	(2.31)	4.74
gpp250-3_eq	No	(61.10)	(20)	(3.06)	No	(46.5)	(21)	(2.21)	4.94
gpp250-4_eq	No	(77.7)	(20)	(3.89)	No	(66.4)	(24)	(2.77)	5.05

Table 8: Results of SDPT3, SDPLIB with free var.

	SDPT3 (P1)				SDPT3 (P3)				
name	opt.	CPU	iter.	C/i	opt.	CPU	iter.	C/i	conv.
theta1_eq	Yes	0.6	13	0.05	Yes	1.5	12	0.13	0.08
theta2_eq	Yes	8.3	15	0.55	Yes	20.6	15	1.37	2.70
theta3_eq	Yes	63.1	16	3.94	Yes	97.0	14	6.93	22.75
theta4_eq	Yes	296.2	17	17.42	Yes	421.1	17	24.77	120.77
theta5_eq	Yes	950.8	16	59.43	Yes	1141.1	16	71.32	449.29
qap5_eq	Yes	0.9	12	0.08	Yes	1.2	12	0.1	0.150
qap6_eq	Yes	1.6	13	0.12	Yes	2.7	12	0.23	0.970
qap7_eq	Yes	3.8	14	0.27	Yes	6.4	12	0.53	3.10
qap8_eq	Yes	9.4	15	0.63	Yes	15.1	12	1.26	4.72
qap9_eq	Yes	22.2	15	1.48	Yes	35.4	14	2.53	19.89
qap10_eq	Yes	49.6	15	3.31	Yes	60.6	12	5.05	33.40
mcp124-1_eq	Yes	2.1	14	0.15	Yes	6.9	20	0.35	0.11
mcp124-2_eq	Yes	2.0	13	0.15	Yes	5.7	19	0.30	0.12
mcp124-3_eq	Yes	2.2	14	0.16	Yes	5.9	19	0.31	0.13
mcp124-4_eq	Yes	2.2	13	0.17	Yes	6.0	19	0.32	0.12
mcp250-1_eq	Yes	6.9	16	0.43	Yes	34.8	20	1.74	0.47
mcp250-2_eq	Yes	7.4	15	0.49	Yes	35.8	19	1.88	0.42
mcp250-3_eq	Yes	8.0	14	0.57	Yes	26.3	18	1.46	0.48
mcp250-4_eq	Yes	7.9	14	0.56	Yes	27.5	18	1.53	0.54
gpp124-1_eq	Yes	2.5	15	0.17	Yes	6.1	19	0.32	0.53
gpp124-2_eq	Yes	2.4	14	0.17	Yes	5.3	17	0.31	0.50
gpp124-3_eq	Yes	2.4	14	0.17	Yes	5.0	16	0.31	0.56
gpp124-4_eq	Yes	2.2	13	0.17	Yes	4.7	15	0.31	0.54
gpp250-1_eq	Yes	14.1	20	0.71	Yes	42.0	25	1.68	4.93
gpp250-2_eq	Yes	12.2	17	0.72	Yes	37.0	22	1.68	4.74
gpp250-3_eq	Yes	11.8	16	0.74	Yes	35.4	21	1.69	4.94
gpp250-4_eq	Yes	12.2	17	0.72	Yes	35.1	21	1.67	5.05

[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, Society for Industrial and Applied Mathematics 1999 Philadelphia, PA, ISBN 0-89871-447-8 (paperback).

[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. Kojima and K. Nakata, Exploiting sparsity in primal-dual interior-point methods for semidefinite programming, *Mathematical Programming*, 79 (1997) 235–253.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] R. D. C. Monteiro, Primal-dual path-following algorithms for semidefinite programming, *SIAM Journal on Optimization*, 7 (1997) 663–678.

[10] 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.

[11] 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.

[12] Yu. E. Nesterov and M. J. Todd, Primal-dual interior-point methods for self-scaled cones, *SIAM Journal on Optimization*, 8 (1998) 324-364.

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

[14] K. C. Toh, M. J. Todd and R. H. Tütüncü, SDPT3 — a MATLAB software package for semidefinite programming, version 1.3, *Optimization Methods and Software*, 11 & 12 (1999) 545–581. Available at http://www.math.nus.edu.sg/˜mattohkc.

[15] H. Waki, S. Kim, M. Kojima and M. Muramatsu, Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity, Research Report B-411, Deptartment of Mathematical and Computing Sciences, Tokyo Institute of Technology, Meguro, Tokyo 152-8552 ( ).

[16] 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.

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