Implementation and Evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0)

Makoto Yamashtia[†], Katsuki Fujisawa[*] and Masakazu Kojima[‡]

Abstract. The SDPA (SemiDefinite Programming Algorithm) is a software package for solving general SDPs (SemiDefinite Programs). It is written in C++ with the help of *LAPACK* for numerical linear algebra for dense matrix computation. The purpose of this paper is to present a brief description of the latest version of the SDPA and its high performance for large scale problems through numerical experiment and comparison with some other major software packages for general SDPs.

Key words. Semidefinie Program, Interior-Point Method, Optimization, Software, Numerical Experiment.

† Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan
e-mail: *Makoto.Yamashita@is.titech.ac.jp*

∗ Department of Mathematical Sciences
Tokyo Denki University
Hatoyama, Saitama 350-0394, Japan
e-mail: *fujisawa@r.dendai.ac.jp*

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

1 Introduction

SDP (Semidefinite Program) is an extension of LP (Linear Program) in the Euclidean space to the space of symmetric matrices. We start from an LMI (Linear Matrix Inequality) as an example of SDPs arising from the system and control theory [4]. Let $\mathbb{S}^n$ be the set of $n \times n$ symmetric matrices, and let $\boldsymbol{A}_i \in \mathbb{S}^n$ $(0 \leq i \leq p)$. We use the notation $\boldsymbol{X} \succeq \boldsymbol{O}$ to denote that $\boldsymbol{X} \in \mathbb{S}^n$ is positive semidefinite. An LMI is defined as

$$\boldsymbol{A}(\boldsymbol{x}) = \boldsymbol{A}_0 + \sum_{i=1}^{p} x_i \boldsymbol{A}_i \succeq \boldsymbol{O},$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)$ is a vector variable in $\mathbb{R}^p$. We want to find an $\boldsymbol{x} \in \mathbb{R}^p$ such that $\boldsymbol{A}(\boldsymbol{x}) \succeq \boldsymbol{O}$ or detect that $\boldsymbol{A}(\boldsymbol{x})$ is not positive semidefinite for any $\boldsymbol{x} \in \mathbb{R}^p$. Introducing an auxiliary variable λ, we transform the LMI into the following SDP.

$$\text{minimize} \quad \lambda \quad \text{subject to} \quad \boldsymbol{X} = \boldsymbol{A}_0 + \sum_{i=1}^{p} x_i \boldsymbol{A}_i + \lambda \boldsymbol{I}, \quad \boldsymbol{X} \succeq \boldsymbol{O}.$$

Here $\boldsymbol{I}$ denotes the $n \times n$ identity matrix. In this minimization problem, we have a linear objective function λ over a linear constraint $\boldsymbol{X} = \boldsymbol{A}_0 + \sum_{i=1}^{p} x_i \boldsymbol{A}_i + \lambda \boldsymbol{I}$ and a positive semidefinite constraint $\boldsymbol{X} \succeq \boldsymbol{O}$. Suppose that the above SDP has a feasible solution $(\bar{\boldsymbol{x}}, \bar{\lambda})$ with a nonpositive objective function value $\lambda = \bar{\lambda}$. Then $\boldsymbol{x} = \bar{\boldsymbol{x}}$ satisfies the LMI. On the other hand, if the optimal value of the SDP is positive, then we know that the LMI has no solution.

Now we introduce a standard form of SDP $\mathcal{P}$, and its dual $\mathcal{D}$.

$$\left.\begin{array}{llllll} \mathcal{P}: & \text{minimize} & \displaystyle\sum_{i=1}^{m} c_i x_i & \text{subject to} & \boldsymbol{X} = \displaystyle\sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0, & \boldsymbol{X} \succeq \boldsymbol{O},\ \boldsymbol{X} \in \mathbb{S}^n. \\ \mathcal{D}: & \text{maximize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} & \text{subject to} & \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \ (i = 1, 2, \ldots, m), & \boldsymbol{Y} \succeq \boldsymbol{O},\ \boldsymbol{Y} \in \mathbb{S}^n. \end{array}\right\} \quad (1)$$

Here $\boldsymbol{U} \bullet \boldsymbol{V}$ denotes the inner product of $\boldsymbol{U}$ and $\boldsymbol{V}$ in $\mathbb{S}^n$, *i.e.*, $\boldsymbol{U} \bullet \boldsymbol{V} = \sum_{i=1}^{n} \sum_{j=1}^{n} \boldsymbol{U}_{ij} \boldsymbol{V}_{ij}$. We call $(\boldsymbol{x}, \boldsymbol{X})$ a primal feasible solution of the SDP if $(\boldsymbol{x}, \boldsymbol{X})$ satisfies the constraints of $\mathcal{P}$, and $(\boldsymbol{x}, \boldsymbol{X})$ a primal interior feasible solution if $\boldsymbol{X} \succ \boldsymbol{O}$ in addition to the feasibility. If a primal feasible solution $(\boldsymbol{x}, \boldsymbol{X})$ attains the minimum objective value $\sum_{i=1}^{m} c_i x_i$ of $\mathcal{P}$ among all primal feasible solutions, we call $(\boldsymbol{x}, \boldsymbol{X})$ a primal optimal solution. We define a dual feasible solution, a dual interior feasible solution, and dual optimal solution similarly. We say that $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is a feasible solution (an interior feasible solution, or an optimal solution, respectively) if $(\boldsymbol{x}, \boldsymbol{X})$ is a primal feasible solution (a primal interior feasible solution, or a primal optimal solution, respectively) and $\boldsymbol{Y}$ is a dual feasible solution (a dual interior-feasible solution or a dual optimal solution, respectively).

SDP is not only an extension of LP but also includes convex quadratic optimization problems and some other convex optimization problems. It has a lot of applications in various fields such as combinatorial optimization [9], control theory [4], robust optimization [3, 24] and quantum chemistry [15, 16]. See [20, 23, 24] for a survey on SDPs and the papers in their references.

The PDIPA (primal-dual interior-point algorithm) [10, 12, 14, 17] is known as the most powerful and practical numerical method for solving general SDPs. The method is an extension of the PDIPA [11, 19] developed for LPs. The SDPA (SemiDefinete Programming Algorithm) presented in this paper is a PDIPA software package for general SDPs based on the paper [7, 12]. Besides the SDPA, several software packages for solving general SDPs, DSDP [2], CSDP [5], SeDuMi [18] and SDPT3 [21] have been released so far. They are available at the Web site http://www-neos.mcs.anl.gov/neos/server-solvers.html. The main features of the SDPA are:

- It exploits the sparsity of data matrices [7] to save computational time, and solves large scale SDPs efficiently.
- The SDPA is written in C++ language, so that one can call its routines from his own C++ program.

Some numerical results of the SDPA 4.0, an old version of the SDPA with its comparison to some other software packages including the SDPT3 and the CSDP were reported in the paper [8]. Since then, the performance of the SDPA and those software packages has been considerably improved. In particular, the SDPA 6.0 (the latest version at the moment of writing) incorporated *LAPACK* [1], and adopted one dimensional array data structure to store dense matrices so as to fit to dense matrix computation using *LAPACK*. These two new devices have made the SDPA much faster than the previous version (Version 5.01).

The main purpose of this paper is to present a high performance of the SDPA 6.0 in comparison to some major software packages, the SDPT3[21], the CSDP[5] and the SeDuMi[18]. Section 2 presents an algorithmic framework of the SDPA. Section 3 explains the usage of the SDPA shortly. Section 4 is devoted to some numerical results on the SDPA 6.0 and its comparison to the SDPT3, the CSDP, the SeDuMi and the SDPA 5.01 through randomly generated SDPs, large scale SDPs arising from quantum chemistry [15, 16] and SDPLIB[6] benchmark problems. For the first two types of problems, the SDPA 6.0 exhibited the highest performance among the software packages listed above. In particular, the SDPA 6.0 successfully solved large scale duals SDP $\mathcal{D}$ of the second type, having more than $m = 15000$ equality constraints, which all other software packages could not handle because of out of memory. We also observe in Section 4 that the SDPA 6.0 and the SDPT3 worked better on SDPLIB benchmark problems than the CSDP, the SeDuMi and the SDPA 5.01.

2 The Primal-Dual Interior-Point Algorithm of the SDPA

Suppose that the SDP (1) has an interior feasible solution. Then the KKT system (2) below gives a necessary and sufficient condition for $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ to be an optimal solution of the SDP (1).

$$\boldsymbol{X} = \textstyle\sum_{i=1}^m \boldsymbol{F}_i x_i - \boldsymbol{F}_0,\ \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \quad (i = 1, 2, \ldots, m),\ \boldsymbol{XY} = \boldsymbol{O},\ \boldsymbol{X} \succeq \boldsymbol{O},\ \ \boldsymbol{Y} \succeq \boldsymbol{O} \tag{2}$$

Replacing the complementarity condition $\boldsymbol{XY} = \boldsymbol{O}$ by a perturbed complementarity condition $\boldsymbol{XY} = \mu \boldsymbol{I}$ with $\mu > 0$, we have

$$\boldsymbol{X} = \textstyle\sum_{i=1}^m \boldsymbol{F}_i x_i - \boldsymbol{F}_0,\ \ \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \quad (i = 1, 2, \ldots, m),\ \boldsymbol{XY} = \mu \boldsymbol{I},\ \ \boldsymbol{X} \succeq \boldsymbol{O},\ \ \boldsymbol{Y} \succeq \boldsymbol{O}. \tag{3}$$

It is known that for every $\mu > 0$ the perturbed KKT system (3) has a unique solution $(\boldsymbol{x}_\mu, \boldsymbol{X}_\mu, \boldsymbol{Y}_\mu)$, and that the set $\mathcal{C} = \{(\boldsymbol{x}_\mu, \boldsymbol{X}_\mu, \boldsymbol{Y}_\mu) : \mu > 0\}$ forms a smooth curve converging to a solution $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ of the KKT system (2), which corresponds to an optimal solution of the SDP (1), as $\mu \to 0$. We call $\mathcal{C}$ the central path. By construction, we know if $\mu > 0$ and $(\boldsymbol{x}_\mu, \boldsymbol{X}_\mu, \boldsymbol{Y}_\mu)$ is on the central path $\mathcal{C}$ then $\mu = \boldsymbol{X} \bullet \boldsymbol{Y}/n$.

Roughly speaking, the SDPA starts from a given initial point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ satisfying $\boldsymbol{X} \succ \boldsymbol{O}$ and $\boldsymbol{Y} \succ \boldsymbol{O}$ and numerically traces the central path $\mathcal{C}$. Letting $\mu = \beta(\boldsymbol{X} \bullet \boldsymbol{Y}/n)$, it chooses a target point $(\boldsymbol{x}_\mu, \boldsymbol{X}_\mu, \boldsymbol{Y}_\mu)$ on the central path $\mathcal{C}$ to move from the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$, where $\beta \in [0, 1]$. Then the SDPA computes a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ to approximate the point $(\boldsymbol{x}_\mu, \boldsymbol{X}_\mu, \boldsymbol{Y}_\mu)$, and updates the current point as $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \leftarrow (\boldsymbol{x} + \alpha_p d\boldsymbol{x}, \boldsymbol{X} + \alpha_p d\boldsymbol{X}, \boldsymbol{Y} + \alpha_d d\boldsymbol{Y})$, where α_p and α_d are primal and dual step lengths to keep $\boldsymbol{X} + \alpha_p d\boldsymbol{X}$ and $\boldsymbol{Y} + \alpha_d d\boldsymbol{Y}$ positive definite. The SDPA repeats this procedure until it attains an $(\bar{\epsilon}, \epsilon^*)$-approximate solution described below.

Suppose that $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \in \mathbb{R}^m \times \mathbb{S}^n \times \mathbb{S}^n$ satisfies $\boldsymbol{X} \succeq \boldsymbol{O}$ and $\boldsymbol{Y} \succeq \boldsymbol{O}$. The SDPA employs the quantities

$$\max \left\{ \left| [\boldsymbol{X} - \sum_{i=1}^m \boldsymbol{F}_i x_i + \boldsymbol{F}_0]_{pq} \right| \ : \ p, q = 1, 2, \ldots, n \right\} \quad \text{and} \tag{4}$$

$$\max \{ |\boldsymbol{F}_i \bullet \boldsymbol{Y} - c_i| \ : i = 1, 2, \ldots, m \} \tag{5}$$

as the primal feasibility error and the dual feasibility error, respectively. We refer to the maximum of the primal and the dual feasibility errors as the primal-dual feasibility error or simply the feasibility error. We say that $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is an $\bar{\epsilon}$-approximate feasible solution of the SDP (1) if the feasibility error is less than $\bar{\epsilon}$, where $\bar{\epsilon}$ is a parameter whose default value is set at 1.0×10^{-7} in the SDPA. Define the relative duality gap by

$$\frac{|objP - objD|}{\max\{(|objP| + |objD|)/2.0,\ 1.0\}} \tag{6}$$

Here $objP$ denotes the primal objective function value $\sum_{i=1}^m c_i x_i$, and $objD$ the dual objective function value $\boldsymbol{F}_0 \bullet \boldsymbol{Y}$. Let ϵ^* be a sufficiently small positive number; the default value of ϵ^* is set at 1.0×10^{-7}

in the SDPA. We say that $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is an $(\bar{\epsilon}, \epsilon^*)$-approximate optimal solution of the SDP (1) if it is an $\bar{\epsilon}$-approximate feasible solution and the relative duality gap is less than ϵ^*.

To compute the search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$, the SDPA employs Mehrotra type predictor-corrector procedure [13] with use of the HRVW/KSH/M search direction [10, 12, 14]. Let $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) = (\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ be a current point satisfying $\boldsymbol{X} \succ \boldsymbol{O}$ and $\boldsymbol{Y} \succ \boldsymbol{O}$.

In Mehrotra type predictor procedure, we first choose a centering parameter β_p for the predictor-step. If $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is an $\bar{\epsilon}$-approximate feasible solution of the SDP (1), then let $\beta_p = 0$, else let $\beta_p = \bar{\beta}$, where $\bar{\beta}$ is a parameter whose default value is 0.2. Ideally we want to solve the following system to compute a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ for the next point $(\boldsymbol{x} + d\boldsymbol{x}, \boldsymbol{X} + d\boldsymbol{X}, \boldsymbol{Y} + d\boldsymbol{Y})$ being on the central path $\mathcal{C}$.

$$\begin{array}{rcl} \boldsymbol{X} + d\boldsymbol{X} &=& \sum_{i=1}^m \boldsymbol{F}_i(x_i + dx_i) - \boldsymbol{F}_0, \\ \boldsymbol{F}_i \bullet (\boldsymbol{Y} + d\boldsymbol{Y}) &=& c_i \quad (i = 1, 2, \ldots, m), \\ (\boldsymbol{X} + d\boldsymbol{X})(\boldsymbol{Y} + d\boldsymbol{Y}) &=& \beta_p \mu \boldsymbol{I}, \end{array}$$

where $\mu = (\boldsymbol{X} \bullet \boldsymbol{Y})/n$. But this system is nonlinear, so that we ignore the nonlinear term $d\boldsymbol{X}d\boldsymbol{Y}$ in the third equation to obtain the modified Newton system, a system of linear equations in a predictor search direction $(d\boldsymbol{x}_p, d\boldsymbol{X}_p, d\boldsymbol{Y}_p)$:

$$\left.\begin{array}{rcl} \sum_{i=1}^m \boldsymbol{F}_i dx_i - d\boldsymbol{X} &=& \boldsymbol{P}, \\ \boldsymbol{F}_i \bullet \widehat{d\boldsymbol{Y}} &=& d_i \ (i = 1, 2, \ldots, m), \\ d\boldsymbol{X}\boldsymbol{Y} + \boldsymbol{X}\widehat{d\boldsymbol{Y}} &=& \boldsymbol{R}, \ \ d\boldsymbol{Y} = (\widehat{d\boldsymbol{Y}} + \widehat{d\boldsymbol{Y}}^T)/2, \end{array}\right\} \tag{7}$$

where $\boldsymbol{P} = \boldsymbol{F}_0 - \sum_{i=1}^m \boldsymbol{F}_i x_i + \boldsymbol{X}$, $d_i = c_i - \boldsymbol{F}_i \bullet \boldsymbol{Y}$ and $\boldsymbol{R} = \beta_p \mu \boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y}$. Note that $d\boldsymbol{Y} = (\widehat{d\boldsymbol{Y}} + \widehat{d\boldsymbol{Y}}^T)/2$ symmetrize the matrix $\widehat{d\boldsymbol{Y}}$ for $\boldsymbol{Y} + d\boldsymbol{Y}$ being a symmetric matrix. This symmetrization is based on the HRVW/KSH/M search direction [10, 12, 14]. It should be remarked that for any $\boldsymbol{X} \succ \boldsymbol{O}$, $\boldsymbol{Y} \succ \boldsymbol{O}$, $\boldsymbol{F}_i$ $(i = 1, 2, \ldots, m)$, $\boldsymbol{P} \in \mathbb{S}^n$, $\boldsymbol{R} \in \mathbb{R}^{n \times n}$ and $d_i \in \mathbb{R}$ $(i = 1, 2, \ldots, m)$, the modified Newton system (7) is guaranteed to have a unique solution $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ even in case when the central path $\mathcal{C}$ does not exists [12].

Then we proceed to Mehrotra type corrector procedure. We first compute a centering parameter β_c. Let

$$\beta_{aux} = \left(\frac{(\boldsymbol{X} + d\boldsymbol{X}_p) \bullet (\boldsymbol{Y} + d\boldsymbol{Y}_p)}{\boldsymbol{X} \bullet \boldsymbol{Y}} \right)^2.$$

The definition of β_{aux} is slightly different from the one proposed in the paper [13]. The computation of our β_{aux} is cheaper than the original one that requires the computation of the step sizes

$$\max\{\alpha : \boldsymbol{X} + \alpha d\boldsymbol{X}_p \succeq \boldsymbol{O}\} \ \text{ and } \ \max\{\alpha : \boldsymbol{Y} + \alpha d\boldsymbol{Y}_p \succeq \boldsymbol{O}\}.$$

We confirmed through numerical experiments that our β_{aux} works as effective as the original one. If $\beta_{aux} > 1.0$, let $\beta_c = 1.0$. Otherwise we consider two possible cases depending on the feasibility of the current point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$. If $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is an $\bar{\epsilon}$-approximate feasible solution of the SDP (1), let $\beta_c = \max\{\beta^*, \beta_{aux}\}$, else let $\beta_c = \max\{\bar{\beta}, \beta_{aux}\}$, where β^* and $\bar{\beta}$ are parameters whose default values are set at 0.1 and 0.2, respectively. Replacing $\boldsymbol{R} = \beta_p \mu \boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y}$ by $\boldsymbol{R} = \beta_c \mu \boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y} - d\boldsymbol{X}_p d\boldsymbol{Y}_p$, we then solve the modified Newton equation (7) to compute the predictor-corrector search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$.

We can reduce the modified Newton system (7) into the following system of linear equations.

$$\begin{array}{rcl} \boldsymbol{B} d\boldsymbol{x} &=& \boldsymbol{r}, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (8) \\ d\boldsymbol{X} &=& \displaystyle\sum_{i=1}^m \boldsymbol{F}_i dx_i - \boldsymbol{P}, \ \widehat{d\boldsymbol{Y}} = \boldsymbol{X}^{-1}(\boldsymbol{R} - d\boldsymbol{X}\ \boldsymbol{Y}), \ \ d\boldsymbol{Y} = (\widehat{d\boldsymbol{Y}} + \widehat{d\boldsymbol{Y}}^T)/2, \end{array}$$

where

$$\begin{array}{rcl} B_{ij} &=& (\boldsymbol{X}^{-1}\boldsymbol{F}_i\boldsymbol{Y}) \bullet \boldsymbol{F}_j \quad (1 \le i \le m, 1 \le j \le m), \\ r_i &=& -d_i + \boldsymbol{F}_i \bullet (\boldsymbol{X}^{-1}(\boldsymbol{R} + \boldsymbol{P}\boldsymbol{Y})) \quad (1 \le i \le m). \end{array}$$

We call the equation (8) as the Schur complement equation. To solve the Schur complement equation (8), the SDPA applies the Cholesky factorization to the coefficient matrix $\boldsymbol{B}$, which is a completely dense positive definite matrix in general; hence it requires $O(m^3)$ arithmetic operations. The SDPA solves the Schur complement equation twice in Mehrotra type predictor and corrector procedures explained above. But they share a common coefficient matrix $\boldsymbol{B}$. Therefore the SDPA computes the elements of $\boldsymbol{B}$ and factorizes it only once in each iteration. If we applied the standard dense matrix multiplication, the computation of $\boldsymbol{B}$ would require $\mathcal{O}(mn^3 + m^2n^2)$ arithmetic operations, and it would occupy the major part of the total CPU time of the execution of the SDPA. The SDPA exploits the sparsity of data matrices $\boldsymbol{F}_i$ $(i = 1, 2, \dots, m)$, and utilizes three distinct formulae developed in the paper [7] for computing B_{ij} taking account of the sparsity of $\boldsymbol{F}_i$ and $\boldsymbol{F}_j$; the first formula deals with the case where both $\boldsymbol{F}_i$ and $\boldsymbol{F}_j$ are dense, and the second the case where $\boldsymbol{F}_i$ is dense and $\boldsymbol{F}_j$ is sparse, and the third the case where both $\boldsymbol{F}_i$ and $\boldsymbol{F}_j$ are sparse. This significantly contributes to the computational efficiency of the SDPA. This sparsity handling technique was employed also in the SDPT3 [21]. See [7, 8] for more details.

We summarize the entire algorithm of the SDPA below.

The Primal-Dual Interior-Point Algorithm of the SDPA

Step 0 (Initialization): Choose an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ satisfying $\boldsymbol{X}^0 \succ \boldsymbol{O}$ and $\boldsymbol{Y}^0 \succ \boldsymbol{O}$. Let $k = 0$.

Step 1 (Checking Feasibility): If $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ is an $(\bar{\epsilon}, \epsilon^*)$-approximate optimal solution of the SDP (1), stop the iteration.

Step 2 (Computing a search direction): As described above, apply Mehrotra type predictor-corrector procedure to generate a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$.

Step 3 (Generating a new iterate): We first compute $\bar{\alpha_p}$ and $\bar{\alpha_d}$ as the maximum primal and dual step lengths so that $\boldsymbol{X}^{k+1} = \boldsymbol{X}^k + \alpha_p d\boldsymbol{X}$ and $\boldsymbol{Y}^{k+1} = \boldsymbol{Y}^k + \alpha_d d\boldsymbol{Y}$ remain positive semidefinite.

$$\bar{\alpha_p} = \max\{\alpha \in (0, \alpha_{\max}] \mid \boldsymbol{X}^k + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}\}, \quad \bar{\alpha_d} = \max\{\alpha \in (0, \alpha_{\max}] \mid \boldsymbol{Y}^k + \alpha d\boldsymbol{Y} \succeq \boldsymbol{O}\},$$

where $\alpha_{\max}$ is a sufficiently large constant; we take $\alpha_{\max} = 100$. The SDPA employs the Lanczos method for the computation of $\bar{\alpha_p}$ and $\bar{\alpha_d}$ [22]. If $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ is not an $\bar{\epsilon}$-approximate feasible solution of $\mathcal{P}$ then take the primal step size $\alpha_p = \min\{\gamma^* \bar{\alpha_p}, 1.0\}$, else take $\alpha_p = \gamma^* \bar{\alpha_p}$, where γ^* is a parameter whose default value is 0.9 to keep $\boldsymbol{X}^{k+1}$ positive definite. When $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ is not an $\bar{\epsilon}$-approximate feasible solution, we know that the equality

$$\boldsymbol{X}^k + \alpha_p d\boldsymbol{X} = \sum_{i=1}^{m} \boldsymbol{F}_i (x_i^k + \alpha_p dx_i) - \boldsymbol{F}_0$$

hold in general if and only if $\alpha_p = 1$; note that if $\boldsymbol{X}^k + \alpha_p d\boldsymbol{X} \succeq \boldsymbol{O}$ in addition, then $(\boldsymbol{X}^k + \alpha_p d\boldsymbol{X}, \boldsymbol{x}^k + \alpha_p d\boldsymbol{x})$ gets feasible. This is the reason why we impose $\alpha_p \leq 1.0$ on the primal step length in such a case. It should also be emphasized that if $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ is feasible then $(\boldsymbol{x}^l, \boldsymbol{X}^l)$ is also feasible for any $l \geq k$. We set the next iterate

$$(\boldsymbol{x}^{k+1}, \boldsymbol{X}^{k+1}, \boldsymbol{Y}^{k+1}) = (\boldsymbol{x}^k + \alpha_p d\boldsymbol{x}, \boldsymbol{X}^k + \alpha_p d\boldsymbol{X}, \boldsymbol{Y}^k + \alpha_d d\boldsymbol{Y})$$

Let $k \leftarrow k + 1$. Go to Step 1.

3 The SDPA Software Package

The SDPA user's manual and the source code of the SDPA are available at the WWW site

`http://www.is.titech.ac.jp/~kojima/sdpa/index.html`.

Since the SDPA 6.0 (the latest version at the moment of writing) incorporates *LAPACK* [1], one needs to compile and link it to the SDPA according to the SDPA user's manual. In this section, we show how to execute the SDPA using a small example of SDP.

We consider the standard form SDP (1) with the data.

$$m = 3, \quad n = 2, \quad \boldsymbol{c} = (48, -8, 20)^T,$$
$$\boldsymbol{F}_0 = \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix}, \; \boldsymbol{F}_1 = \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix}, \; \boldsymbol{F}_2 = \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix}, \; \boldsymbol{F}_3 = \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix}.$$

The following input data file for the SDPA contains these quantities.

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
   3  =  mDIM
   1  =  nBLOCK
   2  = bLOCKsTRUCT
{48, -8, 20}
0 1 1 1 -11
0 1 2 2 23
1 1 1 1 10
1 1 1 2 4
2 1 2 2 -8
3 1 1 2 -8
3 1 2 2 -2
```

Here we have named this file as 'example1.dat-s', but we could replace 'example1' by any name. The postfix 'dat-s' means the file is written in the SDPA-sparse-format. The SDPA accepts two different input file formats, the SDPA-sparse-format and the SDPA-dense-format. We need to use the postfix 'dat-s' for the former input format and 'dat' for the latter. The user can use freely either of the input formats depending on the sparsity of data matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$. The file 'example1.dat-s' is contained in the software package SDPA.

The 1st line "Example 1: mDim = 3, nBLOCK = 1, 2" of the file 'example1.dat-s' is a comment line. The 2nd line '3 = mDIM' indicates that the number of equality constraints of the dual SDP $\mathcal{D}$ is 3. The 3rd and 4th lines are used to describe the block diagonal structure illustrated below that is shared among the data matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$, the primal matrix variable $\boldsymbol{X}$ and the dual matrix variable $\boldsymbol{Y}$. In this example, they are all one block 2×2 matrices. Hence 'nBLOCK' (the number of blocks) is 1 and 'bLOCKsTRUCT' (the block structure) is 2. The 5th line '{48, -8, 20}' describes elements in vector $\boldsymbol{c}$. The lines below the 5th line define elements in matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \boldsymbol{F}_2, \boldsymbol{F}_3$. For instance, the 9th line '1 1 1 2 4' describes that the 1st block of $\boldsymbol{F}_1$ has 4 in the position $(1, 2)$, and the 12th line '3 1 2 2 -2' describes that the 1st block of $\boldsymbol{F}_3$ has -2 in the position $(2, 2)$. Note that only elements in the upper triangular part of $\boldsymbol{F}_0, \boldsymbol{F}_1, \boldsymbol{F}_2, \boldsymbol{F}_3$ need to be assigned since all the matrices are symmetric.

The SDPA-sparse-format is designed to describe large scale sparse SDPs which often appear in various applications. To describe a general block diagonal structure of the form

$$\left.\begin{array}{rcl} \boldsymbol{F} & = & \begin{pmatrix} \boldsymbol{B}_1 & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B}_2 & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{B}_\ell \end{pmatrix}, \\ \boldsymbol{B}_i & : & \text{a } p_i \times p_i \text{ symmetric matrix } \ (i = 1, 2, \ldots, \ell), \end{array}\right\} \tag{9}$$

we define 'nBLOCK' (the number of blocks) and 'bLOCKsTRCTUT' (the block diagonal structure) as follows:

$$\begin{array}{rcl} \text{nBLOCK} & = & \ell, \\ \text{bLOCKsTRUCT} & = & (\beta_1,\ \beta_2,\ \ldots,\ \beta_\ell), \\ \beta_i & = & \left\{ \begin{array}{rl} p_i & \text{if } \boldsymbol{B}_i \text{ is a symmetric matrix,} \\ -p_i & \text{if } \boldsymbol{B}_i \text{ is a diagonal matrix.} \end{array} \right. \end{array}$$

For example, if

$$\left(\begin{array}{ccc|cc|cc} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 & 0 & 0 \\ 3 & 5 & 6 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{array}\right), \tag{10}$$

then

$$\text{nBLOCK} = 3 \quad \text{and} \quad \text{bLOCKsTRUCT} = \quad (3,\ 2,\ -2).$$

To solve the example1 whose data is described in the file 'example1.dat-s', type

```
$ ./sdpa example1.dat-s example1.result
```

Here './sdpa' is the name of the SDPA executable binary and 'example1.result' is an output file, which contains solutions and some other information. The user can see, in the file 'example1.result', an approximate optimal solution $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ of the example1 with `the relative gap` $= 9.15 \times 10^{-8}$, `the primal feasible error` $= 1.50 \times 10^{-14}$, and `the dual feasible error` $= 3.13 \times 10^{-12}$. When the SDPA runs, it reads several parameters from the parameter file 'param.sdpa'. The user can control the execution of the SDPA through those parameters which include ϵ^*, $\bar{\epsilon}$, β^*, $\bar{\beta}$ and γ^* explained in Section 2; see also Table 1 for their default values.

Parameter	default value	
ϵ^*	1.0×10^{-7}	the accuracy of relative gap
$\bar{\epsilon}$	1.0×10^{-7}	the accuracy of feasibility
β^*	0.1	the centering parameter (feasible case)
$\bar{\beta}$	0.2	the centering parameter (infeasible case)
γ^*	0.9	the step size parameter

Table 1: Some Parameters of the SDPA

For more details of the SDPA and its callable library, see the SDPA user's manual.

4 Numerical Results on the SDPA 6.0

In this section, we evaluate the performance (efficiency and stability) of the SDPA 6.0 through numerical results on three types of problems, randomly generated SDPs with fully dense data matrices, SDPs picked up from SDPLIB [6], and large scale SDPs arising from quantum chemistry [15, 16]. We used the default parameters listed in Table 1 for all problems. We also compare the SDPA 6.0 with some existing software packages for solving general SDPs. We chose the SDPT3 [21], the CSDP [5],the SeDuMi [18] and the SDPA 5.01 (the previous version of the SDPA) [8] as competitors, since they demonstrated high performance in the SDPLIB benchmark [6]. All of the software packages incorporate the PDIPA (primal-dual interior-point algorithm). The main differences in the SDPA 6.0 and these software packages are:

- **Programming languages used for implementation**
  The SDPT3 and the SeDuMi are written in Matlab script and MEX-files, the CSDP is written in C, and the SDPA is written in C++. The difference in languages influences their user interface, callable library and efficiency of data access. But, their performance of dense matrix computation must be similar, since *LAPACK* is used in all these software packages,

- **Search direction**
  The SDPT3, the CSDP and the SDPA use the HRVW/KSH/M direction [10, 12, 14] as search directions in Mehrotra type predictor-corrector procedure, while the SeDuMi uses the NT direction [17]. The NT direction possesses various nice theoretical properties such as a low computational complexity and an extensibility to more general problems. In practical computation, especially for large scale sparse SDPs, however, the HRVW/KSH/M direction is cheaper than the NT direction.

- **Feasibility vs optimality**
  The SeDuMi is based on the homogeneous self-dual embedding of the PDIPA, and all others (the SDPA, the SDPT3 and the CSDP) on the PDIPA that can start from an infeasible interior point. In the former case, pursuing the feasibility and pursuing the optimality (or reducing the duality gap) during the iterations are simultaneously and automatically done by the homogeneous self-dual

embedding. In the latter case, we can control each of them independently. The SDPA places more weight in pursuing the feasibility than pursuing the optimality compared to the SDPT3 and the CSDP. This difference sometime affects their performance when they are applied to SDPs having narrow primal or dual feasible regions.

We set all software packages to stop their iterations when the relative gap becomes smaller than 1.0×10^{-7}. As the iterate $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ approaches an optimal solution, the primal variable matrix $\boldsymbol{X}$, the dual variable matrix $\boldsymbol{Y}$ and the coefficient matrix $\boldsymbol{B}$ in the Schur complement equation (8) becomes ill-conditioned, and the Cholesky factorization (or the inversion) of these matrices may cause serious numerical instability and/or serious numerical error. To avoid such difficulties, each software package incorporates some other stopping criteria besides the relative duality gap. In the SDPA 5.01 and 6.0, we took $\bar{\epsilon} = \epsilon^* = 1.0 \times 10^{-7}$. They attain an $(\epsilon^*, \bar{\epsilon})$-approximation optimal solution when they terminate normally. But, before having found an $(\epsilon^*, \bar{\epsilon})$-approximation optimal solution, they may terminate to avoid numerical instability caused by the Cholesky factorization of the ill-conditioned coefficient matrix $\boldsymbol{B}$ in the Schur complement equation (8).

4.1 Randomly Generated Standard Form SDPs

First, we report numerical results on randomly generated standard form SDPs. Each element of $\boldsymbol{F}_i (i = 0, 2, 3, \ldots, m)$ was chosen from the uniform distribution of $[-1, 1]$, while $\boldsymbol{F}_1$ was set at the $n \times n$ identity matrix $\boldsymbol{I}_n$. We chose $c_i = \boldsymbol{F}_i \bullet \boldsymbol{I}_n (i = 1, 2, \ldots, m)$. By construction, if we take $\boldsymbol{x} = (n+1, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $\boldsymbol{X} = (n+1)\boldsymbol{F}_1 - \boldsymbol{F}_0$ and $\boldsymbol{Y} = \boldsymbol{I}_n$, then $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ is a primal and dual interior feasible solution of the resulting SDP. Hence we know that the resulting SDP has a primal-dual optimal solution with no duality gap.

Table 2 shows the performance of the SDPA 6.0 for randomly generated SDPs. The numerical experiment was done on Pentium IV 2.2 GHz with 1GB memory under Linux 2.4.18. The first and the second columns of Table 2 denote the number m of the equality constraints of the dual SDP $\mathcal{D}$ and the size n of matrices $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{F}_0, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$, respectively. The third and the fourth columns display the computational time in second and the total number of iterations to solve each problem, respectively. The fifth column denotes the relative gap defined in (6), and the last column the primal-dual feasibility error which is defined as the maximum of the primal and the dual feasibility errors given in (5). In all cases, the SDPA 6.0 successfully attained $(\bar{\epsilon}, \epsilon^*)$-approximate optimal solutions in 14 - 16 iterations.

m	n	CPU time(second)	iter	relgap	feaserr
20	20	0.06	14	3.31×10^{-8}	1.86×10^{-13}
40	20	0.15	14	5.51×10^{-8}	1.89×10^{-13}
80	20	0.29	14	2.04×10^{-8}	1.53×10^{-13}
200	40	3.74	15	4.30×10^{-8}	3.92×10^{-13}
200	100	24.85	15	5.49×10^{-8}	1.38×10^{-12}
400	40	11.14	15	2.17×10^{-8}	1.31×10^{-12}
400	100	75.94	16	1.18×10^{-8}	1.82×10^{-12}
800	40	36.61	14	1.78×10^{-8}	2.07×10^{-12}
800	100	244.94	16	1.13×10^{-8}	3.47×10^{-12}
800	200	1243.33	16	1.39×10^{-8}	5.57×10^{-12}

Table 2: Performance of the SDPA 6.0 applied to randomly generated SDPs

As shown in Table 3, the SDPA 6.0 achieves the fastest among the software packages we tested. The main reason is that the SDPA converts the input date read as a sparse format to a dense format automatically if its density exceeds a threshold value. In addition, the SDPA 6.0 employs *LAPACK* for dense matrix computation. Since data matrices $\boldsymbol{F}_1, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ of the above randomly generated SDPs are fully dense, *LAPACK* significantly improves the performance of the SDPA 5.01. One point we have to note is that the CPU time shown in Table 3 includes the time of reading data from a file; in some of the software packages including the SDPA 6.0, this part can not be clearly separated because when they read data as the SDPA-sparse-format, they simultaneously transform the data to another data format or data structure

m	n	SDPA 6.0	SDPT3	CSDP	SeDuMi	SDPA 5.01
20	20	0.06	0.64	0.11	0.49	0.13
40	20	0.15	0.77	0.23	0.46	0.24
80	20	0.29	1.47	0.62	0.83	0.65
200	40	3.74	11.63	11.39	9.07	13.17
200	100	24.85	68.61	72.54	83.48	125.11
400	40	11.14	26.18	40.54	20.81	40.46
400	100	75.94	157.91	251.91	182.37	337.63
800	40	36.61	66.54	153.68	61.14	134.19
800	100	244.94	345.33	976.03	435.36	1012.84
800	200	1243.33	$*$	5667.43	2904.71	5494.47

Table 3: Comparison in CPU time in second to solve randomly generated SDPs by the SDPA 6.0, the SDPT3, the CSDP, the SeDuMi and the SDPA 5.01 ('$*$' stands for 'out of memory').

name	m	nBLOCK	bLOCKsTRUCT
arch8	174	1	(161)
control11	1596	2	(110, 55)
gpp500-1	501	1	(500)
mcp500-3	500	1	(500)
theta6	4375	1	(300)
truss8	496	34	(19, 19, . . . , 19, 1)
equalG11	801	1	(801)
maxG32	2000	1	(2000)
thetaG51	6910	1	(1001)

Table 4: Selected problems from SDPLIB. Here m denotes the number of the equality constraints of the dual SDP $\mathcal{D}$, and nBLOCK and bLOCKsTRUCT define the block diagonal structure of each problem (see Section 3).

for succeeding efficient computation. The SDPT3 took a longer time for this part than the SDPA 6.0. This fact partially reflects the difference in the SDPA 6.0 and the SDPT3.

4.2 SDPLIB

SDPLIB[6] is a set of benchmark SDPs. Since SDPLIB collects 92 SDPs from various fields such as control theory and graph theory, we select 9 problems listed in Table 4 to show their numerical results in Table 5.

Table 5 shows the CPU time in second for each software to solve each selected problem from SDPLIB and the total CPU time in second to solve all problems in SDPLIB except maxG55 and maxG60. The two problems maxG55 and maxG60 need so large memory and any software we chose could not solve because of out of memory.

We know from Table 5 that the SDPA 6.0 is the fastest software to solve many problems in SDPLIB. In particular, the SDPA 6.0 shows a higher performance for problems with a larger number of equality constraints compared to the size of data matrices such as theta6 and thetaG51. On the other hand, the SDPT3 shows a higher performance for problems with a large size of data matrices compared to the number of equality constraints such as maxG32, and the CSDP for the problems with many small block diagonal data matrices such as truss8. The SDPA 6.0 apparently spent a longer CPU time to solve control11 than the SDPT3. This is because the SDPA 6.0 employs a strategy that places more weight to perusing the feasibility than reducing the relative gap as mentioned above. Actually, the SDPA attained a smaller feasibility error, 6.04×10^{-8}, than that of the SDPT3, 5.35×10^{-7} in this problem. We take this strategy since a small relative gap at a point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \in \mathbb{R}^m \times \mathbb{S}^n_+ \times \mathbb{S}^n_+$ is meaningful only when the feasibility error at the point $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y}) \in \mathbb{R}^m \times \mathbb{S}^n_+ \times \mathbb{S}^n_+$ is sufficiently small. But this strategy required more CPU time since the feasible region of control11 is very narrow.

name	SDPA 6.0	SDPT3	CSDP	SeDuMi	SDPA 5.01
arch8	5.2	15.5	13.7	38.1	19.46
control11	498.4	264.5	1028.3	531.7	1294.63
gpp500-1	28.8	66.7	104.8	2676.7	762.91
mcp500-3	19.7	28.6	53.4	250.8	256.82
theta6	292.5	390.2	1176.8	1989.9	1441.2
truss8	24.2	19.2	8.4	8.9	18.26
equalG51	252.0	484.7	484.3	4062.9	3197.68
maxG32	2263.7	784.2	4514.1	14477.5	$*$
thetaG51	3501.4	6540.6	12017.0	14342.1	13693.1
total 90 problems	10344.2	11432.6	27034.1	81465.5	18516.8 $+ *$

Table 5: Numerical Results on SDPLIB benchmark problems: CPU time in second to solve each selected problem and the total CPU time in second to solve all 90 problems of SDPLIB ('$*$' stands for 'out of memory').

From the total CPU time for 90 problems in Table 5, we may conclude the SDPA 6.0 and the SDPT3 attained a higher performance for SDPLIB benchmark problems than the CSDP, the SeDuMi and the SDPA 5.01.

4.3 Application to Quantum Chemistry

Variational calculations of fermion second-order reduced density matrix is an application of SDPs arising from quantum chemistry [15, 16]. It is a minimization problem of the total energy $\boldsymbol{H} \bullet \boldsymbol{Y}$ of fermion system subject to linear constraints and the positive semidefinite constraints on $\boldsymbol{Y} \in \mathbb{S}^n$, which is formulated in the dual standard form SDP $\mathcal{D}$ of (1). Here $\boldsymbol{H}$ denotes the Hamiltonian. The linear equality constraints involves various physical constraints on the number of electrons, spin squared operator, eigen state of the number of electrons, etc.. The positive semidefinite constraint $\boldsymbol{Y} \succeq \boldsymbol{O}$ comes from the character of second-order reduced density matrices. See [15, 16] for more details of the formulation.

There are some points to be noted on this application. First, the resulting problem has quite many equality constraints, *i.e.*, the number of equality constraints m of the dual SDP $\mathcal{D}$ of (1) can be very large; $m \geq 15000$ in the largest problems that we solved. See Table 6 which shows the number m of equality constraints and the block diagonal structure of the problems that we tested. This causes a difficulty in storing the $m \times m$ fully dense coefficient matrix $\boldsymbol{B}$ of the Schur complement equation (8) and tremendeous computational time in its Cholesky factorization. The other point is that the data matrices are very sparse, so that the SDPA 6.0 could fully utilized the sparsity handling technique proposed in the paper [7] to considerably save the computational time.

System,state,basis	m	nBLOCK	bLOCKsTRUCT
BH_3,$^1A_1$,STO-6G	2897	2	(120,120)
CH_3,$^1A_1$,STO-6G	2897	2	(120,120)
HF,$^1\Sigma^+$,STO-6G	4871	3	(66,66,144)
OH^+,$^3\Sigma^-$,STO-6G	4871	3	(66,66,144)
NH_2,$^2A_1$,STO-6G	8993	3	(91,91,196)
H_2O,$^1A_1$,STO-6G	8993	3	(91,91,196)
CH_4,$^1A_1$,STO-6G	15313	3	(120,120,256)
LiF, $^1\Sigma$,STO-6G	15313	3	(120,120,256)
Ne, ^1S, Valence double-ζ	15313	3	(120,120,256)

Table 6: SDPs arisen from quantum chemistry

As shown in Tables 7 and 8, the SDPA 6.0 successfully solved all problems in a reasonably high accuracy. Table 7 shows compariosn in CPU time among the SDPA 6.0 the SDPT3, the CSDP, SeDuMi and the

System, state, basis	SDPA	SDPT3	CSDP	SeDuMi	SDPA 5.01
BH_3,$^1A_1$,STO-6G	183.48	395.30	644.04	520.41	768.74
CH_3,$^1A_1$,STO-6G	183.31	336.44	688.35	529.83	769.40
HF,$^1\Sigma^+$,STO-6G	562.7	542.50	3089.34	2450.77	2383.44
OH^+,$^3\Sigma^-$,STO-6G	644.73	614.76	3327.68	2474.36	2334.35
NH_2,$^2A_1$,STO-6G	3340.12	3619.80	13562.28	$*$	17360.88
H_2O,$^1A_1$,STO-6G	3167.07	3843.09	14629.91	$*$	23590.75
CH_4,$^1A_1$,STO-6G	18789.17	$*$	$*$	$*$	$*$
LiF, $^1\Sigma$,STO-6G	20823.65	$*$	$*$	$*$	$*$
Ne, ^1S, Valence double-ζ	20802.14	$*$	$*$	$*$	$*$

Table 7: Numerical Results on quantum chemical problems: CPU time in second to solve problem ('$*$' stands for 'out of memory').

System, state, basis	SDPA	SDPT3	CSDP	SeDuMi	SDPA 5.01
BH_3,$^1A_1$,STO-6G	1.28×10^{-8}	4.55×10^{-9}	1.18×10^{-8}	1.57×10^{-8}	1.16×10^{-8}
CH_3,$^1A_1$,STO-6G	3.00×10^{-8}	6.30×10^{-8}	1.11×10^{-8}	1.04×10^{-8}	2.85×10^{-8}
HF,$^1\Sigma^+$,STO-6G	6.65×10^{-6}	2.90×10^{-8}	2.87×10^{-8}	1.03×10^{-10}	3.98×10^{-5}
OH^+,$^3\Sigma^-$,STO-6G	1.13×10^{-5}	3.03×10^{-8}	4.79×10^{-8}	6.41×10^{-8}	6.72×10^{-6}
NH_2,$^2A_1$,STO-6G	3.03×10^{-6}	2.97×10^{-7}	4.86×10^{-8}	$*$	7.93×10^{-7}
H_2O,$^1A_1$,STO-6G	6.54×10^{-6}	1.11×10^{-7}	1.34×10^{-6}	$*$	2.02×10^{-6}
CH_4,$^1A_1$,STO-6G	6.19×10^{-6}	$*$	$*$	$*$	$*$
LiF, $^1\Sigma$,STO-6G	5.58×10^{-6}	$*$	$*$	$*$	$*$
Ne, ^1S, Valence double-ζ	9.64×10^{-6}	$*$	$*$	$*$	$*$

Table 8: Numerical Results on quantum chemical problems: Relative gap ('$*$' stands for 'out of memory').

SDPA 5.01. We observe that the SDPA 6.0 and the SDPT3 solved the problems with $m = 2897$, 4871 and 8993 faster than the other software packages. Only the SDPA 6.0 could solve the largest problem with $m = 15313$ because all other software packages encountered out of memory. The numerical experiments on this application were done on ATHLON 1.2 GHz with 2GB memory under Linux 2.4.18.

Table 8 shows the relative gap defined as (6) at each approximate optimal solution obtained by the software packages under comparison. At a glance, the SDPT3 attained a smaller relative gap than the SDPA 6.0 in comparable time, so that we may think the SDPT3 generated higher quality approximate solutions than the SDPA 6.0. This observation is not necessarily true because we need to check their feasibility errors too. Table 9 shows that the feability error, defined as (5), at approximate solutions obtained by the SDPA 6.0 is smaller than that of the SDPT3 in the problems that they both could solve. This is due to the strategy that the SDPA 6.0 places more weight to persuing the feasibility than reducing the relative gap as mentioned above.

System, state, basis	SDPA	SDPT3
BH_3,$^1A_1$,STO-6G	3.05×10^{-13}	5.90×10^{-10}
CH_3,$^1A_1$,STO-6G	2.09×10^{-11}	2.90×10^{-9}
HF,$^1\Sigma^+$,STO-6G	2.44×10^{-9}	1.44×10^{-7}
OH^+,$^3\Sigma^-$,STO-6G	9.79×10^{-8}	1.97×10^{-7}
NH_2,$^2A_1$,STO-6G	2.99×10^{-8}	7.86×10^{-7}
H_2O,$^1A_1$,STO-6G	6.01×10^{-8}	1.12×10^{-7}

Table 9: Numerical Results on quantum chemical problems: Feasible error.

5 Concluding Remarks

Through numerical results, we have checked the stability and the performance of the SDPA 6.0. In particular, exploiting sparsity provides us great efficiency. It contributes to solving large scale and sparse problems arising from various field.

We have compared the performance of the SDPA 6.0 with other existing software packages. The SDPA 6.0 achieves the highest speed for many of the problems tested. We have to point out, however, that the accuracy attained by the SDPA 6.0 is lower than that of other software packages in SDPs arising from quantum chemistry although the feasibility error attained by the SDPA 6.0 is smaller. Ideally both higher accuracy and smaller feasibility error are preferable, but not so easy to attain in some problems. Further investigation is necessary to guarantee a well-balanced high quality solution.

The authors hope the SDPA will help many researchers' study in various fields. Any feedback from practical applications will bring big benefits to the SDPA. They will continue to refine the SDPA in order to apply it to more real problems.

ACKNOWLEDGMENTS
The authors thank Mr. Maho Nakata of Graduate School of Engineering, Kyoto University, for many valuable suggestions and helpful cooperation on applications of the SDPA to quantum chemistry. The authors also thank Professor Kim Chaun Toh of National University of Singapore for a modification of the SDPT3 so as to solve relatively large randomly generated SDPs correctly.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. 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).

[2] S. J. Benson and Y. Ye, DSDP home page. http://www.mcs.anl.gov/~benson/dsdp (2002).

[3] A. Ben-Tal and A. Nemirovskii *Lectures on Moden Convex Optimizatin Analysis, Alogorithms, and Engineering Applications*, (SIAM, Philadelphia, 2001).

[4] S. Boyd *et al*, "Linear matrix inequalities in system and control theory" *Society for Industrial and Applied Mathematics 1994 Philadelphia*, PA, ISBN 0-89871-334-X

[5] B. Borchers, " CSDP, A C Library for Semidefinite Programming," *Optimization Methods and Software* **11 & 12** (1999) 613–623.

[6] B. Borchers, "SDPLIB 1.2, a library of semidefinte programming test problems," *Optimization Methods and Software* **11 & 12** (1999) 683–690.

[7] K. Fujisawa, M. Kojima and K. Nakata, "Exploiting Sparsity in Primal-Dual Interior-Point Methods for Semidefinite Programming," *Mathematical Programming* **79** (1997) 235–253.

[8] 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) pp.267-301.

[9] M. X. Goemans and D. P. Williamson, "Improved approximation alogrithoms for maxmum cut and satisfiability problems using semidefinite programming," *Journal of Association for Computing Machinery* **42(6)** (1995) 1115–1145.

[10] 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.

[11] M. Kojima, S. Mizuno and A. Yoshise, "A Primal-Dual Interior Point Algorithm for Linear Programming", in: N. Megiddo, ed., *Progress in Mathematical Programming: Interior Point and Related Methods* (Springer-Verlag, New York, 1989) 29–47.

[12] M. Kojima, S. Shindoh and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problems," *SIAM Journal on Optimization* **7** (1997) 86-125.

[13] S.Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization* **2** (1992) 575–601.

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

[15] M. Nakata, H. Nakatsuji, M. Ehara, M. Fukuda, K. Nakata and K. Fujisawa, "Variational calculations of fermion second-order deduced density matrices by semidefinite programming algorithm," *Journal of Chemical Physics* **114** (2001) 8282–8292.

[16] M. Nakata, M. Ehara, and H. Nakatsuji, "Density matrix variational theory: Application to the potential energy surfaces and strongly correlated systems," *Journal of Chemical Physics* **116** (2002) 5432–5439.

[17] Yu. E. Nesterov and M. J. Todd, "Primal-Dual Interior-Point Methods for Self-Scaled Cones," *SIAM Journal on Optimization* **8** (1998) 324–364.

[18] J. F. Strum, "SeDuMi 1.02, a MATLAB toolbox for optimizatin over symmetric cones", *Optimization Methods and Software* **11 & 12** (1999) 625–653.

[19] K. Tanabe, "Centered Newton Method for Mathematical Programming," in: M. Iri and K. Yajima, eds., *System Modeling and Optimization* (Springer, New York, 1988) 197–206.

[20] M.J. Todd, "Semidefinite optimization," *Acta Numerica* **10** (2001) 515–560.

[21] M. J. Todd, K. C. Toh and R. H. Tütüncü, "SDPT3 – a MATLAB software package for semidefinite programming, version 1.3.", *Optimization Methods and Software* **11 & 12** (1999) 545–581.

[22] K. C. Toh, "A note on the calculation of step-lengths in interior-point methods for semidefinite programming," *Computational Optimization and Applications* **21** (2002) 301–310.

[23] L. Vandenberghe, S. Boyd, "Positive-Definite Programming," *Mathematical Programming: State of the Art 1994* J.R.Birge and K.G.Murty ed.s, U. of Michigan, 1994.

[24] H. Wolkowicz, R. Saigal and L. Vandenberghe, *Handbook of Semidefinite Programming, Theory, Algorithms, and Applications*, (Kluwe Academic Publishers, Massachusetts, 2000).