SDPA PROJECT : SOLVING LARGE-SCALE SEMIDEFINITE PROGRAMS

Katsuki Fujisawa
Chuo University

Kazuhide Nakata
Tokyo Institute of Technology

Makoto Yamashita
Kanagawa University

Mituhiro Fukuda
Tokyo Institute of Technology

Abstract   The Semidefinite Program (SDP) has recently attracted much attention of researchers in various fields for the following reasons: (i) It has been intensively studied in both theoretical and numerical aspects. Especially the primal-dual interior-point method is known as a powerful tool for solving large-scale SDPs with accuracy. (ii) Many practical problems in various fields such as combinatorial optimization, control and systems theory, robust optimization and quantum chemistry can be modeled or approximated by using SDPs. (iii) Several software packages for solving SDPs and related problems (ex. the Second-Order Cone Program : SOCP) are available on the Internet.

   In 1995, we started the SDPA Project aimed for solving large-scale SDPs with numerical stability and accuracy. The SDPA (SemiDefinite Programming Algorithm) is a C++ implementation of a Mehrotra-type primal-dual predictor-corrector interior-point method for solving the standard form SDP and its dual. We have also developed some variants of the SDPA to handle SDPs with various features. The SDPARA is a parallel version of the SDPA on multiple processors and distributed memory, which replaces two major bottleneck components of the SDPA by their parallel implementation using MPI and ScaLAPACK. The SDPARA on parallel computer has attractive features; it can load a large-scale SDP into the distributed memory and solve it in a reasonable time. In this paper, we show through some numerical experiments that the SDPARA attains high performance. The SDPARA-C is an integration of two software SDPARA and SDPA-C which is a primal-dual interior-point method using the positive definite matrix completion technique. The SDPARA-C requires a small amount of memory when we solve sparse SDPs with a large-scale matrix variable and/or a large number of equality constraints. The paper also explains a grid portal system for solving SDPs, which we call the SDPA Online Solver.

   In this paper, we review the major achievements of the SDPA Project on solving large-scale SDPs. This paper provides an introductory and comprehensive materials for researchers who are interested in practical computational aspects of the SDPs.

   Keywords: Optimization, semidefinite program, primal-dual interior-point method, parallel computing, grid computing

1. Introduction

The progress in information technology has dramatically changed how we solve large-scale optimization problems. Many optimization problems classified into mathematical programs have a lot of practical applications in various fields including operations research, economics, chemistry, and biology. Solving large-scale optimization problems generally requires a huge amount of computation time and resources, and consequently, the size of such problems that we can solve has been limited. The new technologies of parallel computing has recently received much attention as a powerful and inexpensive methodology for solving large-scale optimization problems. In particular, the grid technology is the next generation computation infrastructure which enables us to easily access computational resources including hardware and software libraries distributed across a wide-area networks like the Internet.

The Semidefinite Program (SDP) is one of the most popular mathematical programs. In the last decade, the SDP has been intensively studied in both theoretical and practical aspects. The SDP has many applications covering various fields such as combinatorial optimization, polynomial optimization, control and systems theory, robust optimization and quantum chemistry [6, 11, 18, 19, 21, 28, 30, 31, 34]. Many researchers proposed polynomial time algorithms for SDPs. The Primal-Dual Interior-Point Method (PDIPM) [1, 14, 17, 20, 25] is an outstanding method among them, since it solves large-scale SDPs with numerical stability and accuracy. Other methods for solving large-scale SDPs can be found for instance in [5, 16, 24]. Note that both the size of SDPs and the numerical accuracy of solutions are important factors to discuss the performance of different methods for large-scale SDPs. The numerical accuracy obtained by the PDIPM is acceptable for many practical SDP applications.

The main objective of the SDPA Project started in 1995 is to implement the PDIPM for SDPs and to solve large-scale SDPs with various characteristics, e.g., sparsity, large number of constraints and so on. The SDPA (SemiDefinite Programming Algorithm) [33] is a C++ implementation of a Mehrotra-type predictor-corrector PDIPM for solving the standard form SDP and its dual. We can obtain other implementations of PDIPM such as SeDuMi [26], SDPT3 [29] and CSDP [4] from the Internet.

In the recent applications of SDPs, however, we must generate and solve large-scale SDPs that an existing single-unit CPU cannot process. There are two major time consuming components at each iteration in the PDIPM, even if we exploit the sparsity of the data matrices. The first component is the computation of all elements of the Schur complement matrix. The second component is its Cholesky factorization. The SDPARA [32] developed in 2002 replaces these two major bottleneck components of the SDPA by their parallel implementation using MPI and ScaLAPACK. The SDPARA on a parallel computer has the capability to load a large-scale SDP into distributed memory and solves it quickly compared with the SDPA. The SDPARA-C [23] is an integration of two software packages: the SDPA-C [22] which is a variant of the PDIPM using the positive definite matrix completion technique and the SDPARA. The SDPARA-C successfully reduces the required memory space for sparse SDPs with a large-scale matrix variable and/or a large number of equality constraints. In this paper, we show through numerical experiments that exploiting sparsity and parallel computation are effective in solving large-scale SDPs. Finally, the SDPA Online Solver is a grid portal system which enables users with few computational resources to execute software packages, the SDPA, the SDPARA and the SDPARA-C, on the parallel computing environments via grid and cluster computing technology and to receive benefits of parallel computing.

In this paper, we survey the main achievements of the SDPA Project on the development of the SDPA and its variants which we call the SDPA Family. Along more than 10 years of the existence of the project, algorithm efficiency were persuaded over the latest progress of information technology.

The paper is organized as follows. In Section 2, we briefly explain the SDPA and other software packages of the SDPA Family. Then we outline the algorithmic framework of the PDIPM, and the block diagonal structure which the SDPA Family handles. Section 3 describes the positive matrix completion theory and a general framework of exploiting the aggregate sparsity pattern over all data matrices of large-scale and sparse SDPs which is used in SDPA-C and SDPARA-C. In Section 4, we explain parallel computation techniques for solving large-scale SDPs implemented in two parallel SDP software packages, the SDPARA and the SDPARA-C. In particular, the SDPARA on a parallel computer can solve the

largest and general SDPs which the other state-of-the-art SDP software packages cannot solve (general SDPs mean that they have no special structure such as rank-1 condition). In Section 5, we present a successful application in quantum chemistry and show that the SDPARA attains high performance in the numerical experiments. Finally, Section 6 explains a grid portal system for solving SDPs, the SDPA Online Solver.

2. SDPA Family

In this section, we briefly explain the outline and purpose of each software package which composes the SDPA Family. We also describe the algorithmic framework of the PDIPM and the block diagonal structure handled in the SDPA Family. More detailed information of the SDPA Family is available at the SDPA Web site:

<div align="center">

http://sdpa.indsys.chuo-u.ac.jp/sdpa/

</div>

All software packages of the SDPA Family are also available from the above Web site.

2.1. Outline of each software package

1. SDPA [7, 8, 33]

   The SDPA (SemiDefinite Programming Algorithm) is a software package for solving SDPs and plays the central role in the SDPA Family. It is based on the Mehrotra-type predictor-corrector infeasible PDIPM. The SDPA handles the standard form SDP and its dual problem stated in Section 2.2. It is implemented in C++ language and utilizes the ATLAS [1] (or BLAS) and LAPACK [2] libraries for matrix computations. The SDPA incorporates dynamic memory allocation and deallocation. Therefore, the maximum size of loaded SDPs depends on the size of the computational memory where the SDPA is installed. The SDPA has the following additional features:

   (a) Callable library for the SDPA functions
   (b) Block diagonal and sparse matrix data structures
   (c) Some information on the feasibility and optimality
   (d) Exploiting the sparsity of the data matrices

2. SDPARA [32] (parallel version)

   The SDPARA (SemiDefinite Programming Algorithm PARAllel version) is a parallel version of the SDPA written in C++ language. The SDPARA works as a stand-alone software package and solves SDPs in parallel with the help of the MPI (Message Passing Interface) and ScaLAPACK (Scalable LAPACK) [3]. However, its functions are not available as a callable library. To install and use the package, we assume that users can access to a parallel computer where the MPI is installed on.

3. SDPA-C [22] (with the positive definite matrix completion)

   The SDPA-C (SemiDefinite Programming Algorithm with the positive definite matrix Completion) is a C++ software package which solves sparse SDPs very efficiently (in computation time and memory) if the data matrices of the SDP are large-scale and sparse. For instance, SDPs with number of equality constraints $m < 2000$, matrix sizes $n < 20000$, and 5% of nonzero density. More specifically, the SDPA-C is more efficient when there is a special structure in the aggregated sparsity pattern of the data matrices. The positive definite matrix completion enables the SDPA-C to store only the sparse matrices and perform matrix computations taking advantages of the sparsity. Besides the ATLAS (or BLAS) and LAPACK libraries for matrix computations, it utilizes the

---

[1] http://www.netlib.org/atlas/
[2] http://www.netlib.org/lapack/
[3] http://www.netlib.org/scalapack/

METIS [4] and SPOOLES [5] libraries for finding a proper structure of the aggregated sparsity pattern.

4. SDPARA-C [23] (parallel version of the SDPA-C)
   The SDPARA-C is a parallel version of the SDPA-C: a variant of the SDPARA which incorporates the positive definite matrix completion technique. As the SDPA-C, the following packages are necessary to be installed a priori: ATLAS (or BLAS), ScaLAPACK, METIS, SPOOLES, and MPI.

5. SDPA-M [9] (MATLAB interface)
   The SDPA-M provides a MATLAB interface for the SDPA. It shares all the features with the SDPA except that no callable library is provided. The SDPA-M can be combined with your programs in a MATLAB environment.

2.2. Standard form SDP and block diagonal structure

Each software package of the SDPA Family solves the following standard form semidefinite program and its dual.

$$
\text{SDP} \begin{cases}
\mathcal{P}: & \text{minimize} & \sum_{k=1}^{m} c_k x_k \\
& \text{subject to} & \boldsymbol{X} = \sum_{k=1}^{m} \boldsymbol{F}_k x_k - \boldsymbol{F}_0, \quad \boldsymbol{X} \succeq \boldsymbol{O}, \\
& & \boldsymbol{X} \in \mathcal{S}^n. \\
\mathcal{D}: & \text{maximize} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\
& \text{subject to} & \boldsymbol{F}_k \bullet \boldsymbol{Y} = c_k \ (k = 1, 2, \ldots, m), \quad \boldsymbol{Y} \succeq \boldsymbol{O}, \\
& & \boldsymbol{Y} \in \mathcal{S}^n.
\end{cases}
$$

We denote the primal-dual pair $\mathcal{P}$ and $\mathcal{D}$ by SDP. Throughout this paper, we use the following notations.

$\mathcal{S}^n$: the set of $n \times n$ real symmetric matrices.

$\boldsymbol{F}_k \in \mathcal{S}^n \ (k = 0, 1, 2, \ldots, m)$: constraint matrices.

$\boldsymbol{O} \in \mathcal{S}^n$: the zero matrix.

$\boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \cdot \\ c_m \end{pmatrix} \in \mathbb{R}^m$: a cost vector, $\quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ x_m \end{pmatrix} \in \mathbb{R}^m$: a variable vector,

$\boldsymbol{X} \in \mathcal{S}^n$, $\boldsymbol{Y} \in \mathcal{S}^n$: variable matrices,

$\boldsymbol{U} \bullet \boldsymbol{V}$: The inner product of $\boldsymbol{U}, \boldsymbol{V} \in \mathcal{S}^n, i.e., \sum_{i=1}^{n} \sum_{j=1}^{n} U_{ij} V_{ij}$

$\boldsymbol{U} \succeq \boldsymbol{O} \iff \boldsymbol{U} \in \mathcal{S}^n$ is positive semidefinite,

$\boldsymbol{U} \succ \boldsymbol{O} \iff \boldsymbol{U} \in \mathcal{S}^n$ is positive definite.

The SDP is determined by m, n, $\boldsymbol{c} \in \mathbb{R}^m$, $\boldsymbol{F}_k \in \mathcal{S}^n \ (k = 0, 1, 2, \ldots, m)$. When $(\boldsymbol{x}, \boldsymbol{X})$ is a feasible solution (or a minimum solution, resp.) of the primal problem $\mathcal{P}$ and $\boldsymbol{Y}$ is a

---

[4] http://glaros.dtc.umn.edu/gkhome/views/metis/
[5] http://www.netlib.org/linalg/spooles/

feasible solution (or a maximum solution, resp.) of the dual problem $\mathcal{D}$, we call $(\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{Y})$ a feasible solution (or an optimal solution, resp.) of the SDP.

The SDPA handles block diagonal matrices as we have mentioned in Section 2.1. The common matrix data structure for the constraint matrices $\boldsymbol{F}_0$, $\boldsymbol{F}_1$, ..., $\boldsymbol{F}_m$ is expressed by the number of blocks, denoted by nBLOCK, and the block structure vector, denoted by bLOCKsTRUCT. For instance, if we consider a block diagonal matrix $\boldsymbol{F}$ of the form

$$\boldsymbol{F} = \begin{pmatrix} \boldsymbol{B}_1 & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B}_2 & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ . & . & . & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{B}_\ell \end{pmatrix}, \tag{2.1}$$

with $\boldsymbol{B}_i \in \mathcal{S}^{p_i}$ $(i = 1, 2, \ldots, \ell)$, we define these two parameters by

$$\begin{aligned} \text{nBLOCK} &= \ell, \text{ and} \\ \text{bLOCKsTRUCT} &= (\theta_1, \theta_2, \ldots, \theta_\ell), \\ \theta_i &= \begin{cases} p_i & \text{if } \boldsymbol{B}_i \text{ is a symmetric matrix,} \\ -p_i & \text{if } \boldsymbol{B}_i \text{ is a diagonal matrix.} \end{cases} \end{aligned}$$

For example, if $\boldsymbol{F}$ is of the form

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 & 0 & 0 \\ 3 & 5 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}, \tag{2.2}$$

we have

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

If

$$\boldsymbol{F} = \begin{pmatrix} \star & \star & \star \\ \star & \star & \star \\ \star & \star & \star \end{pmatrix}, \quad \text{where } \star \text{ denotes a real number,}$$

is a usual symmetric matrix with a single block, we have

$$\text{nBLOCK} = 1 \quad \text{and} \quad \text{bLOCKsTRUCT} = 3.$$

2.3. Primal-dual interior-point method

All software packages of the SDPA Family are based on the Primal-Dual Interior-Point Method (PDIPM) [1, 14, 17, 20, 25]. The main feature of the PDIPM is to solve the primal SDP $\mathcal{P}$ and the dual SDP $\mathcal{D}$ simultaneously. Keeping the positive definiteness of variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, the PDIPM reduces the duality gap until approximate optimal solutions of $\mathcal{P}$ and $\mathcal{D}$ are found.

Let $(\boldsymbol{x}^+, \boldsymbol{X}^+, \boldsymbol{Y}^+)$ and $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ be a feasible and an optimal solutions of the SDP, respectively. Under the assumption that the SDP has an interior feasible solution, there is no duality gap between the primal and dual optimal objective values, that is,

$$\boldsymbol{F}_0 \bullet \boldsymbol{Y}^+ \le \boldsymbol{F}_0 \bullet \boldsymbol{Y}^* = \sum_{k=1}^m c_k x_k^* \le \sum_{k=1}^m c_k x_k^+.$$

Since the optimal solution $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ satisfies the primal matrix equality and dual linear constraints, it follows that

$$\boldsymbol{X}^* \bullet \boldsymbol{Y}^* \;\; = \;\; \sum_{k=1}^{m} c_k x_k^* - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^* = 0.$$

Hence, we can derive $\boldsymbol{X}^*\boldsymbol{Y}^* = \boldsymbol{O}$ from the conditions $\boldsymbol{X}^* \succeq \boldsymbol{O}$ and $\boldsymbol{Y}^* \succeq \boldsymbol{O}$. As a result, we obtain the Karush-Kuhn-Tucker (KKT) optimality condition:

$$\text{(KKT)} \quad \begin{cases} \boldsymbol{X}^* = \sum_{k=1}^{m} \boldsymbol{F}_k x_k^* - \boldsymbol{F}_0, \\ \boldsymbol{F}_k \bullet \boldsymbol{Y}^* = c_k \quad (k = 1, 2, \ldots, m), \\ \boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{O}, \\ \boldsymbol{X}^* \succeq \boldsymbol{O}, \boldsymbol{Y}^* \succeq \boldsymbol{O}. \end{cases}$$

The central path is a key concept in the design and analysis of most interior-point methods. It is composed of points defined by a perturbed KKT optimality condition:

$$\text{The central path} \equiv \{(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu)) \in \mathbb{R}^m \times \mathcal{S}^n \times \mathcal{S}^n : \mu > 0\},$$

where $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ satisfies

$$\text{(perturbed KKT)} \quad \begin{cases} \boldsymbol{X}(\mu) = \sum_{k=1}^{m} \boldsymbol{F}_k x_k(\mu) - \boldsymbol{F}_0, \\ \boldsymbol{F}_k \bullet \boldsymbol{Y}(\mu) = c_k \quad (k = 1, 2, \ldots, m), \\ \boldsymbol{X}(\mu)\boldsymbol{Y}(\mu) = \mu\boldsymbol{I}, \\ \boldsymbol{X}(\mu) \succ \boldsymbol{O}, \boldsymbol{Y}(\mu) \succ \boldsymbol{O}. \end{cases}$$

For any $\mu > 0$, there exists a unique $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu)) \in \mathbb{R}^m \times \mathcal{S}^n \times \mathcal{S}^n$ satisfying (perturbed KKT), and the central path forms a smooth curve. We also note that $\boldsymbol{X}(\mu) \bullet \boldsymbol{Y}(\mu) = n\mu$ from $\boldsymbol{X}(\mu)\boldsymbol{Y}(\mu) = \mu\boldsymbol{I}$. It should be emphasized that the central path converges to an optimal solution of the SDP, that is, $(\boldsymbol{x}(\mu), \boldsymbol{X}(\mu), \boldsymbol{Y}(\mu))$ converges to an optimal solution of the SDP as $\mu \to 0$. Thus the PDIPM traces the central path numerically decreasing μ toward 0.

Algorithmic Framework of the PDIPM [7]

Step 0: (Initialization) We choose an initial point $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ with $\boldsymbol{X}^0 \succ \boldsymbol{O}, \boldsymbol{Y}^0 \succ \boldsymbol{O}$. Let $\mu^0 = \boldsymbol{X}^0 \bullet \boldsymbol{Y}^0/n$ and $h = 0$. We set the parameters $0 < \beta < 1$ and $0 < \gamma < 1$.

Step 1: (Checking Convergence) If μ^h is sufficiently small and $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ approximately satisfies the feasibility, we stop the iteration and output the current point $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ as an approximate optimal solution.

Step 2: (Search Direction) We compute a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ toward a target point $(\boldsymbol{x}(\beta\mu^h), \boldsymbol{X}(\beta\mu^h), \boldsymbol{Y}(\beta\mu^h))$ on the central path with $\mu = \beta\mu^h$.

Step 3: (Step Length) We compute step lengths α_p and α_d such that $\boldsymbol{X}^h + \alpha_p d\boldsymbol{X} \succeq \boldsymbol{O}$ and $\boldsymbol{Y}^h + \alpha_d d\boldsymbol{Y} \succeq \boldsymbol{O}$.

Step 4: (Update) We update the current point by $(\boldsymbol{x}^{h+1}, \boldsymbol{X}^{h+1}, \boldsymbol{Y}^{h+1}) = (\boldsymbol{x}^h + \gamma\alpha_p d\boldsymbol{x}, \boldsymbol{X}^h + \gamma\alpha_p d\boldsymbol{X}, \boldsymbol{Y}^h + \gamma\alpha_d d\boldsymbol{Y})$. Let $\mu^{h+1} = \boldsymbol{X}^{h+1} \bullet \boldsymbol{Y}^{h+1}/n$ and $h \leftarrow h + 1$. Go to Step 1.

We only need an initial feasible solution $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ which satisfies $\boldsymbol{X}^0 \succ \boldsymbol{O}, \boldsymbol{Y}^0 \succ \boldsymbol{O}$. When $(\boldsymbol{x}^h, \boldsymbol{X}^h, \boldsymbol{Y}^h)$ is infeasible, the step lengths α_p and α_d in Step 3 need to be chosen so that some feasibility measure improves as well as $\boldsymbol{X}^h + \alpha_p d\boldsymbol{X}$ and $\boldsymbol{Y}^h + \alpha_d d\boldsymbol{Y}$ remain positive definite.

The computation of the search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ in Step 2 usually consumes most of the computation time in solving an SDP. A fundamental strategy to shorten the total computation time in parallel processing is to use a distributed computation in Step 2. This will be discussed in Section 4. We want to take a search direction $(d\boldsymbol{x}, d\boldsymbol{X}, d\boldsymbol{Y})$ so that $(\boldsymbol{x}^h + d\boldsymbol{x}, \boldsymbol{X}^h + d\boldsymbol{X}, \boldsymbol{Y}^h + d\boldsymbol{Y})$ coincides with the targeting point $(\boldsymbol{x}(\beta\mu^h), \boldsymbol{X}(\beta\mu^h), \boldsymbol{Y}(\beta\mu^h))$ on the central path with $\mu = \beta\mu^h$. This leads to

$$\begin{cases} \boldsymbol{X}^h + d\boldsymbol{X} = \sum_{k=1}^m \boldsymbol{F}_k(x_k^h + dx_k) - \boldsymbol{F}_0, \\ \boldsymbol{F}_k \bullet (\boldsymbol{Y}^h + d\boldsymbol{Y}) = c_k \quad (k = 1, 2, \ldots, m), \\ (\boldsymbol{X}^h + d\boldsymbol{X})(\boldsymbol{Y}^h + d\boldsymbol{Y}) = \beta\mu^h \boldsymbol{I}. \end{cases}$$

Here we need not to consider the positive definite conditions $\boldsymbol{X}^h + d\boldsymbol{X} \succeq \boldsymbol{O}$ and $\boldsymbol{Y}^h + d\boldsymbol{Y} \succeq \boldsymbol{O}$ because we recover the conditions by adjusting the step lengths α_p and α_d in Step 3. The above system is almost a linear system except the term $d\boldsymbol{X} d\boldsymbol{Y}$ in the last equality. Neglecting this nonlinear term and introducing an auxiliary matrix $\widetilde{d\boldsymbol{Y}}$, we can reduce the system of nonlinear equations above into the following system of linear equations.

$$\begin{cases} \boldsymbol{B} d\boldsymbol{x} &= \boldsymbol{r}, \\ d\boldsymbol{X} &= \boldsymbol{R}_p + \sum_{k=1}^m \boldsymbol{F}_k dx_k, \\ \widetilde{d\boldsymbol{Y}} &= (\boldsymbol{X}^h)^{-1}(\boldsymbol{R}_c - d\boldsymbol{X}\boldsymbol{Y}^h), \quad d\boldsymbol{Y} = (\widetilde{d\boldsymbol{Y}} + \widetilde{d\boldsymbol{Y}}^T)/2, \end{cases} \tag{2.3}$$

where

$$\begin{cases} B_{pq} &= ((\boldsymbol{X}^h)^{-1}\boldsymbol{F}_p\boldsymbol{Y}^h) \bullet \boldsymbol{F}_q \ (p, q = 1, 2, \ldots, m), \\ r_k &= -d_k + \boldsymbol{F}_k \bullet ((\boldsymbol{X}^h)^{-1}(\boldsymbol{R}_c - \boldsymbol{R}_p\boldsymbol{Y}^h)) \ (k = 1, 2, \ldots, m), \\ \boldsymbol{R}_p &= \sum_{k=1}^m \boldsymbol{F}_k x_k^h - \boldsymbol{F}_0 - \boldsymbol{X}^h, \\ d_k &= c_k - \boldsymbol{F}_k \bullet \boldsymbol{Y}^h \ (k = 1, 2, \ldots, m), \\ \boldsymbol{R}_c &= \beta\mu^h \boldsymbol{I} - \boldsymbol{X}^h \boldsymbol{Y}^h. \end{cases} \tag{2.4}$$

We call the system of linear equations $\boldsymbol{B} d\boldsymbol{x} = \boldsymbol{r}$ the *Schur complement equation* (SCE) and its coefficient matrix $\boldsymbol{B}$ the *Schur complement matrix* (SCM). We first solve the SCE, then compute $d\boldsymbol{X}$ and $d\boldsymbol{Y}$. Note that the size of the SCM $\boldsymbol{B}$ corresponds to the number m of equality constraints in $\mathcal{D}$. Since the SCM $\boldsymbol{B}$ is positive definite through all iterations of the PDIPM, we apply the Cholesky factorization to $\boldsymbol{B}$ for computing the solution $d\boldsymbol{x}$ of the SCE. The computation cost to evaluate the SCM $\boldsymbol{B}$ is $O(m^2 n^2 + mn^3)$ arithmetic operations when the data matrices $\boldsymbol{F}_k \ (k = 1, 2, \ldots, m)$ are fully dense, while its Cholesky factorization requires $O(m^3)$ arithmetic operations. Note that $\boldsymbol{B}$ becomes fully dense in general even when some of the data matrices are sparse. The auxiliary matrix $\widetilde{d\boldsymbol{Y}}$ is introduced to make $d\boldsymbol{Y}$ symmetric. The search direction used here is called the HRVW/KSH/M direction [14, 17, 20].

3. Matrix Completion

3.1. Sparsity via matrix completion

In this section, we focus our attention on solving sparse SDPs where the data matrices $\boldsymbol{F}_k \ (k = 0, 1, \ldots, m)$ are sparse. To represent structural sparsity of the given SDP, we introduce the *aggregate sparsity pattern* of the data matrices:

$$E = \{(i, j) \in V \times V : [\boldsymbol{F}_k]_{ij} \neq 0 \text{ for some } k \in \{0, 1, 2, \ldots, m\}\}.$$

Here V denotes the set $\{1, 2, \ldots, n\}$ of row/column indices of the data matrices $\boldsymbol{F}_0, \boldsymbol{F}_1, \ldots,$ $\boldsymbol{F}_m$, and $[\boldsymbol{F}_k]_{ij}$ denotes the (i, j)th element of $\boldsymbol{F}_k \in \mathcal{S}^n$.

A *positive definite matrix completion* of a partial symmetric matrix $\bar{\boldsymbol{Y}}$ with specified elements $\bar{Y}_{ij} \in \mathbb{R}$ $((i, j) \in E')$ means a positive semidefinite matrix $\boldsymbol{Y}$ such that $Y_{ij} = \bar{Y}_{ij}$ $((i, j) \in E')$. For every pair of subsets S and T of V, we use the notation $\boldsymbol{Y}_{ST}$ for the submatrix obtained by deleting all rows $i \notin S$ and all columns $j \notin T$. Following the definitions and ideas presented in [10], consider a collection of nonempty subsets $C_1, C_2, \ldots, C_\ell$ of V satisfying

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

(ii) Any partial symmetric matrix $\bar{\boldsymbol{Y}}$ with specified elements $\bar{Y}_{ij} \in \mathbb{R}$ $((i, j) \in E')$ has a positive definite matrix completion if and only if the submatrices $\bar{\boldsymbol{Y}}_{C_r C_r} \succ \boldsymbol{O}$ $(r = 1, 2, \ldots, \ell)$.

From condition (i), we observe that values of the objective functions and constraint linear functions $\boldsymbol{F}_k \bullet \boldsymbol{Y}$ $(k = 0, 1, \ldots, m)$ involved in the SDP $\mathcal{D}$ are completely determined only by values of entries Y_{ij} $((i, j) \in E')$ and independent from values of entries Y_{ij} $((i, j) \notin E')$. In other words, if $\boldsymbol{Y}, \boldsymbol{Y}' \in \mathcal{S}^n$ satisfy $Y_{ij} = Y'_{ij}$ $((i, j) \in E')$ then

$$\boldsymbol{F}_k \bullet \boldsymbol{Y} = \boldsymbol{F}_k \bullet \boldsymbol{Y}' \ (k = 0, 1, \ldots, m).$$

The remaining entries Y_{ij} $((i, j) \notin E')$ only affect whether $\boldsymbol{Y}$ is positive definite. By condition (ii), we know whether we can assign some appropriate values to those remaining entries Y_{ij} $((i, j) \notin E')$ so that the whole resultant matrix $\boldsymbol{Y}$ becomes positive definite.

Let $G(V, E')$ be an undirected graph with vertex set V and edge set E'. We call the subset C_r of the vertex set V a clique whenever it induces a clique of $G(V, E')$. Then we call $\{C_1, C_2, \ldots, C_\ell\}$ in (i) a family of maximal cliques of $G(V, E')$. It is known that (ii) holds if and only if the graph $G(V, E')$ is chordal [13, Theorem 7] (*cf.*, also [10, Theorem 2.3]). A graph is said to be *chordal* if every cycle of length ≥ 4 has a chord (an edge connecting two non-consecutive vertices of the cycle). We call E' in (i) the *extended sparsity pattern* of E.

One of the key ideas to solve SDPs efficiently via the positive definite matrix completion is to use of the desirable property (ii) thoroughly to diminish the number of unknowns in the dual matrix variable $\boldsymbol{Y}$. Therefore, we want to strongly emphasize the importance of determining a chordal extension $G(V, E')$ of $G(V, E)$ with as small number of edges as possible. A chordal extension can be easily obtained if we apply a symbolic Cholesky factorization to the symmetric matrix with sparsity pattern specified by E after reordering of its rows/columns specified by the vertex set $V = \{1, 2, \ldots, n\}$. Unfortunately, the problem of finding the best ordering that minimizes the fill-in arising in Cholesky factorization is $\mathcal{NP}$ hard. Hence, it seems reasonable at least in practice to employ some existing heuristic methods, for example, the multilevel nested dissection and/or the minimum degree. SDPA-C and SDPARA-C use METIS and SPOOLES as listed in Section 2.1. Once we determine an ordering, a simple symbolic Cholesky factorization with this ordering provides us with a chordal extension $G(V, E')$ which is a chordal graph. The family of maximal cliques of $G(V, E')$, $\{C_1, C_2, \ldots, C_\ell\}$, can be easily computed from it [10].

3.2. Completion method

In this subsection, we present a PDIPM based on positive definite matrix completion which we can apply directly to the SDPs. We call this *the completion method*. The following lemma plays the central role in the completion method.

Lemma 3.1 ([10]) *The maximum-determinant positive definite matrix completion $\hat{\boldsymbol{Y}}$ of $\bar{\boldsymbol{Y}}$ can be determined in terms of the sparse factorization formula*

$$\boldsymbol{P}\hat{\boldsymbol{Y}}^{-1}\boldsymbol{P}^T = \boldsymbol{M}\boldsymbol{M}^T, \tag{3.1}$$

where $\boldsymbol{M}$ is a lower triangular matrix with possible nonzero elements specified in E' and $\boldsymbol{P}$ is a permutation matrix.

In addition to this lemma, we note that the primal matrix variable

$$\boldsymbol{X} = \sum_{k=1}^{m} \boldsymbol{F}_k x_k - \boldsymbol{F}_0$$

inherits the sparsity of $\boldsymbol{F}_p$'s. Furthermore $\boldsymbol{X} \succeq \boldsymbol{O}$ has a Cholesky factorization

$$\boldsymbol{P}\boldsymbol{X}\boldsymbol{P}^T = \boldsymbol{N}\boldsymbol{N}^T \tag{3.2}$$

without any fill-in except elements specified in $E'\backslash E$.

The standard PDIPM needs to store all elements of the matrices $\boldsymbol{Y}$ and $\boldsymbol{X}^{-1}$. Recall that the inverse $\hat{\boldsymbol{Y}}^{-1}$ is expressed in terms of a sparse factorization (3.1); hence $\hat{\boldsymbol{Y}} = \boldsymbol{P}^T\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{P}$. Also $\boldsymbol{X}^{-1} = \boldsymbol{P}^T\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{P}$ by (3.2). Here, $\boldsymbol{P}$ is a permutation matrix. The matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ have nonzero elements only in the extended sparsity pattern E'. The main feature of the completion method is to store sparse matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ instead of dense matrices $\boldsymbol{Y}$ and $\boldsymbol{X}^{-1}$. Hence, we perform all matrix operations using only sparse matrices, but neither introduce nor store any dense matrix.

In this paper, we describe how to compute three parts of the PDIPM (See [22] for more details).

Schur complement matrix $\boldsymbol{B}$

The completion method computes the Schur complement matrix (SCM) $\boldsymbol{B}$ in (2.4) by the below formula.

$$B_{pq} = \sum_{k=1}^{n} (\boldsymbol{P}^T\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{P}\boldsymbol{e}_k)^T \boldsymbol{F}_q (\boldsymbol{P}^T\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{P}[\boldsymbol{F}_p]_{*k}) \qquad (p,q=1,2,\ldots,m). \tag{3.3}$$

Here $[\boldsymbol{F}_p]_{*k}$ denotes the kth column of the data matrices $\boldsymbol{F}_p \in \mathcal{S}^n$ $(p=1,2,\ldots,m)$, and $\boldsymbol{e}_k \in \mathbb{R}^n$ the vector with the kth element 1 and others 0. When we compute a matrix-vector multiplication with the inverse of either of the matrices $\boldsymbol{M}$, $\boldsymbol{M}^T$, $\boldsymbol{N}$ or $\boldsymbol{N}^T$, $e.g.$, $\boldsymbol{v} = \boldsymbol{M}^{-1}\boldsymbol{e}_k$, we solve the system of linear equations $\boldsymbol{M}\boldsymbol{v} = \boldsymbol{e}_k$ instead. Since these matrices are sparse and triangular, we can solve the corresponding linear equations efficiently.

Computing the auxiliary matrix $\widetilde{d\boldsymbol{Y}}$

Now we compute the auxiliary matrix $\widetilde{d\boldsymbol{Y}}$ in (2.3). Each column of $\widetilde{d\boldsymbol{Y}}$ is computed by

$$[\widetilde{d\boldsymbol{Y}}]_{*k} = \beta\mu\boldsymbol{P}^T\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{P}\boldsymbol{e}_k - [\hat{Y}]_{*k} - \boldsymbol{P}^T\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{P}d\boldsymbol{X}\boldsymbol{P}^T\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{P}\boldsymbol{e}_k \atop (k=1,2,\ldots,n). \tag{3.4}$$

We do not need to compute the elements $[\widetilde{d\boldsymbol{Y}}]_{ij}$ with indices $(i,j) \notin E'$, since they do not contribute in the search direction [10, section 5]. Each matrix-vector multiplication in (3.4) is done in the same way as above.

Step length

To determine the primal and dual step lengths in Step 3 of PDIPM, we compute the minimum eigenvalues of the matrices

$$\bar{\boldsymbol{M}}_r^{-1} d\bar{\boldsymbol{Y}}_{C_r C_r} \bar{\boldsymbol{M}}_r^{-T} \in \mathcal{S}^{C_r} \ (r = 1, 2, \ldots, \ell), \ \text{ and } \ \boldsymbol{N}^{-1} \boldsymbol{P} d\boldsymbol{X} \boldsymbol{P}^T \boldsymbol{N}^{-T} \in \mathcal{S}^n,$$

where $\bar{\boldsymbol{M}}_r$ is a Cholesky factorization of $\bar{\boldsymbol{Y}}_{C_r C_r} \succ \boldsymbol{O}$, that is, $\bar{\boldsymbol{Y}}_{C_r C_r} = \bar{\boldsymbol{M}}_r \bar{\boldsymbol{M}}_r^T$. Since the first ℓ matrices are smaller than the dual matrix variable $\boldsymbol{Y}$, their computation is much faster. In addition, the minimum eigenvalue of $\boldsymbol{N}^{-1} \boldsymbol{P} d\boldsymbol{X} \boldsymbol{P}^T \boldsymbol{N}^{-T}$ can be computed easily by the Lanczos method, because $\boldsymbol{N}$ and $d\boldsymbol{X}$ are sparse matrices whose nonzero elements can appear in E'. In both of the cases, we can avoid dense matrix computation and exploit the sparsity of the matrices.

3.3. Numerical experiments

Here, we present some numerical experiments to evaluate the completion method. We compare the completion method with SDPA 5.01 [8], which is an implementation of the standard PDIPM for SDPs. The comparisons are made over four classes of SDPs, namely, the norm minimization problems, the SDP relaxations of quadratic programs with box constraints, max-cut problems over lattice graphs and graph partition problems over lattice graphs. All of them are randomly generated SDPs. Table 1 shows the comparisons between the "standard" method (SDPA) and the completion method (SDPA-C).

Table 1: Numerical results on norm minimization problems

			standard		completion	
			CPU	memory	CPU	memory
	n	m	(s)	(MB)	(s)	(MB)
norm minimization problems	1000	11	3302.2	321	100.4	5
quadratic programs	2001	1001	3201.3	316	758.3	18
max-cut problems	1000	1000	2806.6	315	320.9	19
graph partition problems	1000	10001	3218.1	315	893.2	27

Numerical results show that the completion method is very efficient when the SDP is large-scale and both its aggregate sparsity pattern and its extended sparsity pattern are sparse. More numerical results are presented in [22].

4. Parallel Computation for Extremely Large-Scale SDPs

As described in Section 1, the range of SDP applications is continuously being expanded. Some applications among them, however, require extremely large-scale SDPs. The size of an SDP can be roughly estimated by two factors m and n (the number of equality constraints of the SDP $\mathcal{D}$ and the dimension of variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively). State-of-the-art SDP solvers on a single-unit CPU have no difficulty to solve SDPs of size $m < 1000$ and $n < 1000$. However, SDPs arising from the quantum chemistry (details can be found in Section 5) often have m being greater than 10000. In addition, we encounter SDPs with both m and n greater than 40000 in combinatorial optimization. Such SDPs are extremely large for a single-unit CPU, mainly due to the limited memory space. Even when we can store the SDPs on the memory space, a long computation time is needed. Meanwhile, the parallel computation provides us a vast reach of memory space and remarkable reductions of the computation time. It is now an essential tool to tackle extremely large-scale SDPs.

We have developed two parallel SDP software packages, the SDPARA [32] and the SDPARA-C [23]. The SDPARA is a straight parallel version of the SDPA for general large-scale SDPs, while the SDPARA-C for sparse large-scale SDPs is an integration of parallel techniques cultivated through the SDPARA developments and the scheme of positive definite matrix completion methods in the SDPA-C.

In this section, we briefly discuss parallel techniques implemented in these two software packages.

4.1. Parallel techniques for general SDPs

First of all, we should focus on which components of the PDIPM consume large amount of memory space and computation time. The most serious subroutines in the PDIPM framework are forming the Schur complement matrix $\boldsymbol{B}$ (SCM) and its Cholesky factorization to obtain $d\boldsymbol{x}$, a part of the search direction. We call the former component 'ELEMENTS' and the later 'CHOLESKY'.

From the viewpoints of memory space, the SCM consumes the most vast memory space through the iterations of the PDIPM when we solve general SDPs, since the dimension of $\boldsymbol{B}$ is m and m is often further larger than n. Distributed memory attached to parallel computation enables us to store $\boldsymbol{B}$ even in the case of $m > 40000$.

These two components also occupy most of the total computation time. For example, these occupy more than 80% of computation time for SDPs arising from control theory and from combinatorial optimization listed in the standard SDP benchmark collection (SDPLIB) [3].

By reviewing the formula of the elements of $\boldsymbol{B}$, we discuss the parallel evaluation of ELEMENTS implemented in the SDPARA. Each element of $\boldsymbol{B}$ is evaluated by

$$B_{pq} = \left(\boldsymbol{X}^{-1}\boldsymbol{F}_p\boldsymbol{Y}\right) \bullet \boldsymbol{F}_q.$$

For the pth row, we compute $\boldsymbol{X}^{-1}\boldsymbol{F}_p\boldsymbol{Y}$ only once, then take the inner-product with $\boldsymbol{F}_q$ ($q = p, p+1, \ldots, m$). Note that $\boldsymbol{B}$ is always symmetric. The computation of each row is completely independent from other rows. Therefore, we assign only one processor to each row. In other words, the assigned processor evaluates all the elements in the row. We show an example of a processors assignment in Figure 1 in the case $m = 10$ and the number of available processors $U = 4$. We call the scheme *row-wise distribution*.

The row-wise distribution seems very simple in appearance. However, by duplicating $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ on all processors before the evaluation, each processor can concentrate on its own row computation without any network communications. This fact is an essential resource of high performance for the SDPARA as numerical reports in the next section show. The row-wise distribution also matches with sparsity techniques adopted in the SDPA [8]. In addition, the distribution still keeps simple memory storage.

For CHOLESKY component, we employ the parallel Cholesky factorization supplied by ScaLAPACK [2]. However, the row-wise distribution for ELEMENTS is inadequate to draw out enough performance of the parallel Cholesky factorization. Therefore, we redistribute the Schur complement matrix to the two-dimensional block-cyclic distribution as Figure 2 on distributed memory space. In Figure 2, for example, $B_{33}, B_{34}, B_{43}, B_{44}$ are stored in the 4th processor, while $B_{17}, B_{18}, B_{27}, B_{28}$ are stored in the 2nd processor. The redistribution, of course, requires network communications. However, the cost of communications can be completely disregarded since the parallel Cholesky factorization of ScaLAPACK on the row-wise distribution is considerably slower than on the two-dimensional block-cyclic distribution.
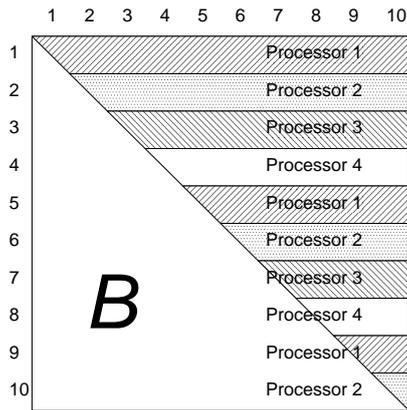
Figure 1: Row-wise distribution for the Schur complement matrix of the dimension 10 with 4 processors
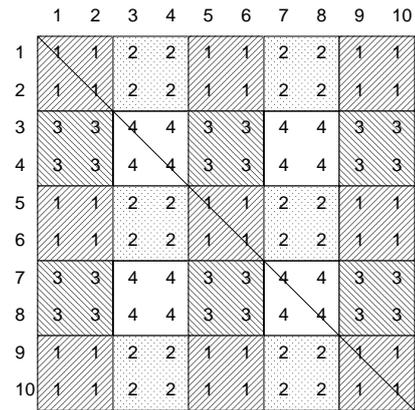

Figure 2: Two-dimensional block-cyclic distribution for parallel Cholesky factorization for 4 processors

In the SDPARA, the parallel components are only ELEMENTS and CHOLESKY and other components are basically the same as the SDPA on a single-unit CPU. Numerical results on quantum chemistry in the next section show how large SDPs the SDPARA can solve and the remarkably high scalability of the SDPARA.

4.2. Parallel techniques for sparse SDPs in an integration with the completion method

The SDPARA-C inherits the parallel components of the SDPARA. We further replace the other component, the computation of $\widetilde{d\boldsymbol{Y}}$, by its parallel variant to integrate the completion method described in the previous section. We call the additional component DMATRIX.

In the completion method, each element of the SCM is evaluated by (3.3). Hence, the row-wise distribution is still basically useful. However, the computational load for the pth row is almost proportional to $\#NZC_p$, where $NZC_p \subset \{1, \ldots, n\}$ is the index set of non-zero column vectors of $\boldsymbol{F}_p$ and $\#NZC_p$ is its cardinality. For instance, in an SDP relaxation of max-cut problem, only $\boldsymbol{F}_1$ is the identity matrix, while each other input matrix $\boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ has only one non-zero column vectors. More precisely, $\#NZC_1 = n$ and $\#NZC_2, \ldots, \#NZC_m = 1$. Applying the row-wise distribution as it stands can not keep the computational load balance between multiple processors. Hence, we can not obtain any advantage of parallel computation in computation time. To recover the load balance, we modify the parallel scheme to resolve such heavy row loads.

Now, we separate the row indexes $\{p : 1 \le p \le m\}$ into two disjoint subsets $\mathcal{Q}$ and $\mathcal{R}$ by

$$\mathcal{Q} = \{p : 1 \le p \le m, \#NZC_p > thres\}, \qquad \mathcal{R} = \{p : 1 \le p \le m\}\backslash\mathcal{Q}$$

where *thres* is a fixed threshold. Since $\mathcal{Q}$ is a collection of heavy computation rows, we assign all processors to each row in $\mathcal{Q}$. For $p \in \mathcal{Q}$, let us assume $NZC_p = \{k_1, k_2, \ldots, k_s\}$ with $s = \#NZC_p$. Then the uth processor $(u = 1, \ldots, U)$ computes

$$B_{pq}^u = \sum_{k_v : (v-u)\%U=0} (\boldsymbol{P}^T\boldsymbol{M}^{-T}\boldsymbol{M}^{-1}\boldsymbol{P}e_{k_v})^T\boldsymbol{F}_q(\boldsymbol{P}^T\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{P}[\boldsymbol{F}_p]_{*k_v}) \qquad (q = 1, 2, \ldots, m),$$

where $v = 1, \ldots, s$ and $a\%b$ indicates the remainder of the division of a by b. After the evaluation on each processor, we gather all information by $B_{pq} = \sum_u B_{pq}^u$ and store the result into the memory space of the pth row in the manner of the row-wise distribution.

For $\mathcal{R}$, we adopt the original row-wise distribution, since the computation cost for the rows are so cheaper than that for $\mathcal{Q}$ that the discrepancy of the load-balance would not be remarkable. We call the parallel scheme of the $\mathcal{Q}$ and $\mathcal{R}$ separation for ELEMENTS component in the SDPARA-C *the hashed row-wise distribution.* Meanwhile, the parallel Cholesky factorization of the SDPARA-C is identical to the SDPARA.

Next, we move to the other component DMATRIX, which is not parallelized in the SDPARA since DMATRIX computation cost is not so heavy for general SDPs. However, for SDPs with structural sparsity, particularly for SDPs which the completion method works effectively, the DMATRIX cost becomes prominent.

The kth column of $\widetilde{d\mathbf{Y}}$ is evaluated by (3.4). The attractive property of this formula is that the computation of each column is also completely independent from the other columns. Furthermore, the computation cost for each column is almost same. Therefore, the substance of its parallel scheme should be simple. We assign the uth processor to the columns which belong to $\{k : 1 \leq k \leq n, \ (k - u)\%U = 0\}$. After all the processors finish their computation, they exchange their own results with each other.

In Table 2, we show the computation time and scalability of the SDPARA-C for a randomly generated max-cut SDP relaxation problem on a lattice graph. The numerical experiments were executed on AIST (National Institute of Advanced Industrial Science and Technology) Super Cluster P-32 (Opteron 246 2.0GHz with 6GB memory) [6]. Since the size of the SDP is $m = n = 5000$, the SDP is not so large. We chose a middle size SDP so that we could solve it even on a single-unit CPU and show its scalability.

Table 2: Computation time and scalability of SDPARA-C for the max-cut problem (time unit is second)

Number of Processors	1 CPU	4 CPUs		16 CPUs		64 CPUs	
	time	time	scal	time	scal	time	scal
ELEMENTS	937.0	270.4	3.4	74.6	12.6	23.0	40.7
CHOLESKY	825.1	142.0	5.8	49.7	16.6	19.9	41.5
DMATRIX	459.5	120.4	3.8	30.9	14.9	9.2	49.9
Total	2239.7	544.4	4.1	166.8	13.4	70.8	31.6

From Table 2, we can observe that all the three parallel components successfully shrinks their computation time. On 64 processors they attain excellent scalability, greater than 40 times speed-up. Sometimes the scalability exceeds the number of available processors, since when the available processors increase, the necessary memory space on each processor becomes smaller and then the cache memory works better. The SDPARA-C is also effective to other SDPs than the above SDPs. More numerical results can be found in [23].

The point we should note at the last of this section is that we select the SDPARA or the SDPARA-C depending on the sparsity structure of the target SDPs. If the sparsity structure works well with the positive completion methods, the SDPARA-C has to be selected. Otherwise the SDPARA will achieve better performance, because the SDPARA is designed for general large-scale SDPs. Investigating the sparsity structure enables us to select more appropriate software. Numerical results in [23, 32] will provide more information in making the selection.

---

[6]http://unit.aist.go.jp/tacc/en/supercluster.html

5. A Successful Application in Quantum Chemistry

The determination of the electronic structure of atoms or molecules is a source of some of the largest and most challenging problems in computational science. It consists in determining the lowest energy, the *ground-state energy*, and the corresponding "state" of electrons of a atom or molecule subject to an external potential (due to the mass and electrical charge of nuclei). See [11] for a detailed description and historical notes. Traditional formulations of the electronic structure problem give rise to large linear and nonlinear Hermitian eigenvalue problems. However, according to a formalism fomented around 1960's, it is possible to retrieve all physical information of electrons solving large-scale SDPs where the matrix variables are approximations of the so-called *Reduced Density Matrices* (RDMs). Computational results came in 1970's for small atoms and molecules. However, because of high computational cost and lack of accuracy, interest in the computational aspects of the RDM method fell off gradually. It was in 2001 when Nakata *et al.* [21] employing the SDPA [33] obtained successful numerical results for the first time for concrete atoms and molecules which resulted in better approximations than obtained by the traditional Hartree-Fock (HF) method. In 2004, Zhao *et al.* [34] employing the SDPARA [32] solved even larger molecules incorporating more restrictive conditions which definitely showed that the RDM method is comparable to the state-of-the-art methods as CCSD(T) implemented in widely-used quantum chemistry packages [12] in terms of quality but not in terms of computation time. Some other recent algorithms to solve these (SDP) problems in quantum chemistry other than interior-point methods are [6, 19].

Table 3 shows typical SDP sizes and the required time when solving these electronic structure calculations in high performance computing environment using the parallel code SDPARA [32] described in the previous section. For each molecule, we can impose the so-called P, Q, and G conditions and also additionally the $T1$ and $T2$ conditions which are all of linear semidefinite type. See [6, 11, 19, 21, 34] for details. If we impose the PQG conditions, the SDP will have 18 block matrices while imposing $PQGT1T2$ conditions, it will have 22; we only display the sizes of the largest block matrices. For instance, 4018×2 means 2 block matrices of size 4018×4018 each. The numerical experiments were conducted on (a) Opteron 850 (2.4GHz) with 8GB of main memory, (b) Opteron 246 (2.0GHz) with 6GB of main memory per node, and (c) Itanium2 (1.3GHz) with 16GB of main memory per node. The last two systems are part of the AIST Super Cluster systems: P-32 and M-64, respectively. The water molecule constitutes our largest SDP solved so far with such data density and required accuracy. Even larger than the previously reported ones [11]. It took about 21 days using 8 processors on (c).

Table 4 shows the ground-state energy obtained solving the SDPs involving the PQG conditions (column 3) and $PQGT1T2$ conditions (column 4). The exact value of the energy (up to the size of the discretization of the solution space) is given by the Full Configuration Interaction (FCI – column 2) which basically corresponds to finding the minimum eigenvalue of a factorial size symmetric matrix. The CCSD(T) (column 5) and HF (column 6) are widely used methods implemented in Gaussian 98 [12] where the results can be retrieved in few seconds. It is clear that imposing the $PQGT1T2$ conditions, one can obtain comparably better energies than any other method.

Finally, Table 5 shows how each component of the SDPARA 1.0.1 (see Section 4) performed on two molecules when changing the number of processors in the high performance computing environment (c). The scalability is given by the quotient between the time per iteration when solving by 1 processor and by 4, 16 and 64 processors, respectively. We can

Table 3: SDP sizes, time, computational environment, and number of processors used by the SDPARA 1.0.1 to solve the SDPs for the electronic structure of different molecules restricted to the PQG and $PQGT1T2$ conditions

Molecule	m	nBLOCK	bLOCKsTRUCT	time (s)	# of CPUs	computer
PQG conditions						
CH_3^+	2964	18	$(128\times1,64\times4,\dots)$	430	1	(a)
NH_4^+	4743	18	$(162\times1,81\times4,\dots)$	1209	1	(a)
BeO	7230	18	$(200\times1,100\times4,\dots)$	3334	4	(a)
HNO	10593	18	$(242\times1,121\times4,\dots)$	5156	2	(a)
BH	15018	18	$(288\times1,144\times4,\dots)$	1481	64	(b)
CH_3N	20709	18	$(338\times1,169\times4,\dots)$	8636	16	(c)
H_2O	27888	18	$(392\times1,196\times4,\dots)$	7984	8	(c)

Molecule	m	nBLOCK	bLOCKsTRUCT	time (s)	# of CPUs	computer
$PQGT1T2$ conditions						
CH_3^+	2964	22	$(736\times2,224\times4,\dots)$	16529	1	(a)
NH_4^+	4743	22	$(1053\times2,324\times4,\dots)$	77614	1	(a)
BeO	7230	22	$(1450\times2,450\times4,\dots)$	78046	4	(a)
HNO	10593	22	$(1936\times2,604\times4,\dots)$	18923	64	(b)
BH	15018	22	$(2520\times2,792\times4,\dots)$	57504	64	(b)
CH_3N	20709	22	$(3211\times2,1014\times4,\dots)$	354518	16	(c)
H_2O	27888	22	$(4018\times2,1274\times4,\dots)$	1764338	8	(c)

Table 4: The ground-state energy calculated by the FCI, CCSD(T) and HF from Gaussian 98 [12], and those from solving SDPs imposing the PQG and $PQGT1T2$ conditions. The FCI column is the reference value and the other columns corresponds to the difference from it. Unit is Hartree $(=4.3598\times10^{-18}$ Joules)

Molecule	FCI	ΔPQG	$\Delta PQGT1T2$	$\Delta CCSD(T)$	ΔHF
CH_3^+	-39.21466	-0.0135	-0.0002	$+0.00017$	$+0.05958$
NH_4^+	-56.48316	-0.0170	-0.0002	$+0.00012$	$+0.08289$
BeO	-89.19956	-0.0131	-0.0017	$+0.00241$	$+0.14788$
HNO	-129.44791	-0.0190	-0.0009	$+0.00123$	$+0.14994$
BH	-25.18765	-0.0065	-0.0001	$+0.00030$	$+0.07398$
CH_3N	-93.88452	-0.0385	-0.0010	$+0.00073$	$+0.15741$
H_2O	-76.15576	-0.0187	-0.0004	$+0.00055$	$+0.14645$

observe an excellent scalability especially for ELEMENTS which is larger than the number of processors since the size of the block matrix n_{max} (the largest block matrix size) is very small compared to the number of constraints m of the SDPs and due to the cache performance. It illustrates well a typical successful application of the SDPARA which can solve large-scale problems with high accuracy.

Table 5: Computation Time per Iteration (CTI) and scalability for molecules LIF and HF imposing the *PQG* conditions for major subroutines of the SDPARA 1.0.1 when changing the number of processors

m	n_{max}		1 CPU	4 CPUs		16 CPUs		64 CPUs	
			CTI (s)	CTI (s)	scal.	CTI (s)	scal.	CTI (s)	scal.
10593	242	ELEMENTS	212	57	3.7	10.6	20.0	2.6	82.6
		CHOLESKY	267	41	6.5	12.3	21.6	4.3	62.2
		Total	491	121	4.1	33.4	14.7	12.5	39.2
15018	288	ELEMENTS	724	151	3.8	23.6	24.5	7.9	72.9
		CHOLESKY	576	117	6.2	32.8	22.1	11.1	65.5
		Total	1637	308	5.3	85.0	19.3	37.3	43.9

6. SDPA Online Solver

Solving large-scale optimization problems requires a huge amount of computational power as shown in Section 5. The SDPARA on the parallel computer is a very useful tool for solving large-scale SDPs. However, not so many users have and/or log on parallel computers. Users must download some source codes from the SDPA Web site and compile them by predetermined procedures even if they have parallel environments. For that reason, we have developed a grid portal system for some software packages of the SDPA Family. We call the system the *SDPA Online Solver*. One can access it from the following Web site:

http://sdpa.indsys.chuo-u.ac.jp/portal/

Users can easily use the SDPA, the SDPARA and the SDPARA-C without their own parallel computers. Users only have to prepare one Web client PC connected with the Internet. Here, we briefly explain how users log on the SDPA Online Solver and solve SDPs by using software packages.

1. Users register with the SDPA Online Solver. The system generates their ID and password, and returns them to users.
2. Users log on the system and upload the parameter file (if necessary) and the data file. Then, select several parameters such as the software package they want to use, the parallel computer, and the number of CPUs when they use the SDPARA and the SDPARA-C (Figure 3).
3. Users push the execute button. After the execution of the software packages, they can download and browse the output file. They can also receive an e-mail to be informed that the execution of the software was completed.

Figure 4 illustrates the mechanism of the SDPA Online Solver on the grid environment. First, a user log on the SDPA Online Solver and the authentication process is done. The Web server composed of Apache [7](HTTP server), PHP [8](hypertext preprocessor), etc. provides a GUI (graphical user interface) for users. The database software MySQL [9] manages information of each user, which is composed of user's ID name, password, uploaded problems and output files. Next, the Web server calls the Ninf GridPRC system (Ninf-G and Ninf-1) and sends meta data of parameters and the uploaded problem to grid resources across the wide-area network like the Internet. Ninf-G [10] [27] on Globus Toolkit [11] is a reference im-

---

[7]http://httpd.apache.org/

[8]http://www.php.net/

[9]http://www.mysql.com

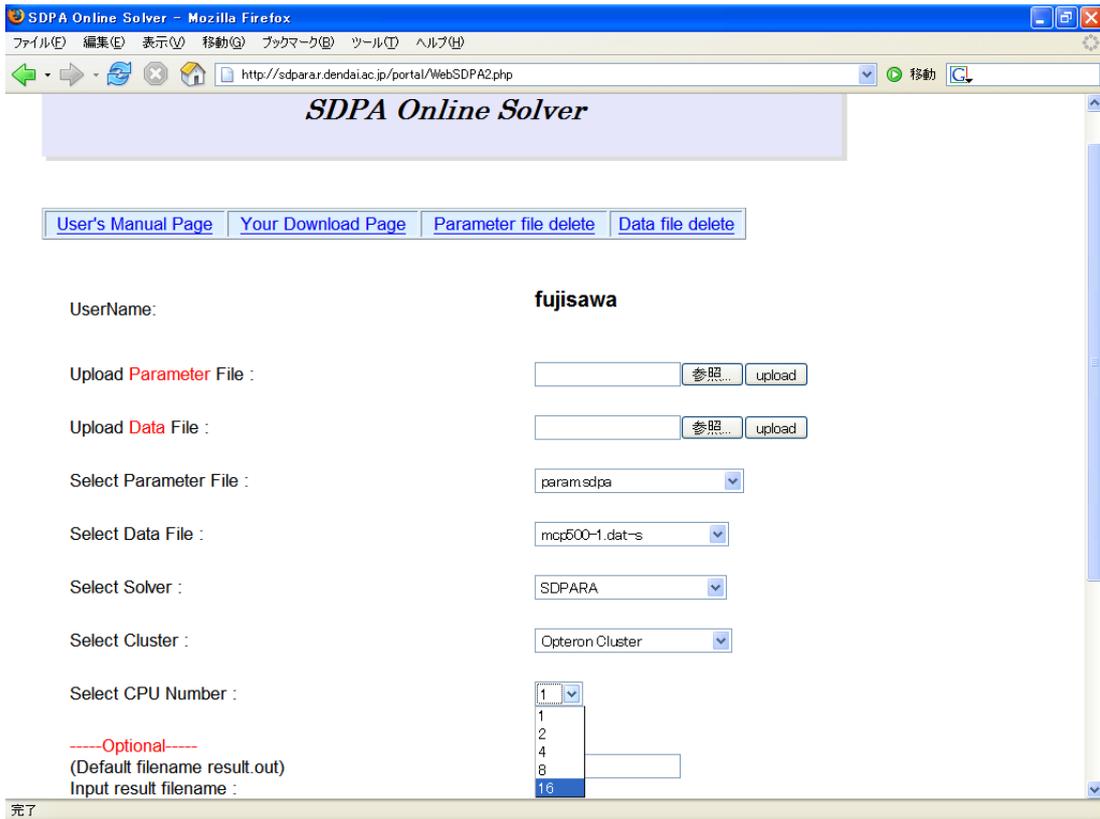[10]http://ninf.apgrid.org/

[11]http://www.globus.org/

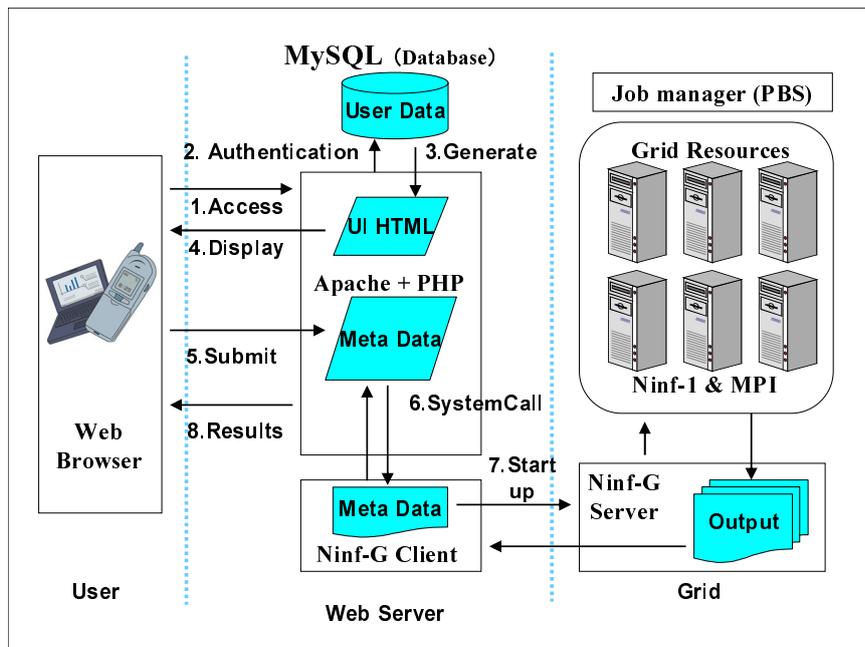Figure 3: Parameter selections of the SDPA Online Solver

Figure 4: The mechanism of the SDPA Online Solver

plementation of GridRPC API. The Ninf-G client program can invoke server programs on remote resources. The Ninf-1 has been developed as an initial product of Ninf-G. Ninf-1 provides a client program similar to the Ninf-G. However, Ninf-1 has no mechanism to support Grid security service. Namely, the Ninf-G is a full re-implementation of the Ninf-1 using Globus Toolkit. We usually use Ninf-G and Ninf-1 when the client program communicates with the server program on the wide-area network and the local-area network, respectively.

As mentioned above, the Web server starts up a Ninf-G client program associated with user's selections, and sends the problem to a Ninf-G server program over the Internet. Then, the Ninf-G server plays the role of a Ninf-1 client, which calls Ninf-1 servers to execute the SDPA, the SDPARA, or the SDPARA-C on the PC cluster. After the execution of the software on the PC cluster, the result is returned to the Web server following the reverse route. This grid portal system for solving large-scale SDPs enables users with few computational resources to receive benefits of parallel computing.

7. Future Works

This paper outlined the SDPA project which implemented the PDIPM for SDPs. We provide several software packages for users in optimization areas. There is still plenty of rooms for improvements for the software packages of the SDPA Family. Here we discuss some future works on the SDPA project.

1. For extremely sparse problems:
   We mentioned that the Schur Complement Matrix (SCM) is fully dense in general even if data matrices are sparse in Section 2.3. However, we found that the SCM often becomes sparse when we solve extremely sparse problems arising from the polynomial optimization [15, 18]. We need to modify the data structure for the sparse SCM, and employ the sparse Cholesky factorization.
2. For good numerical stability:
   We may adopt the multiple-precision arithmetic when the condition numbers of some matrices at any iterate become ill-conditioned. We also consider the iterative refinement method for the residual correction of linear equations.

In addition, we have several plans such as implementation of the IPM for the Second-Order Cone Programs, executing the SDPARA on the Grid environment, and rebuilding the SDPA Online Solver with some recent grid computing technologies.

References

[1] F. Alizadeh, J.-P.A. Haeberly and M.L. Overton: Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results. *SIAM Journal on Optimization*, **8** (1998), 746–768.

[2] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker and R.C. Whaley: *ScaLA-PACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, 1997).

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

[4] B. Borchers: CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, **11 & 12** (1999), 613–623.

[5] S. Burer and R. Monteiro: A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, **95** (2003), 329–357.

[6] E. Cancès, G. Stoltz and M. Lewin: The electronic ground-state energy problem: A new reduced density matrix approach. *Journal on Chemical Physics*, **125** (2006), 064101.

[7] K. Fujisawa, M. Fukuda, M. Kojima and N. Nakata: Numerical evaluation of SDPA (SemiDefinite Programming Algorithm). In H. Frenk, K. Roos, T. Terlaky and S. Zhang (eds.): *High Performance Optimization* (Kluwer Academic Publishers, Massachusetts, 1999).

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

[9] K. Fujisawa, Y. Futakata, M. Kojima, S. Matsuyama, S. Nakamura, K. Nakata and M. Yamashita: SDPA-M (SemiDefinite Programming Algorithm in MATLAB) User's manual — Version 2.00. Research Report B-359, Dept. Math. & Comp. Sciences, Tokyo Institute of Technology, January 2000, revised May 2005.

[10] 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.

[11] M. Fukuda, B.J. Braams, M. Nakata, M.L. Overton, J.K. Percus, M. Yamashita and Z. Zhao: Large-scale semidefinite programs in electronic structure calculation. *Mathematical Programming Series B*, **109** (2007), 553–580.

[12] M.J. Frisch, G.W. Trucks, H.B. Schlegel, et al.: *Gaussian 98, Revision A.11.3* (Gaussian, Inc., Pittsburgh, 2002).

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

[14] 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.

[15] K. Kobayashi, S. Kim and M. Kojima: Correlative sparsity in primal-dual interior-point methods for LP, SDP and SOCP. Research Report B-434, Dept. Math. & Comput. Sciences, Tokyo Institute of Technology, September 2006.

[16] M. Kočvara and M. Stingl: PENNON: A code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, **18** (2003), 317–333.

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

[18] J.B. Lasserre: Global optimization with polynomials and the problems of moments. *SIAM Journal on Optimization*, **11** (2001), 796–817.

[19] D.A. Mazziotti: Realization of quantum chemistry without wave functions through first-order semidefinite programming. *Physical Review Letters*, **93** (2004), 213001.

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

[21] 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.

[22] 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.

[23] K. Nakata, M. Yamashita, K. Fujisawa and M. Kojima: A parallel primal-dual interior-point method for semidefinite programs using positive definite matrix completion. *Parallel Computing*, **32** (2006), 24–43.

[24] M.V. Nayakkankuppam: Solving large-scale semidefinite programs in parallel. *Mathematical Programming Series B*, **109** (2007), 477–504.

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

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

[27] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura and S. Matsuoka: Ninf-G : a reference implementation of RPC-based programming middleware for grid computing. *Journal of Grid Computing*, **1** (2003), 41–51.

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

[29] 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.

[30] L. Vandenberghe and S. Boyd: Positive-definite programming. In J. R. Birge and K. G. Murty (ed.): *Mathematical Programming: State of the Art 1994* (University of Michigan, Michigan, 1994).

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

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

[33] 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.

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

Katsuki Fujisawa
Department of Industrial and Systems Engineering
Chuo University
1-13-27, Kasuga, Bunkyo, Tokyo 112-8551, Japan
E-mail: `fujisawa@indsys.chuo-u.ac.jp`