ISSN 1342-2804

# Research Reports on Mathematical and Computing Sciences

Fast implementation for semidefinite programs with positive matrix completion

Makoto Yamashita, and Kazuhide Nakata

October 2013, B–474

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

# Fast implementation for semidefinite programs with positive matrix completion

Makoto Yamashita [1], Kazuhide Nakata [2]

October 25, 2013

**Abstract**

Solving semidefinite programs (SDP) in a short time is the key to managing various mathematical optimization problems in practical time. The matrix-completion primal-dual interior-point method (MC-PDIPM) extracts a structural sparsity of input SDP by factorizing the variable matrices, and it shrinks the computation time. In this paper, we propose a new factorization based on the inverse of the variable matrix to enhance the performance of the MC-PDIPM. We also combine multithreaded parallel computing to resolve the major bottlenecks in the MC-PDIPM. The numerical results show that the new factorization and the multithreaded computing successfully reduce the computation time for the SDPs that posses the structural sparsity.

**Keywords**

Semidefinite programs, Interior-point methods, Matrix completion, Multithreaded computing

**2010 Mathematics Subject Classification**

90 Operations research, mathematical programming, 90C22 Semidefinite programming, 90C51 Interior-point methods, 97N80 Mathematical software, computer programs.

## 1 Introduction

SemiDefinite Program (SDP) is considered to be a fundamental problem in mathematical optimization. The range of its applications covers from combinatorial optimization [9] to quantum chemistry [7, 21], and sensor network localization problems [4]. More applications can be found at Todd's survey paper [24], and the range is still expanding. Hence, solving SDPs in a short time is the key to managing such applications. The primal-dual interior-point method (PDIPM) [1, 11, 15, 18, 22] is often employed since it can solve SDPs in a polynomial time. Many solvers have been developed based on the PDIPM, for example, SDPA [26], CSDP [5], SeDuMi [23], and SDPT3 [25]. An integration with parallel computing [27] enables us to solve large-scale SDPs arising from practical applications.

A central difficulty of the PDIPM is that the primal variable matrix $\boldsymbol{X}$ must be handled as a fully-dense matrix even when all the input data matrices $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_m$ are considerably sparse. The standard form discussed in this paper is this primal-dual pair.

$$
\begin{array}{lll}
 & \min & : \ \boldsymbol{A}_0 \bullet \boldsymbol{X} \\
(\mathcal{P}) & \text{subject to} & : \ \boldsymbol{A}_k \bullet \boldsymbol{X} = b_k \quad (k = 1, \ldots, m) \\
 & & : \ \boldsymbol{X} \succeq \boldsymbol{O} \\
 & \max & : \ \sum_{k=1}^m b_k z_k \\
(\mathcal{D}) & \text{subject to} & : \ \sum_{k=1}^m \boldsymbol{A}_k z_k + \boldsymbol{Y} = \boldsymbol{A}_0 \\
 & & : \ \boldsymbol{Y} \succeq \boldsymbol{O}
\end{array}
$$

[1] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1-W8-29 Ookayama, Meguro-ku, Tokyo 152-8552, Japan (Makoto.Yamashita@is.titech.ac.jp). The work of the first author was financially supported by the Sasakawa Scientific Research Grant from The Japan Science Society.

[2] Department of Industrial Engineering and Management, Tokyo Institute of Technology, 2-12-1-W9-60 Ookayama, Meguro-ku, Tokyo 152-8552, Japan (nakata.k.ac@m.titech.ac.jp). The work of the second author was partially supported by Grant-in-Aid for Young Scientists (B) 22710136.

Let $\mathbb{S}^n$ be the space of $n \times n$ symmetric matrices. The symbol $\boldsymbol{X} \succeq \boldsymbol{O}(\boldsymbol{X} \succ \boldsymbol{O})$ indicates that $\boldsymbol{X} \in \mathbb{S}^n$ is a positive semidefinite(definite) matrix. The notation $\boldsymbol{U} \bullet \boldsymbol{V}$ is the inner-product between $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{S}^n$ defined by $\boldsymbol{U} \bullet \boldsymbol{V} = \sum_{i=1}^n \sum_{j=1}^n U_{ij} V_{ij}$. The input data are $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_m \in \mathbb{S}^n$ and $b_1, \ldots, b_m \in \mathbb{R}$. The variable in the primal problem $(\mathcal{P})$ is $\boldsymbol{X} \in \mathbb{S}^n$, while the variable in the dual problem $(\mathcal{D})$ is $\boldsymbol{Y} \in \mathbb{S}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$.

The sparsity of the input matrices directly affects the dual matrix $\boldsymbol{Y} = \boldsymbol{A}_0 - \sum_{k=1}^m \boldsymbol{A}_k z_k$. More precisely, $Y_{ij}$ can be nonzero only when $(i, j)$ is covered by the *aggregate sparsity pattern* defined by $\mathcal{A} = \{(i, j) : 1 \leq i \leq n, 1 \leq j \leq n, [\boldsymbol{A}_k]_{ij} \neq 0 \text{ for some } k = 0, \ldots, m\}$. Here, $[\boldsymbol{A}_k]_{ij}$ is $(i, j)$th element of $\boldsymbol{A}_k$. Some examples of the aggregate sparsity pattern are illustrated in Figures 3 and 4; they are the aggregate sparsity patterns generated from the SDPs we solved in the numerical experiments.

On the other hand, all the elements of $\boldsymbol{X}$ in the primal problem $(\mathcal{P})$, in general, must be involved to satisfy $\boldsymbol{X} \succeq \boldsymbol{O}$. The matrix-completion primal-dual interior-point method (MC-PDIPM) proposed by [8, 19] showed that the PDIPM can be executed with the factorization of $\boldsymbol{X}$ in the form

$$\boldsymbol{X} = \boldsymbol{L}_1^T \boldsymbol{L}_2^T \cdots \boldsymbol{L}_{\ell-1}^T \boldsymbol{D} \boldsymbol{L}_{\ell-1} \cdots \boldsymbol{L}_2 \boldsymbol{L}_1 \tag{1}$$

where $\boldsymbol{D}$ is a diagonal-block positive semidefinite matrix and $\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_{\ell-1}$ are lower triangular matrices. A remarkable feature of this factorization is that $\boldsymbol{D}$ and $\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_{\ell-1}$ inherit the sparsity of $\mathcal{A}$. When $\mathcal{A}$ is considerably sparse, these matrices are also sparse, hence, the MC-PDIPM has a considerable advantage over handling the fully-dense matrix $\boldsymbol{X}$. The MC-PDIPM was fist implemented in the solver called SDPA-C(SemiDefinite Programming Algorithm with the Completion method), a variation of SDPA [26], and it was reported in [8, 19] that the MC-PDIPM saves much computation cost of solving SDPs with structural sparsity compared to the standard PDIPM.

The main objective of this paper is further acceleration of the MC-PDIPM. A chief bottleneck in the MC-PDIPM is the repeated computation of the form $\boldsymbol{X}\boldsymbol{v}$ for $\boldsymbol{v} \in \mathbb{R}^n$. The original factorization (1) can be summarized as $\boldsymbol{X} = \boldsymbol{L}^T \boldsymbol{D} \boldsymbol{L}$ with the lower triangular matrix $\boldsymbol{L} = \boldsymbol{L}_{\ell-1} \cdots \boldsymbol{L}_2 \boldsymbol{L}_1$. Instead of this factorization, we introduce the Cholesky factorization of the inverse of $\boldsymbol{X}$; $\boldsymbol{X}^{-1} = \widehat{\boldsymbol{L}} \widehat{\boldsymbol{L}}^T$. We show that the lower triangular matrix $\widehat{\boldsymbol{L}}$ directly inherits the sparsity from $\mathcal{A}$. Another obstacle of (1) is that the presence of $\boldsymbol{D}$ is not a standard form of the Cholesky factorization, and the matrix $\boldsymbol{D}$ prevents us to employ software packages for the sparse Cholesky factorization, such as CHOLMOD[6] and MUMPS[2]. Removing $\boldsymbol{D}$ by $\boldsymbol{X}^{-1} = \widehat{\boldsymbol{L}} \widehat{\boldsymbol{L}}^T$ enables us to naturally integrate the MC-PDIPM framework and these packages. Hence we will obtain the result of $\boldsymbol{X}\boldsymbol{v}$ in a more effective way and shrink the computation time of MC-PDIPM.

We also introduce multithreaded parallel computing to this new factorization. Most processors on modern PCs have multiple cores, and we can process some tasks simultaneously on the different cores. A parallel computing of the MC-PDIPM on multiple PCs connected by a local area network was already discussed in [20]. On the contrary, in this paper, we employ different parallel schemes for multithreading on a single PC, since the difference of memory access, between parallel computing with the Message Passing Interface (MPI) protocol on multiple PCs and multithreading on a single PC, strongly affects the performance of parallel computing. In addition, to enhance the performance of multithreading, we control the number of threads involved in our parallel schemes.

Based on the existing version SDPA-C 6.2.1, we have implemented the new version SDPA-C 7.3.8. (The version numbers reflect the versions of SDPA from which SDPA-C branches.) In the numerical experiments, we show that the new SDPA-C 7.3.8 successfully reduces the computation time due to the effectiveness of $\boldsymbol{X}^{-1} = \widehat{\boldsymbol{L}} \widehat{\boldsymbol{L}}^T$. We also show that the multithreaded computation further expands the difference in the computation time between SDPA-C 6.2.1 and 7.3.8.

This paper is organized as follow. In Section 2, we introduce the two preliminary concepts, the positive matrix completion and the PDIPM. Section 3 will be the main part of this paper, where we will describe the new implementation in details. Section 4 presents the numerical results to show its performance. In Section 5, we summarize this paper and discuss future directions.

Throughout this paper, we use $|S|$ to denote the number of elements of the set $S$. For a matrix $\boldsymbol{X}$ and two sets $S, T \subset \{1, \ldots, n\}$, we use the notation $\boldsymbol{X}_{ST}$ to denote the sub-matrix of $\boldsymbol{X}$ that collects the elements of $X_{ij}$ with $i \in S$ and $j \in T$; for example, $\boldsymbol{X}_{\{2,6\},\{3,4\}} = \begin{pmatrix} X_{23} & X_{24} \\ X_{63} & X_{64} \end{pmatrix}$.

## 2 Preliminaries

Here, we briefly describe the basic concepts of the positive matrix completion and the PDIPM. For more details on the two and their relation, refer to [8, 19] and references therein.

### 2.1 Positive Matrix Completion

The positive matrix completion is closely related to the Cholesky factorization of the variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, in the context of the PDIPM framework. When $\boldsymbol{Y} = \boldsymbol{A}_0 - \sum_{k=1}^{m} \boldsymbol{A}_k z_k$ is positive definite, we can apply the Cholesky factorization to obtain the lower triangular matrix $\boldsymbol{N}$ such that $\boldsymbol{Y} = \boldsymbol{N}\boldsymbol{N}^T$. However, this factorization, in general, generates nonzero elements out of $\mathcal{A}$, and this phenomenon is called fill-in. Therefore, $\mathcal{A}$ is not enough to cover all the nonzeros in $\boldsymbol{N}$. It is known that we can prepare a set of appropriate subsets $C_1, \ldots, C_\ell \subset \{1, 2, \ldots, n\}$ so that the set $\mathcal{E} = \cup_{r=1}^{\ell}(C_r \times C_r)$ covers the nonzero positions of $\mathcal{A}$ and the fill-in. These subsets $C_1, \ldots, C_\ell \subset \{1, 2, \ldots, n\}$ are called cliques in this paper due to a relation with graph theory, and they are obtained by the three steps; we permute the rows/columns of $\boldsymbol{Y}$ with an appropriate order like approximation minimum ordering, then we generate a chordal graph from $\mathcal{A}$, and we extract the maximal cliques there as $C_1, \ldots, C_\ell$.

The set $\mathcal{E}$ is called the *extended sparsity pattern*. Throughout this paper, we assume that $\mathcal{E}$ is considerably sparse; $|\mathcal{A}|$ and $|\mathcal{E}|$ are much less than fully-dense case $n^2$, for instance, $|\mathcal{A}| \leq |\mathcal{E}| < 10^{-2} \times n^2$ for large $n$. In addition, we assume for simplicity that $C_1, \ldots, C_\ell \subset \{1, 2, \ldots, n\}$ are sorted in an appropriate order which satisfies the nice property called the running intersection property in [8]. Such an order can be easily derived from the chordal graph.

Grone *et al.* [10] proved that if a given matrix $\overline{\boldsymbol{X}}$ satisfies the positive definite conditions on all the sub-matrices induced by the cliques $C_1, C_2, \ldots, C_\ell$, that is, $\overline{\boldsymbol{X}}$ satisfies $\overline{\boldsymbol{X}}_{C_1C_1} \succ \boldsymbol{O}, \overline{\boldsymbol{X}}_{C_2C_2} \succ \boldsymbol{O}, \ldots, \overline{\boldsymbol{X}}_{C_\ell C_\ell} \succ \boldsymbol{O}$, then $\overline{\boldsymbol{X}}$ can be completed to $\overline{\overline{\boldsymbol{X}}}$ such that $\overline{\overline{\boldsymbol{X}}}_{C_rC_r} = \overline{\boldsymbol{X}}_{C_rC_r}$ for $r = 1, \ldots, \ell$ and the entire matrix $\overline{\overline{\boldsymbol{X}}}$ is positive definite. Furthermore, it was shown in [8] that the explicit formula (2) below completes $\overline{\boldsymbol{X}}$ to the max-determinant completion $\widehat{\boldsymbol{X}}$ which satisfies

$$\det(\widehat{\boldsymbol{X}}) = \max\{\det(\overline{\overline{\boldsymbol{X}}}) : \overline{\overline{\boldsymbol{X}}}_{C_rC_r} = \overline{\boldsymbol{X}}_{C_rC_r} \text{ for } r = 1, \ldots, \ell, \quad \overline{\overline{\boldsymbol{X}}} \succ \boldsymbol{O}\}.$$

The sparse factorization of $\widehat{\boldsymbol{X}}$ from $\overline{\boldsymbol{X}}$ is given by

$$\widehat{\boldsymbol{X}} = \boldsymbol{L}_1^T \boldsymbol{L}_2^T \ldots \boldsymbol{L}_{\ell-1}^T \boldsymbol{D} \boldsymbol{L}_{\ell-1} \ldots \boldsymbol{L}_2 \boldsymbol{L}_1, \tag{2}$$

where $\boldsymbol{L}_1, \boldsymbol{L}_2, \ldots, \boldsymbol{L}_{\ell-1}$ are the triangular lower matrices of form

$$[\boldsymbol{L}_r]_{ij} = \begin{cases} 1 & (i = j) \\ \left[\overline{\boldsymbol{X}}_{U_rU_r}^{-1} \overline{\boldsymbol{X}}_{U_rS_r}\right]_{ij} & (i \in U_r, j \in S_r) \\ 0 & (otherwise) \end{cases} \tag{3}$$

and $\boldsymbol{D}$ is the diagonal-block matrix

$$\boldsymbol{D} = \begin{pmatrix} \boldsymbol{D}_{S_1S_1} & & & \\ & \boldsymbol{D}_{S_2S_2} & & \\ & & \ddots & \\ & & & \boldsymbol{D}_{S_\ell S_\ell} \end{pmatrix} \tag{4}$$

with

$$\begin{aligned} S_r &= C_r \backslash (C_{r+1} \cup C_{r+2} \cup \cdots \cup C_\ell) \quad (r = 1, 2, \ldots, \ell) \\ U_r &= C_r \cap (C_{r+1} \cup C_{r+2} \cup \cdots \cup C_\ell) \quad (r = 1, 2, \ldots, \ell) \end{aligned}$$

and

$$\boldsymbol{D}_{S_rS_r} = \begin{cases} \overline{\boldsymbol{X}}_{S_rS_r} - \overline{\boldsymbol{X}}_{S_rU_r} \overline{\boldsymbol{X}}_{U_rU_r}^{-1} \overline{\boldsymbol{X}}_{U_rS_r} & (r = 1, 2, \ldots, \ell - 1) \\ \overline{\boldsymbol{X}}_{S_\ell S_\ell} & (r = \ell) \end{cases} .$$

It can be shown that the triangular lower matrix $\boldsymbol{L}$ defined by $\boldsymbol{L} = \boldsymbol{L}_{\ell-1} \ldots \boldsymbol{L}_2 \boldsymbol{L}_1$ is usually fully-dense, destructing the structural sparsity of $\mathcal{E}$. Therefore, when we compute $\boldsymbol{w} = \widehat{\boldsymbol{X}}\boldsymbol{v} = \boldsymbol{L}^T \boldsymbol{D} \boldsymbol{L} \boldsymbol{v}$ for some vector $\boldsymbol{v} \in \mathbb{R}^m$, constructing fully-dense $\boldsymbol{L}$ is not an efficient way. We should note that it can be shown the inverse $\boldsymbol{L}^{-1}$ keeps the sparsity of $\mathcal{E}$, that is, $[\boldsymbol{L}^{-1}]_{ij} = 0$ if $(i, j) \notin \mathcal{E}$, and $\boldsymbol{L}^{-1}$ is a lower triangular matrix[19]. Hence, solving the two equations $\boldsymbol{L}^{-1}\boldsymbol{w}_1 = \boldsymbol{v}$ and $\boldsymbol{L}^{-T}\boldsymbol{w} = \boldsymbol{D}\boldsymbol{w}_1$ can be done with the forward/backward substitutions by exploiting the structure of $\mathcal{E}$ more effectively, and can compute $\boldsymbol{w}$ much faster. In addition, we do not need to compose fully-dense $\widehat{\boldsymbol{X}}$ via the multiplication of $\boldsymbol{L}^T \boldsymbol{D} \boldsymbol{L}$. This idea saves the computation cost of PDIPM as discussed in the next subsection.

## 2.2 Primal-Dual Interior-Point Method

This subsection briefly describes the primal-dual interior-point method(PDIPM) and the modification of the computation formula of the PDIPM by the positive matrix completion method.

A basic framework of the PDIPM can be summarized as follow.

**A basic framework of the primal-dual interior-point method**

Step 0 Prepare an initial point $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{z})$ such that $\boldsymbol{X} \succ \boldsymbol{O}$ and $\boldsymbol{Y} \succ \boldsymbol{O}$. Choose parameters $\beta$ from $0 < \beta < 1$ and $\gamma$ from $0 < \gamma < 1$.

Step 1 If $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{z})$ satisfies some stopping criteria, output $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{z})$ as a solution and terminate.

Step 2 Compute a search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$ based on a modified Newton method.

Step 3 Compute the maximum step length $\alpha_p$ and $\alpha_d$

$$\begin{aligned} \alpha_p &= \max\{\alpha \in (0, 1] : \boldsymbol{X} + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}\} \\ \alpha_d &= \max\{\alpha \in (0, 1] : \boldsymbol{Y} + \alpha d\boldsymbol{Y} \succeq \boldsymbol{O}\}. \end{aligned} \tag{5}$$

Step 4 Update $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{z})$ with $(\boldsymbol{X} + \gamma\alpha_p d\boldsymbol{X}, \boldsymbol{Y} + \gamma\alpha_d d\boldsymbol{Y}, \boldsymbol{z} + \gamma\alpha_d d\boldsymbol{z})$. Go to Step 1.

The heaviest computation in the above framework is usually the computation of the search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$. When we employ the HKM direction [11, 15, 18], the search direction can be obtained by the following system,

$$\begin{aligned} \boldsymbol{B}d\boldsymbol{z} &= \boldsymbol{g} && (6)\\ d\boldsymbol{Y} &= \boldsymbol{G} - \sum_{k=1}^{m} \boldsymbol{A}_k dz_k \\ \widehat{d\boldsymbol{X}} &= \beta\mu\boldsymbol{Y}^{-1} - \boldsymbol{X} - \boldsymbol{X}d\boldsymbol{Y}\boldsymbol{Y}^{-1}, d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^T)/2 && (7) \end{aligned}$$

where

$$\begin{aligned} B_{ij} &= (\boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j \quad (i = 1, \ldots, m,\ j = 1, \ldots, m) && (8)\\ g_k &= \boldsymbol{A}_k \bullet (\beta\mu\boldsymbol{Y}^{-1} - \boldsymbol{X} - \boldsymbol{X}\boldsymbol{G}\boldsymbol{Y}^{-1}) \quad (k = 1, \ldots, m) \end{aligned}$$

with $\mu = \frac{\boldsymbol{X}\bullet\boldsymbol{Y}}{n}$, $\boldsymbol{G} = \boldsymbol{A}_0 - \sum_{k=1}^{m} \boldsymbol{A}_k z_k$. The linear system (6) is often called the *Schur complement equation*(SCE) and its coefficient matrix, evaluated by (8), is called the *Schur complement matrix*(SCM). We first solve the SCE (6) to obtain $d\boldsymbol{z}$, and then we compute $d\boldsymbol{Y}$ and $d\boldsymbol{X}$.

The matrix completion (1) enables us to replace the fully-dense matrices $\widehat{\boldsymbol{X}}$ and $\boldsymbol{Y}^{-1}$ with their sparse matrices in the above computation. As pointed out in [28], one of the main computation bottleneck is the evaluation of SCM $\boldsymbol{B}$. From the property of inner-product, the change from $\boldsymbol{X}$ to $\widehat{\boldsymbol{X}}$ in the formula (8) does not affect $B_{ij}$, therefore, its formula can be transformed into

$$\begin{aligned} B_{ij} &= (\widehat{\boldsymbol{X}}\boldsymbol{A}_i\boldsymbol{Y}^{-1}) \bullet \boldsymbol{A}_j \\ &= \sum_{k=1}^{m} (\boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}\boldsymbol{e}_k)^T \boldsymbol{A}_i (\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_j]_{*k}) && (9) \end{aligned}$$

where $\boldsymbol{e}_k$ and $[\boldsymbol{A}_j]_{*k}$ are the $k$th columns of $\boldsymbol{I}$ and $\boldsymbol{A}_j$, respectively.

In addition, we modify the computation of the primal search direction $d\boldsymbol{X}$ by evaluating its auxiliary matrix $\widehat{d\boldsymbol{X}}$ in column-wise.

$$\begin{aligned} [\widehat{d\boldsymbol{X}}]_{*k} &= \beta\mu\boldsymbol{Y}^{-1}\boldsymbol{e}_k - \widehat{\boldsymbol{X}}\boldsymbol{e}_k - \widehat{\boldsymbol{X}}d\boldsymbol{Y}\boldsymbol{Y}^{-1}\boldsymbol{e}_k \\ &= \beta\mu\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k - \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}\boldsymbol{e}_k - \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}d\boldsymbol{Y}\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k && (10) \end{aligned}$$

As pointed out in the previous subsection, we can avoid the the fully-dense matrices $\widehat{\boldsymbol{X}}$ and $\boldsymbol{Y}^{-1}$ in (9) and (10) by solving the linear equations that involve the sparse matrices $\boldsymbol{L}^{-1}$ and $\boldsymbol{N}$.The computation of the step length $\alpha_p$ in (5) is also decomposed into the sub-matrices

$$\widehat{\alpha}_p = \min_{r=1,2,\ldots,\ell} \max\{\alpha \in (0, 1] : \overline{\boldsymbol{X}}_{C_rC_r} + \alpha d\boldsymbol{X}_{C_rC_r} \succeq \boldsymbol{O}\}, \tag{11}$$

so that $\overline{\boldsymbol{X}}_{C_rC_r} + \widehat{\alpha}_p d\boldsymbol{X}_{C_rC_r}$ is positive definite for $r = 1, \ldots, \ell$ enough to be completed to the positive definite matrix $\widehat{\boldsymbol{X}}$.

The numerical results in [19] indicated that the remove of the fully-dense matrices $\widehat{\boldsymbol{X}}$ and $\boldsymbol{Y}^{-1}$ makes the MC-PDIPM run more effectively than the standard PDIPM (*i.e.* a PDIPM which does not use the positive matrix completion method) for some types of SDP that have the structural sparsity in $\mathcal{E}$.

# 3 Fast implementation in the matrix-completion primal-dual interior-point method

The MC-PDIPM was first implemented in the solver SDPA-C 5[19]. Along with the update of SDPA based on the standard PDIPM to version 6, SDPA-C was also updated to SDPA-C 6. SDPA-C 6 utilized the BLAS (Basic Linear Algebra Subprograms) library [16] to accelerate the linear algebra computation involved in the MC-PDIPM.

The new SDPA-C, version 7.3.8, in this paper further reduces the computation time of the latest version 6.2.1. In this section, we describe the new features of SDPA-C 7.3.8; the change in the factorization of $\widehat{\boldsymbol{X}}$, and the multithreaded parallel computing for the SCM $\boldsymbol{B}$ and the primal auxiliary direction $\widehat{d\boldsymbol{X}}$. In this paper, we abbreviate SDPA-C 6.2.1 and SDPA-C 7.3.8 to SDPA-C 6 and SDPA-C 7, respectively.

## 3.1 New Factorization of the Completed Matrix

The factorization of $\widehat{\boldsymbol{X}}$ into $\widehat{\boldsymbol{X}} = \boldsymbol{L}^T\boldsymbol{D}\boldsymbol{L}$ is not a standard Cholesky factorization due to the diagonal-block matrix $\boldsymbol{D}$. Hence we could not employ software packages for the sparse Cholesky factorization. The completed matrix $\widehat{\boldsymbol{X}}$ is usually fully-dense, while the sparsity of its inverse $\widehat{\boldsymbol{X}}^{-1}$ inherits the structure of $\mathcal{E}$, *i.e.*, $[\widehat{\boldsymbol{X}}^{-1}]_{ij} = 0$ for $(i,j) \notin \mathcal{E}$. Therefore, we focus $\widehat{\boldsymbol{X}}^{-1}$ rather than $\widehat{\boldsymbol{X}}$ and introduce the new factorization of form $\widehat{\boldsymbol{X}}^{-1} = \widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^T$ with the lower-triangular matrix $\widehat{\boldsymbol{L}}$. We want to emphasize here that $\widehat{\boldsymbol{L}}$ also inherits the structure of $\mathcal{E}$. In this subsection, we show that we can obtain the factorized matrix $\widehat{\boldsymbol{L}}$ from $\overline{\boldsymbol{X}}$ in an efficient way using the structure of $S_r$ and $C_r$ $(r = 1, \ldots, \ell)$.

The algorithm to obtain $\widehat{\boldsymbol{L}}$ can be summarized as Algorithm 1. The input of this algorithm is $\overline{\boldsymbol{X}}$, and since $\overline{\boldsymbol{X}}$ should be completed to the positive definite matrix $\widehat{\boldsymbol{X}}$, we suppose that $\overline{\boldsymbol{X}}_{C_rC_r} \succ \boldsymbol{O}$ $(r = 1, \ldots, \ell)$. The validity of algorithm will be discussed later.

**Algorithm 1: An efficient algorithm to obtain the Cholesky factorization of the inverse of the completed matrix**

Step 1 Initialize the memory space for $\widehat{\boldsymbol{L}}$ by $\mathcal{E} = \cup_{r=1}^{\ell}(C_r \times C_r)$.

Step 2 For $r = 1, \ldots, \ell$, apply the Cholesky factorization to $\overline{\boldsymbol{X}}^{-1}_{C_rC_r}$ to obtain the lower triangular matrix $\boldsymbol{L}_r$ that satisfies $\overline{\boldsymbol{X}}^{-1}_{C_rC_r} = \boldsymbol{L}_r\boldsymbol{L}_r^T$. To avoid computing $\overline{\boldsymbol{X}}^{-1}_{C_rC_r}$, we use the following steps.

Step 2-1 Let $\boldsymbol{P}_r$ be the permutation matrix of the dimension $|C_r| \times |C_r|$ with

$$\begin{cases} [\boldsymbol{P}_r]_{ij} = 1 & \text{if} \quad i + j = |C_r| + 1 \\ [\boldsymbol{P}_r]_{ij} = 0 & \text{otherwise} \end{cases}$$

so that $\boldsymbol{P}_r\overline{\boldsymbol{X}}_{C_rC_r}\boldsymbol{P}_r^T$ has the inverse row/column order of $\overline{\boldsymbol{X}}_{C_rC_r}$.

Step 2-2 Apply the Cholesky factorization to $\boldsymbol{P}_r\overline{\boldsymbol{X}}_{C_rC_r}\boldsymbol{P}_r^T$ to obtain the lower triangular matrix $\boldsymbol{M}_r$ that satisfies $\boldsymbol{P}_r\overline{\boldsymbol{X}}_{C_rC_r}\boldsymbol{P}_r^T = \boldsymbol{M}_r\boldsymbol{M}_r^T$.

Step 2-3 Let $\boldsymbol{L}_r$ be $\boldsymbol{P}_r\boldsymbol{M}_r^{-T}\boldsymbol{P}_r^T$.

Step 3 For $r = 1, \ldots, \ell$, put the first $|S_r|$ columns of $\boldsymbol{L}_r$ to the memory space of $\widehat{\boldsymbol{L}}_{C_rS_r}$.

Algorithm 1 requires neither the fully-dense $\widehat{\boldsymbol{X}}$ nor its inverse $\widehat{\boldsymbol{X}}^{-1}$. In addition, since most computation is devoted to the Cholesky factorization of $\boldsymbol{P}_r\overline{\boldsymbol{X}}_{C_rC_r}\boldsymbol{P}_r^T$, we can expect considerable reduction in the computation time when the extended sparsity pattern $\mathcal{E}$ is decomposed into small $C_1, C_2, \ldots, C_\ell$. Furthermore, Algorithm 1 assures that all the nonzero elements of $\widehat{\boldsymbol{L}}$ appear only in $\mathcal{E}$.

*Proof of Algorithm 1*:

We prove the validity of Algorithm 1 based on Lemma 2.6 of [8]. For simplicity, we here focus the fist clique $C_1$ and wrap up the other clique into $C_2' := \cup_{r=2}^{\ell} C_r$. The other cliques $C_2, \ldots, C_r$ can be handled in the same way by the induction on the number of cliques due to the running intersection property. In addition, from this property, we can suppose that $i < j$ for $i \in C_1$ and $j \in C_2'\backslash C_1$. For $\overline{\boldsymbol{X}} \in \mathbb{S}^n$, we decompose $\{1, 2, \ldots, n\}$ into three sets $S = C_1\backslash C_2', U = C_1 \cap C_2'$ and $T = C_2'\backslash C_1$. Note that the extended sparsity pattern $\mathcal{E}$ of $\overline{\boldsymbol{X}}$ is covered by $((S \cup U) \times (S \cup U)) \cup ((U \cup T) \times (U \cup T))$. Hence, the situation we consider here is that $\overline{\boldsymbol{X}}$ is of form

$$\overline{\boldsymbol{X}} = \begin{pmatrix} \overline{\boldsymbol{X}}_{SS} & \overline{\boldsymbol{X}}_{SU} & ? \\ \overline{\boldsymbol{X}}_{US} & \overline{\boldsymbol{X}}_{UU} & \overline{\boldsymbol{X}}_{ST} \\ ? & \overline{\boldsymbol{X}}_{SU} & \overline{\boldsymbol{X}}_{TT} \end{pmatrix}$$

with unknown elements ? in the position $(S \times T) \cup (T \times S)$, and the sub-matrices induced by the cliques $C_1, C_2'$ are positive definite,

$$\overline{\boldsymbol{X}}_{C_1C_1} = \begin{pmatrix} \overline{\boldsymbol{X}}_{SS} & \overline{\boldsymbol{X}}_{SU} \\ \overline{\boldsymbol{X}}_{US} & \overline{\boldsymbol{X}}_{UU} \end{pmatrix} \succ \boldsymbol{O}, \quad \overline{\boldsymbol{X}}_{C_2'C_2'} = \begin{pmatrix} \overline{\boldsymbol{X}}_{UU} & \overline{\boldsymbol{X}}_{UT} \\ \overline{\boldsymbol{X}}_{TU} & \overline{\boldsymbol{X}}_{TT} \end{pmatrix} \succ \boldsymbol{O}.$$

Note that $C_1 = S \cup U$ and $C_2' = U \cup T$ in this situation. Lemma 2.6 of [8] claims that $\overline{\boldsymbol{X}}$ is completed to the max-determinant positive definite matrix $\widehat{\boldsymbol{X}}$ of form

$$\widehat{\boldsymbol{X}} = \begin{pmatrix} \overline{\boldsymbol{X}}_{SS} & \overline{\boldsymbol{X}}_{SU} & \overline{\boldsymbol{X}}_{SU}\overline{\boldsymbol{X}}_{UU}^{-1}\overline{\boldsymbol{X}}_{UT} \\ \overline{\boldsymbol{X}}_{US} & \overline{\boldsymbol{X}}_{UU} & \overline{\boldsymbol{X}}_{ST} \\ \overline{\boldsymbol{X}}_{TU}\overline{\boldsymbol{X}}_{UU}^{-1}\overline{\boldsymbol{X}}_{US} & \overline{\boldsymbol{X}}_{SU} & \overline{\boldsymbol{X}}_{TT} \end{pmatrix} \succ \boldsymbol{O}.$$

Hence, the validity of Algorithm 1 is now reduced to the proof of $\widehat{\boldsymbol{X}} = \widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}$. In Step 2, we factorize the inverse of the positive definite sub-matrices into the lower triangular matrices by the Cholesky factorization as follow.

$$\begin{aligned} \begin{pmatrix} \overline{\boldsymbol{X}}_{SS} & \overline{\boldsymbol{X}}_{SU} \\ \overline{\boldsymbol{X}}_{US} & \overline{\boldsymbol{X}}_{UU} \end{pmatrix}^{-1} &= \begin{pmatrix} \boldsymbol{M}_{SS} & \\ \boldsymbol{M}_{US} & \boldsymbol{M}_{UU} \end{pmatrix} \begin{pmatrix} \boldsymbol{M}_{SS}^T & \boldsymbol{M}_{US}^T \\ & \boldsymbol{M}_{UU}^T \end{pmatrix} \\ \begin{pmatrix} \overline{\boldsymbol{X}}_{UU} & \overline{\boldsymbol{X}}_{UT} \\ \overline{\boldsymbol{X}}_{TU} & \overline{\boldsymbol{X}}_{TT} \end{pmatrix}^{-1} &= \begin{pmatrix} \boldsymbol{N}_{UU} & \\ \boldsymbol{N}_{TU} & \boldsymbol{N}_{TT} \end{pmatrix} \begin{pmatrix} \boldsymbol{N}_{UU}^T & \boldsymbol{N}_{TU}^T \\ & \boldsymbol{N}_{TT}^T \end{pmatrix}. \end{aligned}$$

Since the matrices in the left-hand side are positive definite, we can take the inverse of components of the right-hand side, *e.g.*, $\boldsymbol{M}_{SS}^{-1}$. By comparing the elements of both-sides, we obtain

$$\left\{ \begin{array}{lcl} \overline{\boldsymbol{X}}_{SS} &=& \boldsymbol{M}_{SS}^{-T}\boldsymbol{M}_{SS}^{-1} + \boldsymbol{M}_{SS}^{-T}\boldsymbol{M}_{US}^T\boldsymbol{M}_{UU}^{-T}\boldsymbol{M}_{UU}^{-1}\boldsymbol{M}_{US}\boldsymbol{M}_{SS}^{-1} \\ \overline{\boldsymbol{X}}_{SU} &=& -\boldsymbol{M}_{SS}^{-T}\boldsymbol{M}_{US}^T\boldsymbol{M}_{UU}^{-T}\boldsymbol{M}_{UU}^{-1} \\ \overline{\boldsymbol{X}}_{UU} &=& \boldsymbol{M}_{UU}^{-T}\boldsymbol{M}_{UU}^{-1} = \boldsymbol{N}_{UU}^{-T}\boldsymbol{N}_{UU}^{-1} + \boldsymbol{N}_{UU}^{-T}\boldsymbol{N}_{TU}^T\boldsymbol{N}_{TT}^{-T}\boldsymbol{N}_{TT}^{-1}\boldsymbol{N}_{TU}\boldsymbol{N}_{UU}^{-1} \\ \overline{\boldsymbol{X}}_{UT} &=& -\boldsymbol{N}_{UU}^{-T}\boldsymbol{N}_{TU}^T\boldsymbol{N}_{TT}^{-T}\boldsymbol{N}_{TT}^{-1} \\ \overline{\boldsymbol{X}}_{TT} &=& \boldsymbol{N}_{TT}^{-T}\boldsymbol{N}_{TT}^{-1}. \end{array} \right. \tag{12}$$

In Step 3, the structure of $C_1$ and $C_2'$ locates the elements of the above factorized matrices into $\widehat{\boldsymbol{L}}$ as

$$\widehat{\boldsymbol{L}} = \begin{pmatrix} \boldsymbol{M}_{SS} & & \\ \boldsymbol{M}_{US} & \boldsymbol{N}_{UU} & \\ & \boldsymbol{N}_{TU} & \boldsymbol{N}_{TT} \end{pmatrix}, \tag{13}$$

and because $\widehat{\boldsymbol{L}}$ is lower triangular, its inverse can be explicitly expressed by

$$\widehat{\boldsymbol{L}}^{-1} = \begin{pmatrix} \boldsymbol{M}_{SS}^{-1} & & \\ -\boldsymbol{N}_{UU}^{-1}\boldsymbol{M}_{US}\boldsymbol{M}_{SS}^{-1} & \boldsymbol{N}_{UU}^{-1} & \\ \boldsymbol{N}_{TT}^{-1}\boldsymbol{N}_{TU}\boldsymbol{N}_{UU}^{-1}\boldsymbol{M}_{US}\boldsymbol{M}_{SS}^{-1} & -\boldsymbol{N}_{TT}^{-1}\boldsymbol{N}_{TU}\boldsymbol{N}_{UU}^{-1} & \boldsymbol{N}_{TT}^{-1} \end{pmatrix}.$$

The multiplication $\widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}$ and the relation of (12) leads to $\widehat{\boldsymbol{X}} = \widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}$. This completes the validity of Algorithm 1. □

With this new factorization, the evaluation formula of SCM (9) and the primal auxiliary matrix (10) will be replaced by their efficient ones,

$$B_{ij} = \sum_{k=1}^{m} (\widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}\boldsymbol{e}_k)^T \boldsymbol{A}_i (\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_j]_{*k}) \tag{14}$$

$$\left[\widehat{d\boldsymbol{X}}\right]_{*k} = \beta\mu\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k - \widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}\boldsymbol{e}_k - \widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}d\boldsymbol{Y}\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k. \tag{15}$$

To implement this new factorization based on Algorithm 1 into SDPA-C 7, the five types of the computation related to Cholesky factorization should be employed;

(i) the dense Cholesky factorization for the SCM $\boldsymbol{B}$ and its forward/backward substitution for the SCE(6) if $\boldsymbol{B}$ is fully-dense

(ii) the sparse Cholesky factorization for the SCM $\boldsymbol{B}$ and its forward/backward substitution for the SCE(6) if $\boldsymbol{B}$ is considerable sparse

(iii) the dense Cholesky factorization for the sub-matrices $\overline{\boldsymbol{X}}_{C_1C_1}, \ldots, \overline{\boldsymbol{X}}_{C_\ell C_\ell}$ in Step 2 of Algorithm 1

(iv) the forward/backward substitution of $\widehat{\boldsymbol{L}}$ to solve the linear systems of form $\widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^T\boldsymbol{w} = \boldsymbol{v}$

(v) the sparse Cholesky factorization for the dual variable matrix $\boldsymbol{Y} = \boldsymbol{N}\boldsymbol{N}^T$ and the forward/backward substitution of $\boldsymbol{N}$ to solve the linear systems of form $\boldsymbol{N}\boldsymbol{N}^T\boldsymbol{w} = \boldsymbol{v}$.

For the cases (i) and (ii), the sparsity of the SCM $\boldsymbol{B}$ heavily depends on the types of application that generates the input SDP, as pointed out in [27]. For example, the SDPs generated from quantum chemistry [7, 21] have fully-dense SCMs, while the density of SCMs of SDPs arising from the sensor network localization problems [14] are often less than 1%. Hence, we should choose appropriate software packages for the fully-dense and sparse SCMs. We employed the dense Cholesky factorization routine of LAPACK [3] for (i) and the sparse Cholesky factorization routine of MUMPs [2] for (ii), and we select one of these two routines based on the criteria proposed in [27] with the information from the input SDP. The LAPACK routine is also applied to the case (iii).

For the cases (iv) and (v), we choose CHOLMOD[6] rather than MUMPS[2], since we access the internal data structure of software package in order to locate $\widehat{\boldsymbol{L}}_r$ in the appropriate space of $\widehat{\boldsymbol{L}}$ in Step 3 of Algorithm 1. CHOLMOD internally uses a super-nodal Cholesky factorization. We notice

that the row sets and the column sets of super-nodes computed in CHOLMOD satisfy the running intersection property, hence the row sets and the columns sets can be used as $C_1, C_2, \ldots, C_\ell$ and $S_1, S_2, \ldots, S_\ell$, respectively. CHOLMOD determines the size of super-nodes appropriately using some heuristics like approximate minimum ordering so that the sparse Cholesky factorization and the forward/backward substitution can be processed effectively using the BLAS library. In addition, the structure of memory space allocated for primal $\widehat{\boldsymbol{L}}$ is identical with that of dual $\boldsymbol{N}$ in the MC-PDIPM. Hence, we first obtain the structure of $\boldsymbol{N}$ by constructing the aggregate sparsity pattern $\mathcal{A}$ and applying CHOLMOD to obtain its symbolic sparse Cholesky factorization, then we can extract its super-node information ($C_1, C_2, \ldots, C_\ell$ and $S_1, S_2, \ldots, S_\ell$) to prepare the memory space for $\widehat{\boldsymbol{L}}$.

Table 1 shows a computation time reduction due to the new factorization of $\widehat{\boldsymbol{X}}$. We compare the computation time of SDPA-C 6 with the new SDPA-C 7 on an SDP which is a relaxation of max-clique problem on a lattice graph with the parameters $p = 300$ and $q = 10$. The details of this SDP and the computation environment will be described in Section 4. The dimension of the variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ is $n = p \times q = 3000$, and the extended sparsity pattern $\mathcal{E}$ are decomposed into 438 cliques, $C_1, \ldots, C_{438}$. Since the maximum cardinality of cliques, $\max\{|C_r| : r = 1, \ldots, 438\}$, is only 59, the matrices are decomposed into the small cliques. To specify the computation bottlenecks; we use the following names of three representative bottlenecks, S-ELEMENT is the evaluation time of the SCM elements by (6), (9) or (14), S-CHOLESKY is the appropriate Cholesky factorization routine for the SCM, and P-MATRIX is the computation of the primal auxiliary matrix $\widehat{d\boldsymbol{X}}$ by (7), (10) or (15).

Table 1: Computation time reduction due to the new factorization (the time unit is second.)

| | SDPA-C 6.2.1 | SDPA-C 7.3.8 |
|---|---|---|
| S-ELEMENTS | 4205.70 | 2094.03 |
| S-CHOLESKY | 185.36 | 161.03 |
| P-MATRIX | 316.00 | 242.51 |
| Other | 22.22 | 17.93 |
| Total | 4729.28 | 2515.50 |

Table 1 indicates that the new factorization reduced the evaluation time of SCM (4205 seconds) to half (2094 seconds). The computation time on P-MATRIX was also shrunk from 316 seconds to 242 seconds. Consequently, the new factorization brought us 1.88-times speedup. This is because it does not require the diagonal-block matrix $\boldsymbol{D}$ as the original factorization and it enables us to use the existing efficient package CHOLMOD. As we will show in Section 4, the effect will be amplified for the larger SDPs.

### 3.2 The matrix-completion primal-dual interior-point method with multithreaded parallel computing

To enhance the performance of SDPA-C 7 further, we also take advantage of multithreaded parallel computing. Since the processors on modern PCs have multiple cores (computation unit), we can assign different threads (computation tasks) to the cores and run the multiple tasks simultaneously. For example, multithreaded BLAS libraries are often employed to reduce the time related to linear algebra computation in a variety of numerical optimization problems. However, the effect of multithreaded BLAS libraries is limited to dense linear algebra computation. Hence, we should apply multithreaded parallel computing not only to the dense linear algebra computation but also to the upper level of MC-PDIPM.

We embed multithreaded parallel computing into SDPA-C 7 to resolve the three bottlenecks of MC-PDIPM, S-ELEMENTS, S-CHOLESKY, and P-MATRIX. A parallel computing for these bottlenecks was already examined in SDPARA-C [20] with the MPI(Message Passing Interface) protocol on multiple PCs. To apply multithreaded parallel computing on a single PC, however, we need different parallel schemes. In this section, we use $u$ to denote the number of cores that participate in multithreaded computation.

We start our discussion from S-ELEMENTS. The SCM is evaluated by the formula (14), and we can reuse the results of $\widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}\boldsymbol{e}_k$ and $\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\left[\boldsymbol{A}_j\right]_{*k}$ for all the element in $\left[\boldsymbol{B}\right]_{*j}$, that is, the $j$-th column of $\boldsymbol{B}$. On the other hand, we can not reuse their results for different columns of $\boldsymbol{B}$, since the memory storage of $\widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}\boldsymbol{e}_k$ for all $k = 1, \ldots, n$ is equivalent to hold the fully-dense matrix $\widehat{\boldsymbol{X}}$. It means that we lose the nice sparse structure of $\mathcal{E}$.

Since the computation of each column $\left[\boldsymbol{B}\right]_{*j}$ is independent from the other columns $\{\left[\boldsymbol{B}\right]_{*i} : 1 \leq i \leq m, \quad i \neq j\}$, SDPARA-C simply employs the column-wise distribution, in which the $p$-th thread evaluates the columns assigned by the set $\mathcal{S}_p = \{j : 1 \leq j \leq m, \quad (j-1) \texttt{ mod } u = p-1\}$, where $a \texttt{ mod } b$ stands for the remainder of the division of $a$ by $b$. This simple assignment was necessary for SDPARA-C. since the memory space of $\boldsymbol{B}$ was assumed to be distributed to multiple PCs and we had to fix the column assignments in order not to send the evaluation result on a PC to another PC with heavy network communication. On the contrary, all the threads of multithreading can share the memory space, since they run on a single PC. Hence, we can propose more aggressive parallel schemes as follow to improve the load-balance over all the threads.

**Algorithm 2: Multithreaded parallel computing for the evaluation of SCM**

Step 1 Initialize the SCM $\boldsymbol{B} = \boldsymbol{O}$. Set $\mathcal{S} = \{1, 2, \ldots, m\}$.

Step 2 Generate $u$ threads.

Step 3 For $p = 1, 2, \ldots, u$, run the $p$-th thread on the $p$-th core to execute the following steps:

Step 3-1 If $\mathcal{S} = \emptyset$, terminate.

Step 3-2 Take the smallest element $j$ from $\mathcal{S}$ and update $\mathcal{S} \leftarrow \mathcal{S} \backslash \{j\}$.

Step 3-3 Evaluate $\left[\boldsymbol{B}\right]_{*j}$ by
**for** $k = 1, \ldots, n$
Apply the forward/backward substitution routine of CHOLMOD
to obtain $\boldsymbol{v}_1 := \widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}^{-1}\boldsymbol{e}_k$ and $\boldsymbol{v}_2 := \boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\left[\boldsymbol{A}_j\right]_{*k}$.
**for** $i = 1, \ldots, m$
Update $B_{ij} \leftarrow B_{ij} + \boldsymbol{v}_1^T \boldsymbol{A}_i \boldsymbol{v}_2$.

Figure 1 shows an example of the thread assignment to the SCM $\boldsymbol{B}$ where $\boldsymbol{B} \in \mathbb{S}^8$ and $u = 4$. Note that we evaluate only the lower triangular part of $\boldsymbol{B}$, since $\boldsymbol{B}$ is symmetric. Now we have 4 threads, thus the $p$-th thread evaluates the $p$-th column for $p = 1, 2, 3,$ and 4 in the beginning. Let us assume the computation cost is same over all $B_{ij}$ in Figure 1, then the 4th thread will finish its column evaluation in the shortest time among the 4 threads, so the 4th thread will next evaluate the 5th column. Then, the 3rd thread will finish its first task, and move to the 6th column. On the other hand, if the 4th column requires more computation time than the 3rd column, the 3rd thread will take the 5th column and then the 4th thread will evaluate the 6th column.

The overhead in Algorithm 2 is only Step 3-2 where we should guarantee that only one thread can enter Step 3-2 at the same time. Hence, we can expect Algorithm 2 attains better load-balance than the simple column-wise distribution employed in SDPARA-C. In particular, the main

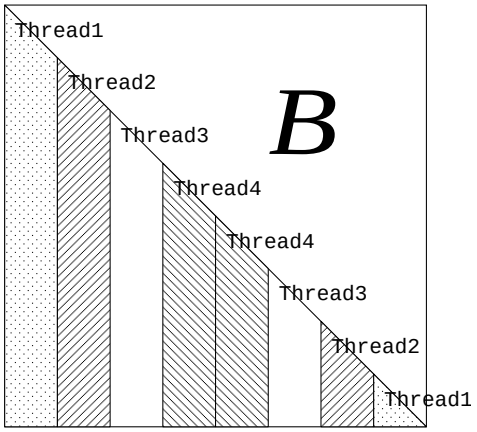


Figure 1: Thread assignment to the Schur complement matrix $\boldsymbol{B}$

computation cost of each column $[\boldsymbol{B}]_{*j}$ is $\widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}\boldsymbol{e}_k$ and $\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_j]_{*k}$, therefore it is almost proportional to the number of nonzero columns of $\boldsymbol{A}_j$. When only a few of $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$ have too-many nonzero columns and the others have only a few, the simple column-wise distribution is difficult to keep nice load-balance. Algorithm 2 can naturally overcome this difficulty.

When we implement Algorithm 2 in the MC-PDIPM, we should pay attention to the number of threads generated by the BLAS library that CHOLMOD internally calls for the forward/backward substitution, $\widehat{\boldsymbol{L}}^{-T}\widehat{\boldsymbol{L}}\boldsymbol{e}_k$ and $\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[A_j]_{*k}$. For example, assume 4 cores are available($u = 4$), if we generate 4 threads in Step 2 of Algorithm 2 and each thread internally generates 4 threads for the BLAS library, then we need to manage 16 threads in total on the 4 cores. The overhead of this management is considerably heavy, and when we implemented the multithreaded parallel computing in this way, SDPA-C took at least 10 times longer computation time than the single-thread computing. Therefore, we turn off the multithreading of the BLAS library before entering the forward/backward substitution routine, and turn it on again after the computation of the routine. This enables us to resolve the thread conflicts.

Now, we move our focus to S-CHOLESKY and P-MATRIX. For S-CHOLESKY, we verified from preliminary experiments that the usage of multithreaded BLAS library is enough for both LAPACK and MUMPS to derive the performance of multithreaded parallel computing. In P-MATRIX, the primal auxiliary matrix $\widehat{d\boldsymbol{X}}$ is evaluated by the formula (15). Since this formula naturally indicates the independence of $\widehat{d\boldsymbol{X}}$ columns, the simple column-wise distribution was employed in SDPARA-C. In the multithreading, all the threads can share the memory space. Hence, we can replace the column-wise distribution with the first-come first-served concept, the same parallel concept as Algorithm 2. We also apply the above scheme to control the number of threads involved in the parallel computing.

Table 2 shows the computation time reduction by the multithreaded parallel computing. We compare the results of the existing SDPA-C 6, the new SDPA-C 7, and SDPARA-C 1 on the same SDP as Table 1. In each bottleneck, the upper row is the computation time, and the lower row is the speed-up ratio compared to a single thread.

In Table 2, SDPA-C 7 with 4 threads reduced S-ELEMENTS from 2094.03 seconds on a single thread to 731.98 seconds, resulting 2.86-times speed-up. The computation time on P-MATRIX was also shrunk from 242.51 seconds to 83.54 seconds, and we obtained 2.90-times speed-up. Consequently, these time reductions brought us the speed-up in the total time from 2515.50 seconds to 889.15, 2.82-times speed-up.

Table 2: Computation time reduction due to the multithreaded computing (the time unit is second.)

| | SDPA-C 6 | SDPARA-C 1 | | | SDPA-C 7 | | |
|---|---|---|---|---|---|---|---|
| threads | 1 | 1 | 2 | 4 | 1 | 2 | 4 |
| S-ELEMENTS | 4205.70 | 3103.51 | 1904.64 | 1309.88 | 2094.03 | 1170.37 | 731.98 |
| | | 1.00x | 1.62x | 2.36x | 1.00x | 1.78x | 2.86x |
| S-CHOLESKY | 185.36 | 312.41 | 110.76 | 65.12 | 161.03 | 85.24 | 47.77 |
| | | 1.00x | 2.82x | 4.79x | 1.00x | 1.88x | 3.37x |
| P-MATRIX | 316.00 | 316.04 | 159.32 | 81.34 | 242.51 | 131.03 | 83.54 |
| | | 1.00x | 1.98x | 3.88x | 1.00x | 1.85x | 2.90x |
| Other | 22.22 | 32.80 | 44.76 | 40.69 | 17.93 | 23.88 | 25.86 |
| Total | 4729.28 | 3764.84 | 2219.48 | 1497.03 | 2515.50 | 1410.52 | 889.15 |
| | | 1.00x | 1.63x | 2.51x | 1.00x | 1.78x | 2.82x |

When we compared SDPA-C 6 that run using only a single thread with SDPA-C 7 with 4 threads, we obtained 5.31-times speed-up. Table 2 indicates the parallel schemes discussed above are effectively integrated into the MC-PDIPM and the new factorization $\widehat{\boldsymbol{X}}^{-1} = \widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^T$.

In addition, SDPARA-C 1 consumed longer time than SDPA-C 7 and this was mainly due to the overhead of MPI protocol. Since the MPI protocol has been designed for multiple PCs, it is not appropriate for a single PC, and multithreaded computation achieves better performance on a single PC. In addition, Algorithm 2 works more effectively in multithreaded computing environment than the simple column-wise distribution of SDPARA-C. This result demonstrates that the speedup of SDPA-C 7 for S-ELEMENTS on 4 threads was 2.86, and it was higher than 2.36, that of SDPARA-C 1.

# 4 Numerical Experiments

In this section, we present some numerical results to evaluate the performance of SDPA-C 7. The computing environment was RedHat Linux with Xeon X5365(3.0GHz, 4cores) and 48GB memory space. We used the three groups of test problems; the max-clique problems over lattice graphs, the max-cut problems over lattice graphs, and the spin-glass problems.

**The max-clique problems over lattice graphs**

We consider a graph $G(V, E)$ with the vertex set $V = \{1, \ldots, n\}$ and the edge set $E \subset V \times V$. A vertex subset $S \subset V$ is called a clique if $(i, j) \in E$ for $\forall i \in S, \forall j \in S$. The max-clique problem is to find a clique which attains the maximum cardinality among all cliques in $G(V, E)$.

Though the max-clique problem itself is NP-hard, Lovász [17] proposed an SDP relaxation method to obtain a good approximation in polynomial time. The SDP problem below gives a good upper bound of the max-clique cardinality for $G(V, E)$,

$$\begin{array}{rcl} \max & : & \boldsymbol{e}\boldsymbol{e}^T \bullet \boldsymbol{X} \\ \text{subject to} & : & \boldsymbol{I} \bullet \boldsymbol{X} = 1 \\ & & \left(\boldsymbol{e}_i\boldsymbol{e}_j^T + \boldsymbol{e}_j\boldsymbol{e}_i^T\right) \bullet \boldsymbol{X} = 0 \quad \text{for} \quad (i,j) \notin E \\ & & \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}$$

For the numerical experiments, we generated SDPs of this type over lattice graphs. A lattice

graph $G(V, E)$ is determined by the two parameters $p$ and $q$, with the vertex set being $V = \{1, 2, \ldots, p \times q\}$ and the edge set $E = \{(i + (j-1) \times p, (i+1) + (j-1) \times p) : i = 1, \ldots, p-1, j = 1, \ldots, q\} \cup \{(i + (j-1) \times p, i + j \times p) : i = 1, \ldots, p, j = 1, \ldots, q-1\}$. An example of lattice graphs is shown in Figure 2, where the parameters are $p = 4, q = 3$.

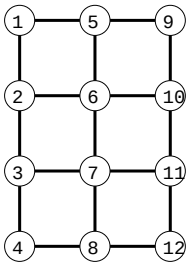


Figure 2: The lattice graph with the parameters $p = 4, q = 3$.

We applied the pre-processing technique proposed in [8] to convert the above SDP into an equivalent SDP which have better sparsity. The aggregate sparsity pattern $\mathcal{A}$ for the max-clique SDP with $p = 300, q = 10$ is illustrated in Figure 3. To draw Figure 3, we applied the approximate minimum degree heuristics to $\mathcal{A}$ and this shows that the structural sparsity embedded in this SDP. Then, the sizes of cliques $C_1, \ldots, C_\ell$ can be much smaller compared to $n = p \times q = 300 \times 10 = 3000$. This $\mathcal{A}$ does not incur any fill-in, that is, $\mathcal{E} = \mathcal{A}$. This SDP was the example SDP solved in Section 3.

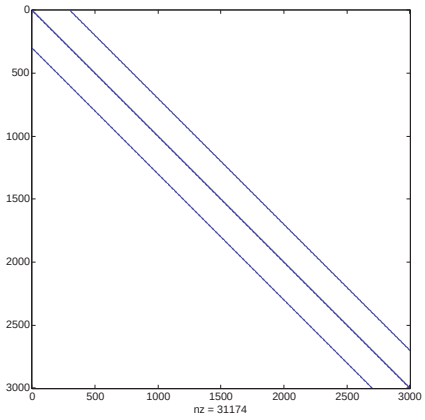


Figure 3: The aggregate sparsity pattern $\mathcal{A}$ for the max-clique SDP with $p = 300, q = 10$.

**The max-cut problems over lattice graphs**

The SDP relaxation method for the max-cut problems due to Goemans and Williamson [9] has been well-known, since it was a starting point of SDP relaxation studies. We consider a graph $G(V, E)$ with the vertex set $V = \{1, \ldots, n\}$ and the edge set $E \subset V \times V$. Each edge $(i, j) \in E$ has the corresponding non-negative weight $w_{ij}$ (for simplicity, $w_{ij} = 0$ if $(i, j) \notin E$). The weight of cut $C \subset V$ is the total weight of edges that traverse between $C$ and $V \backslash C$. The max-cut problem is to find a subset $C$ which maximizes the cut weight,

$$\text{max} \ : \sum_{i \in C, j \in V \backslash C} w_{ij} \qquad \text{subject to} \ : \ C \subset V.$$

An SDP relaxation of this problem is given by

$$\begin{array}{rcl} \min & : & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & : & \left(\boldsymbol{e}_i \boldsymbol{e}_i^T\right) \bullet \boldsymbol{X} = 0 \quad \text{for} \quad i = 1, \ldots, n \\ & & \boldsymbol{X} \succeq \boldsymbol{O}, \end{array}$$

where $\boldsymbol{A}_0 = (-\mathrm{diag}(\boldsymbol{W}\boldsymbol{e}) + \boldsymbol{W})$, $\boldsymbol{W}$ is the matrix whose $(i,j)$ element is $w_{ij}$ for $i = 1, \ldots, n,\ j = 1, \ldots, n$, $\boldsymbol{e}$ is the vector in which all elements are 1, and $\mathrm{diag}(\boldsymbol{w})$ is the diagonal matrix whose diagonal elements are the elements of $\boldsymbol{w}$.

When we generate SDP problems from the max-cut problem over the lattice graphs, the structure of graph will appear in the coefficient matrix the objective function $\boldsymbol{A}_0$. Hence, we can find a similar structure in its aggregate sparsity pattern as Figure 3.

**The spin-glass problems**

The four SDPs of this type were collected as the torus set in the 7th DIMACS benchmark problems[12]. These SDPs arise from the computation for the ground-state energy of Ising spin glasses in quantum chemistry. More information on this energy computation can be found at the web-page of Spin Glass Server [13] and the reference therein.

The Ising spin-glass model has the parameter $p$ (the number of samples), and if generate an SDP from a 3D spin-glass model, then the dimension of the variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ is $n = p^3$. Figure 4 illustrates the aggregate sparsity pattern $\mathcal{A}$ of the spin-glass SDP with $p = 23$ and $n = 23^3 = 12167$.

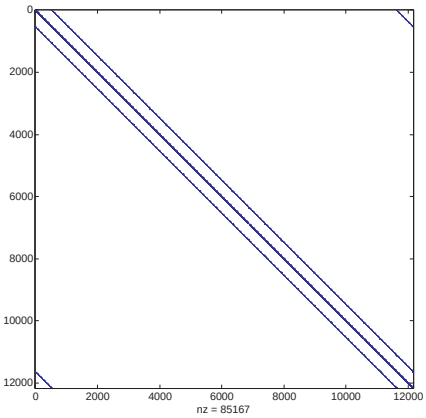


Figure 4: The aggregate sparsity pattern $\mathcal{A}$ of the spin-glass SDP with $p = 23$.

Table 3 summarizes the SDPs we used for the numerical experiments. The first column is the name of SDP and the second $p$ is the parameter to generate the SDP (We fix the parameter $q = 10$ for the max-clique problems and the max-cut problems). The third column $n$ is the dimension of the variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$. The fourth column is the density of aggregate sparsity pattern defined by $\frac{|\mathcal{A}|}{n^2}$. The fifth column $\ell$ is the number of cliques $(C_1, \ldots, C_\ell)$, and the sixth and seventh columns are the average and maximum sizes of the cliques defined by $\frac{\sum_{r=1}^{\ell} |C_r|}{\ell}$ and $\max_{r=1,\ldots,\ell} C_r$, respectively. The eighth(last) column $m$ is the number of input data matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$.

In this section, we compare the computation time of the existing SDPA-C 6 [20], the new SDPA-C 7, and SDPA 7 [26] and SeDuMi 1.3 [23]. The former two implement the MC-PDIPM, while the latter two implement the standard PDIPM. Here, we do not conduct the numerical

Table 3: The size of SDPs for the numerical experiments

| Name | $p$ | $n$ | density | $\ell$ | ave-size | max-size | $m$ |
|---|---|---|---|---|---|---|---|
| MaxClique300 | 300 | 3000 | 0.38% | 348 | 28.36 | 51 | 5691 |
| MaxClique400 | 400 | 4000 | 0.28% | 439 | 29.89 | 59 | 7591 |
| MaxClique500 | 500 | 5000 | 0.23% | 581 | 28.26 | 50 | 9491 |
| MaxCut400 | 400 | 4000 | 0.15% | 1282 | 8.05 | 26 | 4000 |
| MaxCut500 | 500 | 5000 | 0.12% | 1607 | 8.04 | 26 | 5000 |
| MaxCut600 | 600 | 6000 | 0.097% | 1932 | 8.04 | 26 | 6000 |
| MaxCut800 | 800 | 8000 | 0.072% | 2582 | 8.03 | 26 | 8000 |
| MaxCut1000 | 1000 | 10000 | 0.058% | 3232 | 8.02 | 26 | 10000 |
| MaxCut1200 | 1200 | 12000 | 0.048% | 3882 | 8.02 | 26 | 12000 |
| SpinGlass10 | 10 | 1000 | 0.80% | 155 | 25.69 | 294 | 1000 |
| SpinGlass15 | 15 | 3375 | 0.24% | 191 | 29.97 | 773 | 3375 |
| SpinGlass18 | 18 | 5832 | 0.13% | 1118 | 28.13 | 913 | 5832 |
| SpinGlass20 | 20 | 8000 | 0.10% | 1737 | 25.75 | 1080 | 8000 |
| SpinGlass23 | 23 | 12167 | 0.066% | 2556 | 27.30 | 1488 | 12167 |
| SpinGlass25 | 25 | 15625 | 0.051% | 3173 | 29.50 | 1798 | 15625 |

experiment on SDPARA-C, since the overhead due to the MPI protocol is a severe disadvantage when we run it on a single PC, as shown in Section 3.2.

Table 4 shows the computation time of four solvers using their default parameters. We used 4 threads for SDPA-C 7 and SDPA 7. The symbol '>2days' in the table indicates that we gave up the SeDuMi execution since it required at least two days.

Table 4: The computation time of four solvers for the SDPs in Table 3 (the time unit is second.)

| Name | SDPA-C6 | SDPA-C7 | SDPA 7 | SeDuMi1.3 |
|---|---|---|---|---|
| MaxClique300 | 4792.28 | 889.15 | 11680.12 | 10260.15 |
| MaxClique400 | 12681.16 | 1903.13 | 26159.40 | 24824.05 |
| MaxClique500 | 19973.98 | 3733.41 | 38265.02 | 46168.56 |
| MaxCut400 | 386.80 | 539.22 | 3686.78 | 16779.68 |
| MaxCut500 | 683.27 | 876.81 | 6548.26 | 32557.13 |
| MaxCut600 | 1194.89 | 1295.01 | 11098.35 | 60444.33 |
| MaxCut800 | 2518.80 | 2371.43 | 25377.62 | 146235.81 |
| MaxCut1000 | 4301.11 | 4032.80 | 47270.45 | >2days |
| MaxCut1200 | 7400.28 | 6030.00 | 75888.85 | >2days |
| SpinGlass10 | 50.10 | 20.77 | 11.85 | 228.71 |
| SpinGlass15 | 1306.00 | 560.40 | 336.40 | 13789.58 |
| SpinGlass18 | 6734.40 | 2136.88 | 1522.60 | 68570.89 |
| SpinGlass20 | 15450.13 | 4552.10 | 3726.03 | >2days |
| SpinGlass23 | 55942.57 | 13184.25 | 12598.14 | >2days |
| SpinGlass25 | 107502.48 | 24913.20 | 26023.67 | >2days |

We now examine the details of Table 4. For the max-clique problems, the MC-PDIPM solvers were faster than the standard PDIPM solvers. Since the matrix-completion method derived the nice property of lattice graphs; even SDPA-C 6 was twice faster than SDPA 7. The detail time on

MaxClique400 is displayed in Table 5 (Since SeDuMi does not print out its internal computation time, we do not list its detail time). As shown in the column of SDPA 7, the standard PDIPM consumed large computation time for P-MATRIX (7) and Other(mainly, the computation on the step length by (5)). Though these parts required the $O(n^3)$ computation cost, the MC-PDIPM decomposed the full matrix $\boldsymbol{X}$ into the sub-matrices $\overline{\boldsymbol{X}}_{C_rC_r}$ $(r = 1, \ldots, \ell)$, hence, it was able to save the computation cost of these two parts. Furthermore, SDPA-C 7 resolved the heaviest parts of SDPA-C 6 by the combination of the new factorization $\widehat{\boldsymbol{X}}^{-1} = \widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^T$ and the multithreaded parallel computing. Consequently, SDPA-C 7 was the fastest solver among the four solvers; in particular, SDPA-C 7 was 12.36-times faster than SeDuMi for MaxClique500.

Table 5: The computation time on MaxClique400 (the time unit is second)

| | SDPA-C 6 | SDPA-C 7 | SDPA 7 | SeDuMi 1.3 |
|---|---|---|---|---|
| S-ELEMENTS | 9314.34 | 1431.69 | 262.04 | – |
| S-CHOLESKY | 2601.04 | 248.55 | 403.28 | – |
| P-MATRIX | 734.41 | 175.24 | 13151.10 | – |
| Other | 31.37 | 47.65 | 12342.98 | – |
| Total | 12681.16 | 1903.13 | 26159.40 | 24824.05 |

We move our focus to the max-cut problems. Though the MC-PDIPM was again superior to the standard PDIPM, SDPA-C 6 was more effective for MaxCut500 than SDPA-C 7. We now take a close look at the result of MaxCut500 posted at Table 6. In this SDP, SDPA-C 7 took more computation time on S-ELEMENTS and P-MATRIX, both of which utilized the multithreaded computing. We needed an overhead to generate the threads, and the input matrices of the max-cut problems were too simple to derive the benefit from the multithreaded computing. Indeed, each input matrix $\boldsymbol{A}_i = \boldsymbol{e}_i\boldsymbol{e}_i^T$ has only one nonzero element. This can be verified by the short computation time of S-ELEMENTS in SDPA 7. In the standard PDIPM, the computation of S-ELEMENTS is an inexpensive task of (8) with $\boldsymbol{A}_i = \boldsymbol{e}_i\boldsymbol{e}_i^T$ and $\boldsymbol{A}_j = \boldsymbol{e}_j\boldsymbol{e}_j^T$, since we obtain the fully-dense matrices $\boldsymbol{X}$ and $\boldsymbol{Y}^{-1}$ with the extensive memory space and the heavy computation through P-MATRIX and the inverse of the fully-dense matrix.

For the large max-cut problems, however, SDPA-C 7 solved the SDPs faster than SDPA-C 6 and SDPA 7. As shown in the Max1200 result of Table 6, SDPA-C 7 still incurred the overhead of the multithreading for S-ELEMENTS and P-MATRIX, but the multithreaded BLAS library resolved the principal bottleneck S-CHOLESKY. We can say that SDPA-C 7 matches larger SDPs of this type.

For the spin-glass SDPs with $p = 10$, SDPA 7 was the fastest among the four solvers, since the standard PDIPM was more effective for smaller SDPs where the variable matrices were small and their decomposition was unnecessary. When we solved larger SDPs increasing $p$, however, the difference between SDPA 7 and SDPA-C 7 shrunk, and finally, when $p = 25$, SDPA-C 7 was faster than SDPA 7. The reason why the computation time growth in this type was milder in SDPA-C 7 than in SDPA 7 is that the average size of cliques does not grow along with the increment of $p$ as shown in Table 3. In particular, as seen in the detail computation time in Table 7, this affects P-MATRIX(the computation of $\Delta\boldsymbol{X}$ in (10)). The computation time growth of this part was gradual in the MC-PDIPM than standard PDIPM. SpinGlass25 was the largest among the spin-glass SDPs in our experiments, because the generation of SpinGlass26 required almost 48GB memory space, being close to the capacity of our computing environment. We can expect that SDPA-C 7 would be more effective for larger SDPs of the spin-glass type. Actually, the ratios of SpinGlass25 over SpinGlass23 were $\frac{107502.48}{55942.57} = 1.92$ in SDPA-C 6, $\frac{24913.20}{13184.25} = 1.89$ in SDPA-C 7, and $\frac{26023.67}{12598.14} = 2.07$

Table 6: The computation time on MaxCut500 and MaxCut1200 (the time unit is second)

| MaxCut500 | | | | |
|---|---|---|---|---|
| | SDPA-C 6 | SDPA-C 7 | SDPA 7 | SeDuMi 1.3 |
| S-ELEMENTS | 105.25 | 317.56 | 16.17 | – |
| S-CHOLESKY | 502.43 | 226.06 | 214.43 | – |
| P-MATRIX | 61.21 | 315.73 | 3254.05 | – |
| Other | 14.38 | 17.46 | 3063.60 | – |
| Total | 683.27 | 876.81 | 6548.26 | 32557.13 |
| MaxCut1200 | | | | |
| | SDPA-C 6 | SDPA-C 7 | SDPA 7 | SeDuMi 1.3 |
| S-ELEMENTS | 1265.02 | 1800.27 | 107.17 | – |
| S-CHOLESKY | 5562.48 | 2318.17 | 2674.50 | – |
| P-MATRIX | 532.21 | 1837.53 | 39490.81 | – |
| Other | 40.57 | 74.03 | 33616.37 | – |
| Total | 7400.28 | 6030.00 | 75888.85 | >2days |

in SDPA 7. Meanwhile, the computation cost of the other parts 'Other' occupies considerable amount in the SDPA 7 result. This 'Other' contains the miscellaneous parts and most of them are related to the computation of variable matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ in SDPA 7. Since they are miscellaneous and we can not say which part of them is principal, we do not examine the details, but we should note one point that the fully-dense property of $\boldsymbol{X}$ and $\boldsymbol{Y}$ leads to a lowering in the performance of 'Other' in SDPA 7.

We should emphasize that the MC-PDIPM alone is not the factor of SDPA-C 7, as we can see in Table 3 that SDPA-C 6 was much slower than SDPA 7. The new factorization of $\widehat{\boldsymbol{X}}^{-1} = \widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^T$ and the multithreaded computing are the key to solving the spin-glass SDPs in the shortest time.

Table 7: The computation time on Spinglass18 and Spinglass25 (the time unit is second)

| Spinglass18 | | | | |
|---|---|---|---|---|
| | SDPA-C 6 | SDPA-C 7 | SDPA 7 | SeDuMi 1.3 |
| S-ELEMENTS | 2738.18 | 1012.13 | 22.93 | – |
| S-CHOLESKY | 178.15 | 52.70 | 30.69 | – |
| P-MATRIX | 2516.52 | 984.76 | 620.54 | – |
| Other | 1301.5 | 87.29 | 848.44 | – |
| Total | 6734.40 | 2136.88 | 1522.60 | 68570.89 |
| Spinglass25 | | | | |
| | SDPA-C 6 | SDPA-C 7 | SDPA 7 | SeDuMi 1.3 |
| S-ELEMENTS | 35314.96 | 11829.56 | 220.19 | – |
| S-CHOLESKY | 3681.44 | 945.49 | 520.39 | – |
| P-MATRIX | 45238.37 | 11594.76 | 11846.59 | – |
| Other | 23267.71 | 588.39 | 13436.50 | – |
| Total | 107502.48 | 24913.20 | 26023.67 | >2days |

Finally, Table 8 shows the required memory space to sovle the SDPs in Tables 5,6, and 7. The notation '> 31G' indicates that SeDuMi gave up the problem by the time limit (2 days) and used 31gigabyte memory space during the 2-day execution. The MC-PDIPM saved a lot of memory

space by removing the fully-dense matrices. For example, in MaxClique400, SDPA-C7 consumed only $\frac{1}{6}$ times and $\frac{1}{10}$ times memory space of SDPA7 and SeDuMi, respectively. In addition, the employment of the new factorization reduced the memory space for the largest SDP (Spinglass25) from 8.1 gigabytes in SDPA-C6 to 3.7 gigabytes in SDPA-C7. Since the new factorization can reuse the memory structure of CHOLMOD, it is effective to reduce the required memory space.

Table 8: The required memory space for the SDPs in Tables 5,6, and 7 (M and G indicate megabytes and gigabytes, respectively.)

| Name | SDPA-C6 | SDPA-C7 | SDPA 7 | SeDuMi1.3 |
|---|---|---|---|---|
| MaxClique400 | 548M | 516M | 3.0G | 5.2G |
| MaxCut500 | 249M | 236M | 4.1G | 6.1G |
| MaxCut1200 | 1.2G | 1.2G | 23G | $>$ 31G |
| SpinGlass18 | 2.0G | 707M | 5.6G | 8.3G |
| SpinGlass25 | 8.1G | 3.7G | 40G | $>$ 36G |

# 5 Conclusions and Future Directions

We implemented the new SDPA-C 7 using the more effective factorization of $\widehat{\boldsymbol{X}}^{-1} = \widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^T$ and the multithreaded parallel computing. From the numerical experiments, we verified that these two factors strongly enhanced the performance of the MC-PDIPM, and that SDPA-C 7 successfully reduced the computation time of the max-clique and spin-glass SDPs.

SDPA-C 7 is available at the SDPA web site, http://sdpa.sourceforge.net/. Compared to the existing SDPA-C 6, SDPA-C 7 has a callable library and a Matlab interface. Hence, SDPA-C 7 can be embedded into other C++ software packages, and can be directly called from the inside of Matlab. It can be expected that the callable library and the Matlab interface will expand the SDPA-C usage.

Finally, we discuss some future directions. As shown in the numerical experiments on max-cut problems, if the structure of input SDP is very simple, we should automatically turn off the multithreading. However, this would require a complex task to estimate the computation time accurately over the multiple-threading from the input SDPs. Another point is that SDPA-C 7 has a tendency to be faster for large SDPs. This is an excellent feature of SDPA-C 7, but it cannot attain such performance for smaller SDPs. Though this is mainly because the MC-PDIPM is intended to solve large SDPs with the factorization of the variable matrices based on the structural sparsity, we should combine some methods that effectively compute the forward/backward substitution of sparse matrices of small dimensions.

# Acknowledgments

The authors thank Professor Michael Jünger of Universität zu Köln and Professor Frauke Liers of Friedrich-Alexander Universität Erlangen-Nürnberg for providing us for a general instance generator of the Ising spin glasses computation.

## 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 J. Optim.*, 8(2):746–768, 1998.

[2] P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, 184:501–520, 2000.

[3] 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, Philadelphia, PA, USA, 1999.

[4] P. Biswas and Y. Ye. Semidefinite programming for ad-hoc wireless sensor network localization. In *Proceedings of the third international symposium on information processing in sensor networks*, Berkeley, California, 2004. ACM.

[5] B. Borchers. CSDP, a C library for semidefinite programming. *Optim. Methods Softw.*, 11 & 12(1-4):613–623, 1999.

[6] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD: supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):Article No. 22, 2009.

[7] 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. *Math. Program. Series B*, 109(2-3):553–580, 2007.

[8] M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: general framework. *SIAM J. Optim.*, 11(3):647–674, 2000.

[9] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

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

[11] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. Optim.*, 6(2):342–361, 1996.

[12] D. Johnson and G. Pataki, 2000. http://dimacs.rutgers.edu/Challenges/Seventh/.

[13] M. Jünger, 2011. http://www.informatik.uni-koeln.de/spinglass/.

[14] S. Kim, M. Kojima, H. Waki, and M. Yamashita. Algorithm 920: SFSDP: a Sparse version of Full SemiDefinite Programming relaxation for sensor network localization problems. *ACM Trans. Math. Softw.*, 38(4):Article No. 27, 2012.

[15] M. Kojima, S. Shindoh, and S. Hara. Interior-point methods for the monotone semidefinite linear complementarity problems in symmetric matrices. *SIAM J. Optim.*, 7:86–125, 1997.

[16] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5:308—323, 1979.

[17] L. Lovász. On the shannon capacity of a graph. *IEEE Trans. Inf. Theory*, 25:1–7, 1979.

[18] R. D. C. Monteiro. Primal-dual path-following algorithms for semidefinite programming. *SIAM J. Optim.*, 7(3):663–678, 1997.

[19] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima, and K. Murota. Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical results. *Math. Program., Ser B*, 95:303–327, 2003.

[20] K. Nakata, M. Yamashita, K. Fujisawa, and M. Kojima. A parallel primal-dual interior-point method for semidefinite programs. *Parallel Computing*, 32(1):24–43, 2006.

[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. *J. Chem. Phys.*, 114:8282–8292, 2001.

[22] Y. E. Nesterov and M. J. Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM J. Optim.*, 8(2):324–364, 1998.

[23] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, 11 & 12(1-4):625–653, 1999.

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

[25] K.-C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 – a MATLAB software package for semidefinite programming, version 1.3. *Optim. Methods Softw.*, 11 & 12(1-4):545–581, 1999.

[26] M. Yamashita, K. Fujisawa, M. Fukuda, K. Kobayashi, K. Nakta, and M. Nakata. Latest developments in the SDPA family for solving large-scale SDPs. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications*, chapter 24, pages 687–714. Springer, NY, USA, 2011.

[27] M. Yamashita, K. Fujisawa, M. Fukuda, K. Nakata, and M. Nakata. Algorithm 925: Parallel solver for semidefinite programming problem having sparse Schur complement matrix. *ACM Trans. Math. Softw.*, 39(1), 2012. Article No.6.

[28] M. Yamashita, K. Fujisawa, and M. Kojima. SDPARA: SemiDefinite Programming Algorithm paRAllel version. *Parallel Comput.*, 29(8):1053–1067, 2003.