# Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical results

Kazuhide Nakata[*] Katsuki Fujisawa[†] Mituhiro Fukuda[‡]
Masakazu Kojima[§] Kazuo Murota[¶]

**Abstract**

In Part I of this series of articles, we introduced a general framework of exploiting the aggregate sparsity pattern over all data matrices of large scale and sparse semidefinite programs (SDPs) when solving them by primal-dual interior-point methods. This framework is based on some results about positive semidefinite matrix completion, and it can be embodied in two different ways. One is by a conversion of a given sparse SDP having a large scale positive semidefinite matrix variable into an SDP having multiple but smaller positive semidefinite matrix variables. The other is by incorporating a positive definite matrix completion itself in a primal-dual interior-point method. The current article presents the details of their implementations. We introduce new techniques to deal with the sparsity through a clique tree in the former method and through new computational formulae in the latter one. Numerical results over different classes of SDPs show that these methods can be very efficient for some problems.

*Keywords:* Semidefinite programming; Primal-dual interior-point method; Matrix completion problem; Clique tree; Numerical results.

[*]Department of Applied Physics, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8565 Japan (`nakata@zzz.t.u-tokyo.ac.jp`).

[†]Department of Architecture and Architectural Systems, Kyoto University, Kyoto 606-8501 Japan (`fujisawa@is-mj.archi.kyoto-u.ac.jp`).

[‡]Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan (`mituhiro@is.titech.ac.jp`). The author was supported by The Ministry of Education, Culture, Sports, Science and Technology of Japan.

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

[¶]Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502 Japan (`murota@kurims.kyoto-u.ac.jp`).

## 1 Introduction

In recent years, semidefinite programs (SDPs) have appeared in several fields such as system and control theory, finance, architecture, *etc.* It is also known that SDP relaxations of difficult optimization problems such as combinatorial optimization problems, nonconvex quadratic programs, *etc.*, provide with good bounds for their objective values. In most of the cases, their SDP formulations become large scaled and sparse.

This article is a continuation of Part I [8] in which we presented the basic theory of two new methods to exploit the sparsity structure of large scale SDPs based on some fundamental results about positive semidefinite matrix completion [9, 11]. Here, we focus on the implementation details of the proposed *conversion method* and *completion method* [8], and provide with some numerical results over different classes of SDPs for which they are very effective.

The sections of this article are organized as follows. Section 2 gives the basic definitions and revises some results of Part I [8]. In particular, subsection 2.2 can be viewed as a preprocessing of sparse SDPs for the conversion and completion methods. Section 3 gives a brief glance at the standard primal-dual interior-point method for SDPs. The conversion method is presented in details in section 4, together with a heuristic algorithm to obtain a "good" conversion. The completion method is detailed in section 5 where we utilize new computational formulae to exploit the sparse structure of SDPs. The proposed two methods and the standard primal-dual interior-point method for SDPs are compared in terms of required flops and memory in subsection 5.4. Finally, we devote section 6 to some numerical experiments over five different classes of SDPs.

## 2 Sparse semidefinite programs and investigations into their sparse structure

This section connects the sparse structure of SDPs with the matrix completion theory which in turn is deeply related with chordal graphs. We start subsection 2.1 by defining the standard equality form SDP which is assumed to be sparse, and later we discuss how to obtain a positive (semi)definite matrix completion of one of its variables. Subsection 2.2 presents properties of chordal graphs and clique trees, and subsection 2.3 recalls the sparse clique-factorization formula [8] utilized in the completion method (section 5). In practice, subsection 2.2 can be viewed as a preprocessing of SDPs in order to exploit their sparsity in the conversion method (section 4) and in the completion method (section 5).

### 2.1 Relations among semidefinite programming, matrix completion and chordal graphs

Let $\mathcal{S}^n$ denote the space of $n \times n$ symmetric matrices with the Frobenius inner product $\boldsymbol{X} \bullet \boldsymbol{Y} = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} Y_{ij}$ for $\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{S}^n$. We will use the notation $\boldsymbol{X} \in \mathcal{S}^n_+$ ($\mathcal{S}^n_{++}$) to designate that $\boldsymbol{X} \in \mathcal{S}^n$ is positive semidefinite (definite). Given $\boldsymbol{A}_p \in \mathcal{S}^n$ $(p = 0, 1, \ldots, m)$ and $\boldsymbol{b} \in \mathbb{R}^m$, we define the standard equality form SDP by

$$\left.\begin{array}{ll}\text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{A}_p \bullet \boldsymbol{X} = b_p \ (p = 1, 2, \ldots, m), \ \boldsymbol{X} \in \mathcal{S}^n_+\end{array}\right\}, \tag{1}$$

and its dual by

$$\left.\begin{array}{ll}\text{maximize} & \displaystyle\sum_{p=1}^{m} b_p z_p \\ \text{subject to} & \displaystyle\sum_{p=1}^{m} \boldsymbol{A}_p z_p + \boldsymbol{Y} = \boldsymbol{A}_0, \ \boldsymbol{Y} \in \mathcal{S}^n_+\end{array}\right\}. \tag{2}$$

In this article, we are mostly interested in solving sparse SDPs where the data matrices $\boldsymbol{A}_p$ $(p = 0, 1, \ldots, m)$ are sparse, and the dual matrix variable

$$\boldsymbol{Y} = \boldsymbol{A}_0 - \sum_{p=1}^{m} \boldsymbol{A}_p z_p$$

inherits the sparsity of $\boldsymbol{A}_p$'s.

In order to represent the sparse structure of an SDP, we introduce the *aggregate sparsity pattern* of the data matrices:

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

Here $V$ denotes the set $\{1, 2, \ldots, n\}$ of row/column indices of the data matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$, and $[\boldsymbol{A}_p]_{ij}$ denotes the $(i, j)$th element of $\boldsymbol{A}_p \in \mathcal{S}^n$. It is also convenient to identify the aggregate sparsity pattern $E$ with the *aggregate sparsity pattern matrix* $\boldsymbol{A}$ having unspecified nonzero numerical values in $E$ and zero otherwise.

For every pair of subsets $S$ and $T$ of $V$, we use the notation $\boldsymbol{X}_{ST}$ for the submatrix obtained by deleting all rows $i \notin S$ and all columns $j \notin T$. In accordance with the ideas and definitions presented in [8], consider a collection of nonempty subsets $C_1, C_2, \ldots, C_\ell$ of $V$ satisfying

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

(ii) Any partial symmetric matrix $\bar{\boldsymbol{X}}$ with specified elements $\bar{X}_{ij} \in \mathbb{R}$ $((i, j) \in F)$ has a *positive semidefinite (definite) matrix completion* (*i.e.,* given any $\bar{X}_{ij} \in \mathbb{R}$ $((i, j) \in F)$, there exists a positive semidefinite (definite) $\boldsymbol{X} \in \mathcal{S}^n$ such that $X_{ij} = \bar{X}_{ij} \in \mathbb{R}$ $((i, j) \in F)$) if and only if the submatrices $\bar{\boldsymbol{X}}_{C_r C_r} \in \mathcal{S}^{C_r}_+$ $(\bar{\boldsymbol{X}}_{C_r C_r} \in \mathcal{S}^{C_r}_{++})$ $(r = 1, 2, \ldots, \ell)$.

Here $\mathcal{S}^{C_r}_+$ $(\mathcal{S}^{C_r}_{++})$ denotes the set of $\#C_r \times \#C_r$ positive semidefinite (definite) symmetric matrices with elements specified in $C_r \times C_r$, and $\#C_r$ denotes the number of elements of $C_r$. We can assume without loss of generality $C_1, C_2, \ldots, C_\ell$ to be maximal sets with respect to set inclusion.

Let $G(V, F^\circ)$ be an undirected graph with vertex set $V$ and edge set $F^\circ = F \backslash \{(i, i) : i = 1, 2, \ldots, n\}$ where $F$ is given in (i). Henceforth, we will call the subset $C_r$ of the vertex set $V$ a clique whenever it induces a clique of $G(V, F^\circ)$. Then $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ in (i) will be the family of all maximal cliques of $G(V, F^\circ)$. Under this notation, it is known that

(ii) holds if and only if the graph $G(V, F^{\circ})$ is chordal [9, Theorem 7] (*cf.*, also [8, Theorem 2.3]), where a graph is said to be *chordal* if every cycle of length $\geq 4$ has a chord (an edge connecting two nonconsecutive vertices of the cycle).

Henceforth, we always assume that $G(V, F^{\circ})$ denotes a chordal graph. We call $F$ an *extended sparsity pattern* of $E$, and $G(V, F^{\circ})$ a *chordal extension* or *filled graph* of $G(V, E^{\circ})$, respectively.

## 2.2 Chordal extensions and clique tree properties

One of the key ideas to efficiently solve SDPs via positive (semi)definite matrix completion is to make the best use of the nice property (ii) to diminish the number of unknowns in the primal matrix variable $\boldsymbol{X}$. Therefore, it is extremely important to determine a chordal extension $G(V, F^{\circ})$ of $G(V, E^{\circ})$ which has as small a number of edges as possible since this number directly affects the performance of the conversion method (section 4) and the completion method (section 5).

It is known that a chordal extension can be obtained easily if we perform a symbolic Cholesky factorization to the aggregate sparsity pattern matrix $\boldsymbol{A}$ according to any reordering of the vertex set $V = \{1, 2, \ldots, n\}$. Unfortunately, the problem of finding such an ordering that minimizes the fill-in is $\mathcal{NP}$ complete. Hence, it seems reasonable at least in practice to employ some existing heuristic methods such as the multilevel nested dissection [12] and/or the minimum degree [1] to obtain an ordering which possibly produces lesser fill-in. Once we have an ordering, a simple symbolic Cholesky factorization according to this ordering provides us with a chordal extension $G(V, F^{\circ})$.

Chordal graphs are well-known structures in graph theory. Among their nice properties, it is known that a graph is chordal if and only if we can construct a *clique tree* with vertex set equals to the family of all maximal cliques of the same graph (see [2] and references therein for definitions and results that follow). Although there are several equivalent ways to define clique trees, we employ the following one based on the clique-intersection property (CIP) which will be useful throughout the article.

Let $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ be any family of maximal subsets of $V = \{1, 2, \ldots, n\}$. Let $\mathcal{T}(\mathcal{K}, \mathcal{E})$ be a tree formed by vertices from $\mathcal{K}$ and edges from $\mathcal{E} \subseteq \mathcal{K} \times \mathcal{K}$. $\mathcal{T}(\mathcal{K}, \mathcal{E})$ is called a *clique tree* if it satisfies the *clique-intersection property* (CIP):

(CIP) For each pair of vertices $C_r$ and $C_s \in \mathcal{K}$, the set $C_r \cap C_s$ is contained in every vertex on the (unique) path connecting $C_r$ and $C_s$ in the tree.

In particular, we can construct a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ from the chordal extension $G(V, F^{\circ})$ if we take $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ as the family of all maximal cliques of $G(V, F^{\circ})$, and define appropriately the edge set $\mathcal{E} \subseteq \mathcal{K} \times \mathcal{K}$ for $\mathcal{T}(\mathcal{K}, \mathcal{E})$ to satisfy the CIP.

Clique trees can be computed efficiently from a chordal graph [2, 20] or even faster for a special subclass of them [14]. We observe further that clique trees are not uniquely determined for a given chordal graph.

A *topological ordering* of the vertices in any rooted tree is an ordering of the vertices which numbers each child before its parent. Once we obtained a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ from the chordal extension $G(V, F^{\circ})$, we can choose an arbitrary clique as its root. Then, any topological ordering of its maximal cliques $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ satisfies the *running intersection property* (RIP), *i.e.*, for each $r = 1, 2, \ldots, \ell - 1$ it holds that

(RIP) $$\exists s \geq r+1: \quad C_r \cap (C_{r+1} \cup C_{r+2} \cup \cdots \cup C_\ell) \subsetneq C_s.$$

Now, we present another property which characterizes chordal graphs. An ordering $(1, 2, \ldots, n)$ of the vertices of a graph $G(V, E^\circ)$ is called a *perfect elimination ordering* (PEO) if for each $i = 1, 2, \ldots, n-1$, the subset of vertices $\mathrm{Adj}(i) \cap \{i, i+1, \ldots, n\}$ induces a clique in $G(V, E^\circ)$, where $\mathrm{Adj}(i) = \{j \in V : (i, j) \in F^\circ\}$. It is also known that a graph is chordal if and only if there exists a PEO for it.

Finally, we mention that an ordering $(C_1, C_2, \ldots, C_\ell)$ of the maximal cliques of a chordal graph $G(V, F^\circ)$ which satisfies the RIP induces a PEO of the vertices of the graph as follows. For each $r = 1, 2, \ldots, \ell$, we number the vertices in $S_r = C_r \backslash (C_{r+1} \cup C_{r+2} \cup \cdots \cup C_\ell)$ with $\sum_{s=1}^{r-1} |S_s| + 1, \sum_{s=1}^{r-1} |S_s| + 2, \ldots, \sum_{s=1}^{r-1} |S_s| + |S_r|$. We then obtain a PEO of the vertices, in which the vertices in $S_r$ are given consecutive numbers for each $r$ [8].

There are several reasons why we introduce clique trees. As we have mentioned, the existence of clique trees is inherent in chordal graphs. In the next section, we will see that the CIP allows us to determine the overlapping elements of maximal cliques in an efficient manner. Although clique trees are not uniquely determined, it is known that the multiset of separators $\{C_r \cap C_s : (C_r, C_s) \in \mathcal{E}\}$ is invariant for all clique trees $\mathcal{T}(\mathcal{K}, \mathcal{E})$ of a given chordal graph, a fact suitable for our purpose together with the CIP. Here $\mathcal{K}$ denotes the family of all maximal cliques of the given chordal graph. Finally, any ordering of the maximal cliques $\mathcal{K}$ satisfying the RIP, allows us to make a block decomposition of the maximum-determinant positive definite matrix completion of a partial symmetric matrix as we will see next.

## 2.3 The sparse clique-factorization formula

The following lemma plays the central role in the completion method (section 5).

**Lemma 2.1 ([8])** *Let $G(V, F^\circ)$ be a chordal graph, and $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ its family of all maximal cliques indexed according to the RIP. Consider a partial symmetric matrix $\bar{\boldsymbol{X}}$ with elements specified in $F$ which satisfies the clique-PD condition:*

$$\bar{\boldsymbol{X}}_{C_r C_r} \in \mathcal{S}^{C_r}_{++} \quad (r = 1, 2, \ldots, \ell).$$

*Let $\boldsymbol{P}$ be a permutation matrix representing a PEO of $G(V, F^\circ)$ induced by the RIP in such a way that $(1, 2, \ldots, n)$ is a PEO for $\boldsymbol{P}\bar{\boldsymbol{X}}\boldsymbol{P}^T$. Then the maximum-determinant positive definite matrix completion $\hat{\boldsymbol{X}}$ of $\bar{\boldsymbol{X}}$ can be expressed in terms of the sparse clique-factorization formula*

$$\boldsymbol{P}\hat{\boldsymbol{X}}\boldsymbol{P}^T = \boldsymbol{L}_1^T \boldsymbol{L}_2^T \cdots \boldsymbol{L}_{\ell-1}^T \boldsymbol{D} \boldsymbol{L}_{\ell-1} \cdots \boldsymbol{L}_2 \boldsymbol{L}_1, \qquad (3)$$

*where $\boldsymbol{L}_r$ $(r = 1, 2, \ldots, \ell-1)$ are lower triangular matrices, and $\boldsymbol{D}$ is a positive definite block-diagonal matrix consisting of $\ell$ diagonal blocks. More explicitly, let*

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

*Then*

$$[\boldsymbol{L}_r]_{ij} = \begin{cases} 1 & (i = j) \\ [\bar{\boldsymbol{X}}^{-1}_{U_r U_r} \bar{\boldsymbol{X}}_{U_r S_r}]_{ij} & (i \in U_r, j \in S_r) \\ 0 & (\textit{otherwise}) \end{cases}$$

*for $r = 1, 2, \dots, \ell - 1$, and*

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

*with*

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

**Remark 2.2** *It is not difficult to see that we can choose $C_s$ in (RIP) as being the "parent" of $C_r$ (in the rooted clique tree) for $r = 1, 2, \dots, \ell - 1$. Therefore, $U_r = C_r \cap (C_{r+1} \cup C_{r+2} \cup \cdots \cup C_\ell)$ coincides with $C_r \cap C_s$, and it can be simultaneously obtained during the construction of the clique tree.*

Henceforth, we assume that the maximum-determinant positive definite matrix completion $\hat{\boldsymbol{X}}$ of $\bar{\boldsymbol{X}}$ is already reordered according to a PEO, and therefore we drop the matrices $\boldsymbol{P}$ and $\boldsymbol{P}^T$ from the sparse clique-factorization formula (3). Moreover, due to this formula, the matrix $\hat{\boldsymbol{X}}$ can be expressed in the form

$$\hat{\boldsymbol{X}}^{-1} = \boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^T \tag{4}$$

where $\boldsymbol{W} = \boldsymbol{L}_1^{-1}\boldsymbol{L}_2^{-1}\cdots\boldsymbol{L}_{\ell-1}^{-1}$ is a lower triangular matrix with possible nonzero elements specified in $F$.

## 3 Primal-dual interior-point method

In this section, we describe a generic framework for primal-dual interior-point methods applied to SDPs (1) and (2) [10, 13, 16, 18, 21, 24]. Various search directions have been proposed so far for primal-dual interior-point methods [21]. We restrict ourselves to the HRVW/KSH/M search direction [10, 13, 16] in this article. In addition, we only consider a *simple* primal-dual path-following interior-point method which is not a Mehrotra type since we implement this simple method in the *completion method* described in section 5. On the other hand, we can solve the standard equality form SDP resulting from the *conversion method* described in section 4 by Mehrotra type primal-dual path-following interior-point methods.

The simple primal-dual path-following interior-point method using the HRVW/KSH/M search direction for SDPs is described in Algorithm 3.1.

Notice that if the data matrices $\boldsymbol{A}_p$ $(p = 0, 1, \dots, m)$ have a common block-diagonal matrix structure, Algorithm 3.1 can be carried out more efficiently. This fact is exploited in several software packages [3, 6, 19, 23].

In general, when we apply primal-dual interior-point methods to large scaled and sparse SDPs (especially when $n$ is large), it becomes extremely expensive to treat the large and dense matrices (*e.g.*, $\boldsymbol{X}$, $d\boldsymbol{X}$ and $\widetilde{d\boldsymbol{X}}$ at Steps 2 and 3), since their multiplications require $\mathcal{O}(n^3)$ flops [5]. Utilizing the idea of positive (semi)definite matrix completion (section 2), the conversion method and completion method, which will be presented in the next two sections, solve partially the above drawback of primal-dual interior-point methods.

**Algorithm 3.1: Primal-Dual Path-Following Interior-Point Method**

Step 0: Set a stopping criterion, a parameter $\kappa \in [0,1)$, and choose an initial interior-point $(\boldsymbol{X}^0, \boldsymbol{Y}^0, \boldsymbol{z}^0) \in \mathcal{S}^n_{++} \times \mathcal{S}^n_{++} \times \mathbb{R}^m$. Let $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{z}) = (\boldsymbol{X}^0, \boldsymbol{Y}^0, \boldsymbol{z}^0)$.

Step 1: If the current iterate $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{z})$ satisfies the stopping criterion, stop the iteration. Otherwise, compute $\mu = \boldsymbol{X} \bullet \boldsymbol{Y}/n$.

Step 2: Compute the HRVW/KSH/M search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z}) \in \mathcal{S}^n \times \mathcal{S}^n \times \mathbb{R}^m$ which satisfies the system of linear equations

$$\left.\begin{array}{l} \boldsymbol{A}_p \bullet d\boldsymbol{X} = g_p \ (p = 1, 2, \ldots, m), \quad d\boldsymbol{X} \in \mathcal{S}^n, \\ \displaystyle\sum_{p=1}^{m} \boldsymbol{A}_p dz_p + d\boldsymbol{Y} = \boldsymbol{H}, \quad d\boldsymbol{Y} \in \mathcal{S}^n, \ d\boldsymbol{z} \in \mathbb{R}^m, \\ \widetilde{d\boldsymbol{X}}\boldsymbol{Y} + \boldsymbol{X} d\boldsymbol{Y} = \boldsymbol{K}, \quad \widetilde{d\boldsymbol{X}} \in \mathbb{R}^{n\times n}, \ d\boldsymbol{X} = (\widetilde{d\boldsymbol{X}} + \widetilde{d\boldsymbol{X}}^T)/2 \end{array}\right\}, \tag{5}$$

where $g_p = b_p - \boldsymbol{A}_p \bullet \boldsymbol{X} \in \mathbb{R}$ $(p = 1, 2, \ldots, m)$, $\boldsymbol{H} = \boldsymbol{A}_0 - \sum_{p=1}^{m} \boldsymbol{A}_p z_p - \boldsymbol{Y} \in \mathcal{S}^n$, and $\boldsymbol{K} = \kappa\mu\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y}$.

Step 3: Choose a primal step length $\alpha_p$ and a dual step length $\alpha_d$ such that

$$\boldsymbol{X} + \alpha_p d\boldsymbol{X} \in \mathcal{S}^n_{++} \ \text{ and } \boldsymbol{Y} + \alpha_d d\boldsymbol{Y} \in \mathcal{S}^n_{++},$$

and let

$$\boldsymbol{X} = \boldsymbol{X} + \alpha_p d\boldsymbol{X} \ \text{ and } (\boldsymbol{Y}, \boldsymbol{z}) = (\boldsymbol{Y}, \boldsymbol{z}) + \alpha_d(d\boldsymbol{Y}, d\boldsymbol{z}).$$

Step 4: Go to Step 1.

## 4 Conversion method

This section is devoted to a practical implementation of the conversion method. Here, we assume that a chordal extension $G(V, F^\circ)$ of the aggregate sparsity pattern and a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ of $G(V, F^\circ)$ were already constructed, where $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ is the family of all maximal cliques of $G(V, F^\circ)$. See subsection 2.2.

Subsection 4.1 gives an equivalent formulation, which has multiple and smaller size positive semidefinite matrix variables, to the standard primal SDP (1), and therefore, might be solved faster than (1). Although, the conversion method seems attractive, it has the drawback of adding extra equalities to its equivalent formulation when we convert a given standard primal SDP. This drawback is partially removed in subsection 4.2 where we present a heuristic algorithm to diminish these extra equalities. Both subsections make use of the CIP of clique trees (subsection 2.2).

## 4.1 Conversion to an SDP having multiple but smaller size positive definite semidefinite matrix variables

In [8, section 4], it was shown that the standard primal SDP (1) can be converted to an equivalent SDP having multiple but smaller size positive semidefinite matrix variables. There, special attention was paid to deal with the overlapping variables $X_{ij}$ $((i,j) \in (C_r \cap C_s) \times (C_r \cap C_s),\ C_r \cap C_s \neq \emptyset)$ of two distinct positive semidefinite constraints $\boldsymbol{X}_{C_rC_r} \in \mathcal{S}_+^{C_r}$ and $\boldsymbol{X}_{C_sC_s} \in \mathcal{S}_+^{C_s}$.

In fact, the indices of such overlapping variables can be detected easily from a clique tree. As we will see below, a single visit to each edge $(C_r, C_s) \in \mathcal{E}$ of the clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ in any order, and the determination of $C_r \cap C_s$ provides us with these indices. Then, we can write the equivalent SDP [8, section 4] as follows:

$$\left.\begin{array}{lll} \text{minimize} & \displaystyle\sum_{(i,j)\in F} [\boldsymbol{A}_0]_{ij} X_{ij}^{\hat{r}(i,j)} & \\ \text{subject to} & \displaystyle\sum_{(i,j)\in F} [\boldsymbol{A}_p]_{ij} X_{ij}^{\hat{r}(i,j)} = b_p & (p = 1, 2, \ldots, m), \\ & X_{ij}^r = X_{ij}^s & \left(\begin{array}{ll} (i,j) \in (C_r \cap C_s) \times (C_r \cap C_s), & i \geq j, \\ (C_r, C_s) \in \mathcal{E}, & 1 \leq r < s \leq \ell \end{array}\right), \\ & \boldsymbol{X}^r \in \mathcal{S}_+^{C_r} & (r = 1, 2, \ldots, \ell) \end{array}\right\}, \tag{6}$$

where $\hat{r}(i,j) = \min\{r :\ (i,j) \in C_r \times C_r\}$ is introduced to avoid the addition of repeated terms. If we further introduce a block-diagonal symmetric matrix variable of the form

$$\boldsymbol{X}' = \begin{pmatrix} \boldsymbol{X}^1 & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{X}^2 & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{X}^\ell \end{pmatrix},$$

and appropriately rearrange all data matrices $\boldsymbol{A}_0, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_m$, and the matrices corresponding to the equalities $X_{ij}^r = X_{ij}^s$ to have the same block-diagonal structure as $\boldsymbol{X}'$, we obtain a standard equality form SDP.

Observe that the standard primal SDP (1) has a single matrix variable of size $n \times n$ and $m$ equality constraints. After the conversion, the SDP (6) has

(a) $\ell$ matrices of size $\#C_r \times \#C_r$, $\#C_r \leq n$ $(r = 1, 2, \ldots, \ell)$, and

(b) $m_+ = m + \displaystyle\sum_{(C_r, C_s)\in\mathcal{E}} g(C_r \cap C_s)$ equality constraints where $g(C_r) = \dfrac{\#C_r(\#C_r + 1)}{2}$.

It is not difficult to see that the equations $X_{ij}^r = X_{ij}^s$ in (6) provide with the association we sought. Suppose for instance that $\boldsymbol{X}_{C_rC_r}$ and $\boldsymbol{X}_{C_sC_s}$ share the same variable at the position $(i,j)$, and therefore, we need to associate $[\boldsymbol{X}_{C_rC_r}]_{ij}$ with $[\boldsymbol{X}_{C_sC_s}]_{ij}$. We have $i, j \in C_r \cap C_s$, and since $C_r$ and $C_s$ are vertices in the clique tree, there is a unique path of maximal cliques $[C_r = C^0, C^1, \ldots, C^f = C_s]$ $(f \geq 1)$ which connects them. Using the CIP, we have $i, j \in C^t$ $(t = 0, 1, \ldots, f)$, and the equations in (6) give us

$$[\boldsymbol{X}_{C^0C^0}]_{ij} = [\boldsymbol{X}_{C^1C^1}]_{ij} = \cdots = [\boldsymbol{X}_{C^fC^f}]_{ij}$$

as we wanted. Observe further that we do not have redundant equalities in (6) otherwise we would have a cycle in the clique tree. Finally, due to the invariance of the multiset of separators $\{C_r \cap C_s : (C_r, C_s) \in \mathcal{E}\}$, the equalities in (6) are independent of the choice of the clique tree we make.

**Remark 4.1** *We employed a different notation from the one used in the previous article [8] to describe the converted SDP. In [8, (4.1)], the indices of the overlapping variables were represented by $E_r = \{(i,j) \in C_r \times C_r : (i,j) \in C_s \times C_s$ for some $s < r\}$, and each auxiliary variable $U^r_{ij}$ $((i,j) \in E_r,\ i \geq j,\ r = 2,3,\ldots,\ell)$ was associated with the same variable $X_{ij}$ $((i,j) \in E_r,\ i \geq j,\ r = 2,3,\ldots,\ell)$. Making use of the CIP, the reader can verify that [8, (4.1)] and (6) are the same SDP in practice, although the equalities are rearranged in a different manner. We preferred though to utilize the notation in (6) which is closer to our implementation.*

## 4.2 Obtaining a "good" clique tree for the conversion method

As pointed out in [8, sections 4 and 7], we need to balance the factors (a) and (b) stated above in the conversion method to obtain an equivalent SDP (6) which we expect to be solved in a shorter time than the standard primal SDP (1). Observe that the factors (a) and (b) predict the flops at Steps 3 and 2 in Algorithm 3.1, respectively. Although we know a rough estimate of the flops for typical primal-dual interior-point methods applied to (6) (see subsection 5.4), it seems a hard task to determine a suitable chordal extension, and consequently a clique tree which also takes into account the sparsity and the sparse structure of (6) to diminish the flops.

We propose, therefore, the following simple strategy to obtain a "good" clique tree (chordal extension) for the conversion method. We first construct a clique tree from a chordal graph which has as little fill-in as possible through the method described in subsection 2.2. We can use for instance the best ordering between the multilevel nested dissection [12] and the minimum degree [1] for that. This clique tree has possibly several maximal cliques of small sizes. Then, the lemmas below guarantee that we can successively diminish the number of vertices in the clique tree maintaining its structure. Consequently, we can diminish the quantity $m_+$ in (b) and the number $\ell$ of matrices in (a), at the same time we increase the sizes of these matrices.

**Lemma 4.2** *Let $\mathcal{T}(\mathcal{K},\mathcal{E})$ be a clique tree of $G(V,F^\circ)$, and suppose that $(C_r, C_s) \in \mathcal{E}$. We construct a new tree $\mathcal{T}'(\mathcal{K}',\mathcal{E}')$ merging $C_r$ and $C_s$, i.e., replacing $C_r, C_s \in \mathcal{K}$ by $C_r \cup C_s \in \mathcal{K}'$, deleting $(C_r, C_s) \in \mathcal{E}$, and replacing $(C_r, C), (C_s, C) \in \mathcal{E}$ by $(C_r \cup C_s, C) \in \mathcal{E}'$. Then $\mathcal{T}'(\mathcal{K}',\mathcal{E}')$ is a clique tree of $G(V, F'^\circ)$, where $F' = \{(i,j) \in C'_r \times C'_r : r = 1,2,\ldots,\ell'\}$ for $\mathcal{K}' = \{C'_1, C'_2, \ldots, C'_{\ell'}\}$, $\ell' = \ell - 1$. Moreover, let $m_+$ be defined as in (b), and $m'_+$ be the corresponding one for $\mathcal{T}'(\mathcal{K}',\mathcal{E}')$. Then $m'_+ = m_+ - g(C_r \cap C_s)$.*

*Proof:* We first verify the CIP for $\mathcal{T}'(\mathcal{K}',\mathcal{E}')$. Let $C, C' \in \mathcal{K}'$, and $[C = C^1, C^2, \ldots, C^f = C']$ $(f > 2)$ be the unique path in $\mathcal{T}'(\mathcal{K}',\mathcal{E}')$ which connects $C$ and $C'$. We consider three cases for the proof. Case a: $C_r \cup C_s = C^t$ for some $t \in \{2,3,\ldots,f-1\}$; case b: $C_r \cup C_s = C^t$ for $t = 1$ or $t = f$; case c: otherwise. Case a: since the cliques in $\mathcal{T}(\mathcal{K},\mathcal{E})$ satisfy the CIP, $C \cap C' \subseteq C^1, C^2, \ldots, C^{t-1}, C_r, C^{t+1}, \ldots, C^f$ and/or

$C \cap C' \subseteq C^1, C^2, \ldots, C^{t-1}, C_s, C^{t+1}, \ldots, C^f$. Therefore, $C \cap C' \subseteq C^1, C^2, \ldots, C^{t-1}, C_r \cup C_s, C^{t+1}, \ldots, C^f$, and the CIP is valid. Case b: we assume without loss of generality that $C_r \cup C_s = C^1$ (which includes the case $C_r \cup C_s = C^f$). Since $\mathcal{T}(\mathcal{K}, \mathcal{E})$ satisfies the CIP, $C_r \cap C^f, C_s \cap C^f \subseteq C^2, C^3, \ldots, C^{f-1}$, and therefore $C \cap C' = (C_r \cup C_s) \cap C^f = (C_r \cap C^f) \cup (C_s \cap C^f) \subseteq C^2, C^3, \ldots, C^{f-1}$ as we wished. Case c: the remaining case ($(C_r \cup C_s) \neq C^t$ for $t = 1, 2, \cdots, f$) is obvious. Next, we show the maximality of the elements in $\mathcal{K}'$. Since it is obvious that $C_r \cup C_s \not\subseteq C$, $\forall C \in \mathcal{K}'$, $C \neq C_r \cup C_s$, we just need to show that $C_r \cup C_s \not\supseteq C$, $\forall C \in \mathcal{K}'$, $C \neq C_r \cup C_s$. Suppose that there is a $C \in \mathcal{K}'$ such that $C_r \cup C_s \supsetneq C$. We can assume without loss of generality that $C_r$ is on the path connecting $C$ to $C_s$ in $\mathcal{T}(\mathcal{K}, \mathcal{E})$. Then, since the CIP is valid in $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$, $C \cap C_s \subseteq C_r$, and therefore $C = (C \backslash C_s) \cup (C \cap C_s) \subseteq C_r \cup C_r = C_r$, which contradicts the maximality of $C$ in $\mathcal{K}$. Consequently, $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ is a clique tree. If we construct a new graph $G(V, F'^{\circ})$ from the family of maximal subsets $\mathcal{K}' = \{C'_1, C'_2, \ldots, C'_{\ell'}\}$ and the edge set $\mathcal{E}'$, we know from the discussion in subsection 2.2 that it is also a chordal graph. $m'_+ = m_+ - g(C_r \cap C_s)$ follows by simple inspection and the CIP. ∎

**Lemma 4.3** *Let $\mathcal{T}(\mathcal{K}, \mathcal{E})$ be a clique tree of $G(V, F^{\circ})$, and suppose that $(C_r, C_q), (C_s, C_q) \in \mathcal{E}$. We construct a new tree $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ in the following way:*

1. *If $C_r \cup C_s \not\supseteq C_q$, merge $C_r$ and $C_s$, i.e., replace $C_r, C_s \in \mathcal{K}$ by $C_r \cup C_s \in \mathcal{K}'$ and replace $(C_r, C), (C_s, C) \in \mathcal{E}$ by $(C_r \cup C_s, C) \in \mathcal{E}'$;*

2. *Otherwise, merge $C_r$, $C_s$ and $C_q$, i.e., replace $C_r, C_s, C_q \in \mathcal{K}$ by $C_r \cup C_s \cup C_q \in \mathcal{K}'$, delete $(C_r, C_q), (C_s, C_q) \in \mathcal{E}$ and replace $(C_r, C), (C_s, C), (C_q, C) \in \mathcal{E}$ by $(C_r \cup C_s \cup C_q, C) \in \mathcal{E}'$.*

*Then $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ is a clique tree of $G(V, F'^{\circ})$, where $F' = \{(i,j) \in C'_r \times C'_r : r = 1, 2, \ldots, \ell'\}$ for $\mathcal{K}' = \{C'_1, C'_2, \ldots, C'_{\ell'}\}$, $\ell' = \ell - 1$ (case 1) and $\ell' = \ell - 2$ (case 2). Moreover, let $m_+$ be defined as in (b), and $m'_+$ be the corresponding one for $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$. Then, we have respectively*

1. *$m'_+ = m_+ - g(C_r \cap C_q) - g(C_s \cap C_q) + g((C_r \cup C_s) \cap C_q)$ and;*

2. *$m'_+ = m_+ - g(C_r \cap C_q) - g(C_s \cap C_q)$.*

*Proof:* The same arguments of Lemma 4.2 guarantee that $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ satisfies the CIP. We now show the maximality of the elements in $\mathcal{K}'$ for the case 1 ($C_r \cup C_s \not\supseteq C_q$). Once more, we just need to show that $C_r \cup C_s \not\supseteq C$, $\forall C \in \mathcal{K}'$, $C \neq C_r \cup C_s$. Suppose that there is some $C \in \mathcal{K}'$ such that $C_r \cup C_s \supsetneq C$. If $C$ is such that $C_q$ does not lie on the path which connects $C$, and $C_r$ or $C_s$ in $\mathcal{T}(\mathcal{K}, \mathcal{E})$, the arguments of Lemma 4.2 apply. Otherwise, suppose that $C$ is such that $C_q$ always lies on the path which connects $C$ and both $C_r$ and $C_s$ in $\mathcal{T}(\mathcal{K}, \mathcal{E})$. We have $C \cap C_r, C \cap C_s \subseteq C_q$ from the CIP. Since $C_r \cup C_s \supsetneq C$, $C = C \cap (C_r \cup C_s) = (C \cap C_r) \cup (C \cap C_s) \subseteq C_q$ which contradicts the maximality of $C$ in $\mathcal{K}$. We know now that the unique case for which $C_r \cup C_s \not\supseteq C$, $C \in \mathcal{K}'$, $C \neq C_r \cup C_s$ might not hold is when $C = C_q$. We are in case 2, and merging $C_q$ with $C_r \cup C_s$, the maximality of the cliques in $\mathcal{K}'$ follows. Using the same arguments of the proof of Lemma 4.2, the resulting $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ is a clique tree of the chordal graph $G(V, F'^{\circ})$. The second part of the lemma follows by simple inspection and the CIP. ∎

Observe that since the tree $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$ resulting from Lemmas 4.2 and 4.3 is a clique tree of $G(V, F'^{\circ})$, it satisfies the CIP. Also $G(V, F'^{\circ})$ is a chordal graph. It is important to notice here that the merging operations of Lemmas 4.2 and 4.3 are equivalent to adding extra edges to the chordal extension $G(V, F^{\circ})$ which results in $G(V, F'^{\circ})$.

As we have mentioned, it seems difficult to find an exact criterion which decides when two maximal cliques $C_r$ and $C_s$ satisfying the hypothesis of Lemmas 4.2 or 4.3 should be merged or not to balance the factors (a) and (b) in terms of the number of flops. Therefore, we adopted the following simple criterion which seems reasonable for our purpose.

Let $\sigma \in (0, 1)$. We merge the cliques $C_r$ and $C_s$ if

$$\min\left\{\frac{\#(C_r \cap C_s)}{\#C_r}, \frac{\#(C_r \cap C_s)}{\#C_s}\right\} \geq \sigma. \tag{7}$$

Although the criterion (7) is not complete, it takes into account the sizes of the involved cliques $C_r$ and $C_s$, and compares them with the size of common indices $\#(C_r \cap C_s)$. Also, the minimization among the two quantities avoids the merging of large and small cliques which share a reasonable number of indices if compared with the smaller one.

We summarize in Algorithm 4.4 the heuristic algorithm we implemented to obtain a "good" clique tree for the conversion method.

**Algorithm 4.4: Diminishing the number of maximal cliques in the clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$**

```
Choose a maximal clique in $\mathcal{K}$ to be the root for $\mathcal{T}(\mathcal{K}, \mathcal{E})$, and let $\sigma \in (0,1)$
    for each maximal clique $C$ which was visited for the last time in $\mathcal{T}(\mathcal{K}, \mathcal{E})$ in a
    depth-first search
        Set $C_q = C$
        for each pair of descendents $C_r$ and $C_s$ of $C_q$ in $\mathcal{T}(\mathcal{K}, \mathcal{E})$
            if criterion (7) is satisfied, and $m'_+ < m_+$ in Lemma 4.3
                then merge $C_r$ and $C_s$ (or $C_q$, $C_r$ and $C_s$), and let $\mathcal{T}(\mathcal{K}, \mathcal{E}) =$
                $\mathcal{T}'(\mathcal{K}', \mathcal{E}')$
        end(for)
        Set $C_r = C$
        for each descendent $C_s$ of $C_r$ in $\mathcal{T}(\mathcal{K}, \mathcal{E})$
            if criterion (7) is satisfied
                then merge $C_r$ and $C_s$ and let $\mathcal{T}(\mathcal{K}, \mathcal{E}) = \mathcal{T}'(\mathcal{K}', \mathcal{E}')$
        end(for)
    end(for)
```

After we have obtained a clique tree $\mathcal{T}(\mathcal{K}, \mathcal{E})$ from Algorithm 4.4, it is sufficient to visit each edge of the resulting clique tree to obtain the overlapping variables of (6) as discussed in subsection 4.1. The unique parameter $\sigma$ in (7) involved in Algorithm 4.4 is unfortunately SDP dependent. We can, however, obtain a "good" conversion even if we fix it to a specific value as we can observe from the numerical experiments (section 6) over our set of SDPs.

# 5 Completion method

One disadvantage of the conversion method is an increase in the number of equality constraints. In this section, we present a primal-dual interior-point method based on positive definite matrix completion which we can apply directly to the SDPs (1) and (2) without adding any equality constraint. In this *completion method* [8], we perform all matrix operations using only sparse matrices, and neither introduce nor store any dense matrix.

This section is organized as follows. In subsection 5.1, we compute the duality gap (Step 1 of Algorithm 3.1) by effectively utilizing the sparsity of the matrix variables. In subsection 5.2, we compute the search direction (Step 2 of Algorithm 3.1) by utilizing a sparse factorization of the primal matrix variable. In subsection 5.3, we compute the primal and dual step lengths (Step 3 of Algorithm 3.1) by utilizing a positive definite matrix completion of the primal matrix variable. Finally, in subsection 5.4, we make a rough comparison among the standard primal-dual interior-point method, the conversion method and the completion method in terms of the number of flops and the memory amount.

We assume that an extended sparsity pattern $F$ of the aggregate sparsity pattern $E$ is already determined, and the maximal cliques $\mathcal{K} = \{C_1, C_2, \ldots, C_\ell\}$ of the chordal extension $G(V, F^\circ)$ are already indexed according to the RIP (subsection 2.2).

Throughout this section, we use the following notation:

- $\mathcal{S}^n(F, ?)$: the set of $n \times n$ partial symmetric matrices with elements specified in $F$;
- $\mathcal{S}^n_{++}(F, ?)$: the set of $n \times n$ partial symmetric matrices with elements specified in $F$ which can be completed to positive definite matrices, *i.e.*, $\mathcal{S}^n_{++}(F,?) = \{\bar{\boldsymbol{X}} \in \mathcal{S}^n(F,?) : \exists \boldsymbol{X} \in \mathcal{S}^n_{++},\ \bar{X}_{ij} = X_{ij} \text{ for } (i,j) \in F\}$;
- $\mathcal{S}^n(F, 0)$: the set of $n \times n$ symmetric matrices with vanishing elements outside $F$, *i.e.*, $\mathcal{S}^n(F,0) = \{\boldsymbol{X} \in \mathcal{S}^n : X_{ij} = 0, \text{ if } (i,j) \notin F\}$;
- $\mathcal{S}^n_{++}(F, 0)$: the set of $n \times n$ positive definite symmetric matrices with vanishing elements outside $F$, *i.e.*, $\mathcal{S}^n_{++}(F,0) = \mathcal{S}^n_{++} \cap \mathcal{S}^n(F,0) = \{\boldsymbol{X} \in \mathcal{S}^n_{++} : X_{ij} = 0 \text{ if } (i,j) \notin F\}$.

Let $(\bar{\boldsymbol{X}}, \boldsymbol{Y}, \boldsymbol{z}) \in \mathcal{S}^n_{++}(F,?) \times \mathcal{S}^n_{++}(E,0) \times \mathbb{R}^m$ be a point obtained at an iteration of Algorithm 3.1 or given initially. Here the feasibility of the point $(\bar{\boldsymbol{X}}, \boldsymbol{Y}, \boldsymbol{z})$ is not assumed; $\bar{\boldsymbol{X}}$ and $(\boldsymbol{Y}, \boldsymbol{z})$ need not satisfy the equality constraints of the SDPs (1) and (2), respectively.

## 5.1 Duality gap

Since $\boldsymbol{Y} \in \mathcal{S}^n_{++}(E,0)$, we use the partial symmetric matrix $\bar{\boldsymbol{X}} \in \mathcal{S}^n_{++}(F,?)$ to compute the duality gap

$$\mu = \frac{1}{n} \sum_{(i,j) \in E} \bar{X}_{ij} Y_{ij}$$

at Step 1 of Algorithm 3.1. The number of required flops amounts to the number of elements in the set $E$.

## 5.2 Search direction

In order to compute the HRVW/KSH/M search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$ at Step 2 of Algorithm 3.1, we reduce the system of linear equations (5) to

$$\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{s}, \tag{8}$$

$$d\boldsymbol{Y} = \boldsymbol{H} - \sum_{p=1}^{m} \boldsymbol{A}_p dz_p, \tag{9}$$

$$\widetilde{d\boldsymbol{X}} = (\kappa\mu\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y} - \boldsymbol{X}d\boldsymbol{Y})\boldsymbol{Y}^{-1}, \tag{10}$$

$$d\boldsymbol{X} = (\widetilde{d\boldsymbol{X}} + \widetilde{d\boldsymbol{X}}^T)/2, \tag{11}$$

where

$$\left.\begin{array}{ll} B_{pq} = \text{Trace } \boldsymbol{A}_p\boldsymbol{X}\boldsymbol{A}_q\boldsymbol{Y}^{-1} & (p,q = 1,2,\ldots,m), \\ s_p = g_p - \text{Trace } \boldsymbol{A}_p(\kappa\mu\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{H})\boldsymbol{Y}^{-1} & (p = 1,2,\ldots,m) \end{array}\right\}.$$

The standard primal-dual interior-point method needs to store all elements of the matrices $\boldsymbol{X}$ and $\boldsymbol{Y}^{-1}$, and employs efficient methods to compute the coefficient matrix $\boldsymbol{B}$ of (8) by exploiting the sparsity of the data matrices $\boldsymbol{A}_p$ $(p = 1,2,\ldots,m)$ (see for instance [7]). The completion method also relies on (8) to compute a search direction $(d\boldsymbol{X}, d\boldsymbol{Y}, d\boldsymbol{z})$, but avoids using the matrices $\boldsymbol{X}$ and $\boldsymbol{Y}^{-1}$ explicitly since they become dense in general. Therefore, we propose a new formula for computing the coefficient matrix $\boldsymbol{B}$.

Let $\hat{\boldsymbol{X}}$ be the maximum-determinant positive definite matrix completion of $\bar{\boldsymbol{X}} \in \mathcal{S}^n_{++}(F, ?)$ obtained by Lemma 2.1, which we will use as the primal matrix variable $\boldsymbol{X}$ in (8). Recall that the inverse $\hat{\boldsymbol{X}}^{-1}$ of $\hat{\boldsymbol{X}}$ is expressed in terms of a sparse factorization (4); hence $\hat{\boldsymbol{X}} = \boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}$, where $\boldsymbol{W}$ is a lower triangular matrix and $\boldsymbol{D}$ is a block-diagonal matrix. Also $\boldsymbol{Y} \in \mathcal{S}^n_{++}(E, 0)$ has a Cholesky factorization $\boldsymbol{Y} = \boldsymbol{N}\boldsymbol{N}^T$ without any fill-in except for elements specified in $F \backslash E$. It should be noted that all the matrices $\boldsymbol{W}$, $\boldsymbol{D}$ and $\boldsymbol{N}$ have possible nonzero elements in the extended sparsity pattern $F$. In addition, the sets of off-diagonal nonzero elements of $\boldsymbol{W}$, $\boldsymbol{D}$ and $\boldsymbol{W}^T$ do not intersect with each other.

Based on matrix-vector multiplications, we now compute the coefficient matrix $\boldsymbol{B}$ and the right hand side vector $\boldsymbol{s}$ of (8) simultaneously.

$$B_{pq} = \sum_{k=1}^{n} (\boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{e}_k)^T\boldsymbol{A}_q(\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_p]_{*k}) \qquad (p,q = 1,2,\ldots,m),$$

$$s_p = b_p + \sum_{k=1}^{n} \left[(\boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{e}_k)^T\boldsymbol{H}(\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_p]_{*k}) - \kappa\mu\boldsymbol{e}_k^T\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_p]_{*k}\right] \qquad (p = 1,2,\ldots,m).$$

Here $[\boldsymbol{A}_p]_{*k}$ denotes the $k$th column of the data matrices $\boldsymbol{A}_p \in \mathcal{S}^n$ $(p = 1,2,\ldots,m)$, and $\boldsymbol{e}_k \in \mathbb{R}^n$ the vector with the $k$th element 1 and others 0. More precisely, we compute $\boldsymbol{B}$ and $\boldsymbol{s}$ according to Algorithm 5.1, which exploits the sparsity of the matrices involved therein as follows:

- The second loop over the index $k$ is not executed when $[\boldsymbol{A}_p]_{*k}$ is equal to the zero vector.

- When we perform a matrix-vector multiplication with the inverse of either of the matrices $\boldsymbol{W}$, $\boldsymbol{W}^T$, $\boldsymbol{N}$ or $\boldsymbol{N}^T$, *e.g.*, $\boldsymbol{v} = \boldsymbol{W}^{-1}\boldsymbol{e}_k$, we solve the system of linear equations $\boldsymbol{W}\boldsymbol{v} = \boldsymbol{e}_k$ instead. Since these matrices are sparse and triangular, we can solve the corresponding system of linear equations efficiently.
- The matrices $\boldsymbol{A}_p$ $(p = 1, 2, \ldots, m)$, $\boldsymbol{H}$ and $\boldsymbol{D}$ are also sparse matrices with possible nonzero elements in $F$, and then, each matrix-vector multiplication involving these matrices is also carried out efficiently.

**Algorithm 5.1: Computation of $\boldsymbol{B}$ and $\boldsymbol{s}$**

```
Set $\boldsymbol{B} = \boldsymbol{O}$ and $\boldsymbol{s} = \boldsymbol{b}$
for $p = 1, 2, \ldots, m$
    for $k = 1, 2, \ldots, n$ with $[\boldsymbol{A}_p]_{*k} \neq \boldsymbol{0}$
        Compute $\boldsymbol{v}_1 = \boldsymbol{N}^{-T}\boldsymbol{N}^{-1}[\boldsymbol{A}_p]_{*k}$ and $\boldsymbol{v}_2 = \boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{e}_k$
        for $q = 1, 2, \ldots, m$
            Compute $\boldsymbol{v}_2^T\boldsymbol{A}_q\boldsymbol{v}_1$ and add to $B_{pq}$
        end(for)
        Compute $\boldsymbol{v}_2^T\boldsymbol{H}\boldsymbol{v}_1$ and add to $s_p$
        Compute $\kappa\mu\boldsymbol{e}_k^T\boldsymbol{v}_1$ and subtract from $s_p$
    end(for)
end(for)
```

As in the standard primal-dual interior-point method, we then apply the Cholesky factorization to the matrix $\boldsymbol{B}$, which is fully dense in general, to obtain a solution $d\boldsymbol{z}$ of the system of linear equations (8). Next, we compute $d\boldsymbol{Y} \in \mathcal{S}^n(E, 0)$ by (9). Since $\boldsymbol{H}, \boldsymbol{A}_p \in \mathcal{S}^n(E, 0)$ $(p = 1, 2, \ldots, m)$, the computation of $d\boldsymbol{Y}$ is performed only for elements specified in $E$.

Now we compute $\widetilde{d\boldsymbol{X}}$ by (10). Each column of $\widetilde{d\boldsymbol{X}}$ is computed by

$$[\widetilde{d\boldsymbol{X}}]_{*k} = \kappa\mu\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k - [\boldsymbol{X}]_{*k} - \boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}d\boldsymbol{Y}\boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k \quad (k = 1, 2, \ldots, n). \tag{12}$$

However, we do not need to compute the elements $[\widetilde{d\boldsymbol{X}}]_{ij}$ with indices $(i, j) \notin F$, since they do not contribute in the search direction [8, section 5]. Each matrix-vector multiplication in (12) is done in the same way as above.

Finally, let $d\boldsymbol{X} = (\widetilde{d\boldsymbol{X}} + \widetilde{d\boldsymbol{X}}^T)/2$; again only elements with indices $(i, j) \in F$ need to be calculated to generate the partial symmetric matrix $d\bar{\boldsymbol{X}} \in \mathcal{S}^n(F, ?)$.

## 5.3 Step length

In [8, section 5], we described a method to avoid dense matrix computation for the step lengths. To determine the primal and dual step lengths at Step 3 of Algorithm 3.1, we compute the minimum eigenvalues of the matrices

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

where $\bar{\boldsymbol{X}}_{C_rC_r} = \bar{\boldsymbol{M}}_r\bar{\boldsymbol{M}}_r^T$ is a Cholesky factorization of $\bar{\boldsymbol{X}}_{C_rC_r} \in \mathcal{S}^{\#C_r}_{++}$. Since the first $\ell$ matrices are smaller than the primal matrix variable $\boldsymbol{X}$, their computation is much faster.

In addition, the minimum eigenvalue of $\boldsymbol{N}^{-1}d\boldsymbol{Y}\boldsymbol{N}^{-T}$ can be computed easily by the Lanczos method, because $\boldsymbol{N}$ and $d\boldsymbol{Y}$ are sparse matrices with possible nonzero elements specified in $F$. In both of the cases, we exploit the sparsity of the matrices $\bar{\boldsymbol{X}} \in \mathcal{S}^n_{++}(F,?)$ and $\boldsymbol{Y} \in \mathcal{S}^n_{++}(E,0)$.

## 5.4 Comparison

Each iteration of the completion method updates and stores the partial symmetric matrices $\bar{\boldsymbol{X}}$, $d\bar{\boldsymbol{X}} \in \mathcal{S}^n(F,?)$, the sparse matrices $\boldsymbol{Y}$, $\boldsymbol{D}$, $\boldsymbol{W}$, $\boldsymbol{N}$, $d\boldsymbol{Y}$ with possible nonzero elements specified in $F$, and the smaller size matrices $d\bar{\boldsymbol{X}}_{C_rC_r} \in \mathcal{S}^{\#C_r}$, $\bar{\boldsymbol{M}}_r \in \mathbb{R}^{\#C_r \times \#C_r}$ $(r = 1, 2, \ldots, \ell)$. The efficiency of the completion method and also of the conversion method depends not only on the number of elements of the extended sparsity pattern $F$, but also on its structure, so that their exact evaluation is quite difficult. Therefore, we only make a rough comparison among the flops and the memory required by each iteration of the standard primal-dual interior-point method, the conversion method and the completion method as shown in Table 1. Four factors are considered here: the size $n$ of the primal matrix variable $\boldsymbol{X}$ and the dual matrix variable $\boldsymbol{Y}$, the number $m$ of linear equality constraints in the standard primal SDP (1), the density $\alpha \in [0,1]$ of the extended sparsity pattern = "(the number of elements of $F$) / $n^2$," and the number $m_+$ of the linear equality constraints in the standard primal SDP (6) for the case of the conversion method. We assume that each data matrix $\boldsymbol{A}_p$ has only $O(1)$ nonzero elements $(p = 0, 1, \ldots, m)$. In Table 1, "other parts" includes the computations of $d\boldsymbol{Y} \in \mathcal{S}^n(E,0)$, $d\bar{\boldsymbol{X}} \in \mathcal{S}^n(F,?)$, the primal and dual step lengths, *etc.*

Table 1: Rough comparison among the standard primal-dual interior-point method, conversion method and completion method.

| | | standard method | conversion method | completion method |
|---|---|---|---|---|
| flops | elements of $\boldsymbol{B}$ | $\mathcal{O}(m^2)$ | $\mathcal{O}(m_+^2)$ | $\mathcal{O}(m^2 + mn^2\alpha)$ |
| | solution of $\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{s}$ | $\mathcal{O}(m^3)$ | $\mathcal{O}(m_+^3)$ | $\mathcal{O}(m^3)$ |
| | other parts | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^3\alpha^{1.5})$ | $\mathcal{O}(n^3\alpha)$ |
| memory | solution of $\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{s}$ | $\mathcal{O}(m^2)$ | $\mathcal{O}(m_+^2)$ | $\mathcal{O}(m^2)$ |
| | other parts | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^2\alpha)$ | $\mathcal{O}(n^2\alpha)$ |

For computing the elements of the coefficient matrix $\boldsymbol{B}$, the conversion method and the completion method need more flops than the standard primal-dual interior-point method. For solving the system of linear equations $\boldsymbol{B}d\boldsymbol{z} = \boldsymbol{s}$, the conversion method needs more flops than the other methods since $m_+$ is generally larger than $m$. Finally, for the other parts, both of the conversion and completion methods are more efficient than the standard primal-dual interior-point method.

Considering now the used memory, the conversion method requires to store the $m_+ \times m_+$ coefficient matrix $\boldsymbol{B}$ which is larger than the coefficient matrix of the standard primal-dual interior-point method and the completion method. For the other parts, the standard primal-dual interior-point method requires the largest amount of memory. Considering all, the completion method requires the least memory storage among the three methods.

## 6 Numerical experiments

In this section, we present some numerical experiments to evaluate the conversion method (section 4) and the completion method (section 5) proposed in [8] and detailed in this article. We compare these two methods with the software SDPA 5.0 [6] which is an implementation of the standard primal-dual path-following interior-point method for SDPs. The comparisons are made over four classes of pseudo-randomly generated 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, and two problems from the 7th DIMACS implementation challenge library (semidefinite and related optimization problems). These problems fulfill the characteristics we have required for our methods, *i.e.*, they are large scale, sparse, and both the aggregate sparsity pattern and the extended sparsity pattern are sparse. The only exceptions are the two problems from the DIMACS implementation challenge library which do not have sparse extended sparsity patterns.

In Tables 3 to 9 of the following subsections, the entries "standard" mean that we solved the original SDP by the SDPA 5.0. The entries "conversion" means that we first applied the conversion method to convert the original SDP (according to the algorithms in subsections 2.2 and 4.2), and then that we solved the resulting SDP by the SDPA 5.0. Finally, the entries corresponding to the "completion" were obtained by the new code SDPA-C which incorporates the sparse clique-factorization formula (3), and all features detailed in subsections 5.1, 5.2 and 5.3. All the tables give the computational time and used memory for "standard", "conversion" and "completion". The numbers between parenthesis in the "conversion" columns are the computational time for the conversion itself with $\sigma$ fixed to 0.06 in (7). The last two columns give the sparsity of the aggregate sparsity patterns and of the extended sparsity patterns. The $n$ and $m$ below the tables correspond to the sizes of the matrices and the number of equality constraints for the original SDP in the standard equality form (1) and (2), respectively. The $n$'s and $m_+$ for the converted SDP are omitted here.

All numerical experiments were conducted on a DEC Alpha Station (CPU Alpha 21164-600MHz with 1024MB). We adopted the set of parameters shown in Table 2 for most of the cases when running the SDPA 5.0 with the HRVW/KSH/M direction selected.

### 6.1 Norm minimization problems

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

$$\left.\begin{array}{ll} \text{minimize} & \left\| \boldsymbol{F}_0 + \sum_{i=1}^{t} \boldsymbol{F}_i z_i \right\| \\ \text{subject to} & z_i \in \mathbb{R} \ (i = 1, 2, \ldots, t) \end{array}\right\}.$$

Table 2: Parameters for the SDPA 5.0 [6].

| | | |
|---|---|---|
| $\epsilon^*$ | tolerance for the relative duality gap | $10^{-6}$ |
| $\lambda^*$ | parameter for the initial point | $10^2$ |
| $\epsilon'$ | tolerance for the feasibility error | $10^{-7}$ |
| $\beta^*$ | parameter for the search direction for feasible points | 0.1 |
| $\bar{\beta}$ | parameter for the search direction for infeasible points | 0.2 |
| $\gamma^*$ | reduction factor of the step length | 0.9 |

We can reduce this problem to an SDP:

$$\left.\begin{array}{ll} \text{maximize} & -z_{t+1} \\ \text{subject to} & \displaystyle\sum_{i=1}^{t} \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_i^T \\ \boldsymbol{F}_i & \boldsymbol{O} \end{pmatrix} z_i + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} z_{t+1} + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix} \in \mathcal{S}_+^{q+r} \end{array}\right\}.$$

In our numerical experiments, all $\boldsymbol{F}_i$ $(i = 0, 1, \ldots, t)$ are pseudo-randomly generated dense matrices. Observe that as $q$ becomes large, the aggregate sparsity pattern and the extended sparsity pattern become dense for these problems.

Table 3 shows the comparisons among the "standard" method (SDPA), the conversion method applied before solving with the SDPA, and the completion method (SDPA-C) for a fixed $n = q + r$, and $m = t + 1$. On the other hand, Table 4 compares these three methods for increasing $n = q + r$. The entries "m.e." means that the memory exceeded.

Table 3: Numerical results on norm minimization problems I.

| | standard | | conversion | | completion | | sparsity | |
|---|---|---|---|---|---|---|---|---|
| | CPU | memory | CPU | memory | CPU | memory | aggregate | extended |
| $q$ | (s) | (MB) | (s) | (MB) | (s) | (MB) | (%) | (%) |
| 1 | 3302.2 | 321 | 4.4 (7.3) | 10 | 100.4 | 5 | 0.30 | 0.30 |
| 2 | 4159.4 | 321 | 7.9 (5.9) | 16 | 158.5 | 6 | 0.50 | 0.50 |
| 5 | 5200.7 | 321 | 28.0 (7.3) | 32 | 380.4 | 10 | 1.10 | 1.10 |
| 10 | 6688.0 | 321 | 100.3 (8.1) | 58 | 699.9 | 18 | 2.08 | 2.09 |
| 20 | 6630.4 | 321 | 445.5 (11.6) | 113 | 2491.7 | 45 | 4.02 | 4.06 |
| 50 | 7123.2 | 321 | 3256.5 (25.7) | 249 | — | — | 9.60 | 9.84 |

Problem size: $n = q + r$ =1000, $m = t + 1$ =11.

In the case of norm minimization problems, the conversion method performed much better than the other two methods for $n = q + r = 1000$, specially when $q \ll r$. The completion method is also faster than the "standard" method excepting the last case where we ran out of time. When $n$ increases, we can only solve these problems by the completion method because of the memory requirement. For all the cases including the results for SDP relaxations of quadratic programs with box constraints, max-cut problems over lattice graphs, graph partition problems over lattice graphs, and the DIMACS implementation challenge problems, the amount of used memory for the completion method is extremely small if compared with the "standard" method, and even less than the conversion method.

Table 4: Numerical results on norm minimization problems II.

| | standard | | conversion | | completion | | sparsity | |
|---|---|---|---|---|---|---|---|---|
| | CPU | memory | CPU | memory | CPU | memory | aggregate | extended |
| $q$ | (s) | (MB) | (s) | (MB) | (s) | (MB) | (%) | (%) |
| 5 | m.e. | m.e. | m.e. | m.e. | 1621.3 | 18 | 0.55 | 0.55 |
| 5 | m.e. | m.e. | m.e. | m.e. | 3822.0 | 26 | 0.37 | 0.37 |
| 5 | m.e. | m.e. | m.e. | m.e. | 7933.7 | 34 | 0.27 | 0.27 |
| 5 | m.e. | m.e. | m.e. | m.e. | 14603.3 | 41 | 0.22 | 0.22 |

Problem size: $n = q + r =$2000, 3000, 4000, and 5000, respectively, $m = t + 1 =$11.

## 6.2 Quadratic programs with box constraints

Let $\boldsymbol{Q} \in \mathcal{S}^n$ and $\boldsymbol{q} \in \mathbb{R}^n$. The quadratic program with a box constraint is defined as:

$$\left.\begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \\ \text{subject to} & -1 \leq x_i \leq 1 \ (i = 1, 2, \cdots, n) \end{array}\right\}.$$

We have the following SDP relaxation for the above problem

$$\left.\begin{array}{ll} \text{minimize} & \dfrac{1}{2}\begin{pmatrix} 0 & \boldsymbol{q}^T & \boldsymbol{0}^T \\ \boldsymbol{q} & \boldsymbol{Q} & \boldsymbol{O} \\ \boldsymbol{0} & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \bullet \boldsymbol{X} \\ \text{subject to} & \begin{pmatrix} 1 & \boldsymbol{0}^T & \boldsymbol{0}^T \\ \boldsymbol{0} & \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{0} & \boldsymbol{O} & \boldsymbol{O} \end{pmatrix} \bullet \boldsymbol{X} = 1, \\ & \begin{pmatrix} 0 & \boldsymbol{0}^T & \boldsymbol{0}^T \\ \boldsymbol{0} & \boldsymbol{E}_{ii} & \boldsymbol{O} \\ \boldsymbol{0} & \boldsymbol{O} & \boldsymbol{E}_{ii} \end{pmatrix} \bullet \boldsymbol{X} = 1 \quad (i = 1, 2, \cdots, n), \quad \boldsymbol{X} \in \mathcal{S}_+^{1+2n} \end{array}\right\}.$$

Here $\boldsymbol{E}_{ii} \in \mathcal{S}^n$ denotes the matrix with $(i, i)$th element one and all others zeros.

Table 5 compares the three methods applied to this particular class of SDPs. $\beta$ denotes the average number of nonzeros per column of the matrix $\boldsymbol{Q} \in \mathcal{S}^n$, and the vector $\boldsymbol{q} \in \mathbb{R}^n$. $n =$[1001,1000d] at the bottom of the table means that the primal matrix variable $\boldsymbol{X}$ has a block matrix of size 1001×1001 and a diagonal matrix of size 1000×1000.

Table 5: Numerical results on SDP relaxations of quadratic programs with box constraints.

| | standard | | conversion | | completion | | sparsity | |
|---|---|---|---|---|---|---|---|---|
| | CPU | memory | CPU | memory | CPU | memory | aggregate | extended |
| $\beta$ | (s) | (MB) | (s) | (MB) | (s) | (MB) | (%) | (%) |
| 3.0 | 3201.3 | 316 | 1238.9 (16.3) | 180 | 758.3 | 18 | 0.50 | 2.83 |
| 3.5 | 3121.4 | 316 | 1673.5 (24.1) | 214 | 877.8 | 22 | 0.55 | 4.56 |
| 4.0 | 3059.2 | 316 | 1916.7 (34.2) | 245 | 1088.1 | 29 | 0.60 | 6.43 |
| 4.5 | 2898.8 | 316 | 2129.4 (43.7) | 255 | 1421.4 | 37 | 0.66 | 8.55 |
| 5.0 | 3058.4 | 316 | 2551.6 (71.6) | 275 | 2099.3 | 57 | 0.70 | 10.41 |

Problem size: $n =$[1001,1000d], $m =$1001.

In the case of SDP relaxations of quadratic programs with box constraints, the completion method performed better than the other methods. Notice that the sparsity of the extended sparsity pattern is sensitive to the sparsity of the original quadratic program.

## 6.3 Max-cut problems over lattice graphs

Let $G(V,E)$ be a lattice graph of size $k_1 \times k_2$ with vertex set $V = \{1,2,\ldots,n\}$, and edge set $E \subseteq \{(i,j) : i,j \in V,\ i<j\}$ such that $n$ is equal to $k_1k_2$ and $|E|$ is equal to $2k_1k_2 - k_1 - k_2$. We assign a pseudo-randomly generated weight $C_{ij} = C_{ji}$ to each edge $(i,j) \in E$. The maximum cut problem is a problem of finding a partition $(L,R)$ of $V$ which maximizes the *cut* $c(L,R) = \sum_{i \in L, j \in R} C_{ij}$.

Let us define $\boldsymbol{C} \in \mathcal{S}^n$ by $C_{ji} = C_{ij}$ $((i,j) \in E)$ and $C_{ij} = 0$ $((i,j) \notin E)$, and $\boldsymbol{A}_0 \in \mathcal{S}^n$ by $\boldsymbol{A}_0 = \mathrm{diag}(\boldsymbol{Ce}) - \boldsymbol{C}$, where $\boldsymbol{e} \in \mathbb{R}^n$ denotes the vector of ones and $\mathrm{diag}(\boldsymbol{Ce})$ the diagonal matrix with diagonal equal to $\boldsymbol{Ce} \in \mathbb{R}^n$. Then we can obtain the following SDP relaxation of the maximum cut problem.

$$\left.\begin{array}{ll} \text{minimize} & -\boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1/4 \ (i = 1,2,\ldots,n),\ \ \boldsymbol{X} \in \mathcal{S}^n_+ \end{array}\right\}.$$

Table 6 compares the three methods for this problem. As $k_1$ becomes large, the aggregate sparsity patterns remain sparse, though the extended sparsity patterns become dense for these SDPs.

Table 6: Numerical results on SDP relaxations of the maximum cut problems.

| | standard | | conversion | | completion | | sparsity | |
|---|---|---|---|---|---|---|---|---|
| | CPU | memory | CPU | memory | CPU | memory | aggregate | extended |
| $k_1 \times k_2$ | (s) | (MB) | (s) | (MB) | (s) | (MB) | (%) | (%) |
| 2 × 500 | 2606.6 | 315 | 145.3 (2.0) | 34 | 184.0 | 16 | 0.40 | 0.50 |
| 4 × 250 | 2823.0 | 315 | 250.4 (3.8) | 52 | 214.0 | 17 | 0.45 | 0.86 |
| 5 × 200 | 2679.6 | 315 | 339.5 (4.4) | 57 | 233.4 | 18 | 0.46 | 1.03 |
| 8 × 125 | 2659.8 | 315 | 211.8 (11.0) | 70 | 256.9 | 19 | 0.47 | 1.38 |
| 10 × 100 | 2806.6 | 315 | 239.2 (12.8) | 90 | 320.6 | 19 | 0.48 | 1.57 |
| 20 × 50 | 2806.2 | 315 | 520.2 (18.0) | 170 | 348.1 | 22 | 0.49 | 2.12 |
| 25 × 40 | 2798.3 | 315 | 672.1 (58.1) | 180 | 338.0 | 22 | 0.49 | 2.25 |

Problem size: $n$ =1000, $m$ =1000.

For the SDP relaxations of the maximum cut problem over lattice graphs, the conversion and the completion methods performed much better than the "standard" method.

## 6.4 Graph partition problems over lattice graphs

Consider the same lattice graph $G(V,E)$ defined in the previous subsection, and assume in addition that $n$ is an even number. The graph partition problem is a problem of finding a partition $(L,R)$ of $V$ with the same cardinality, *i.e.*, $|L| = |R| = n/2$, which minimizes the cut $c(L,R) = \sum_{i \in L, j \in R} C_{ij}$. In a similar way to the maximum cut problem, we can derive an SDP relaxation of the graph partition problem:

$$\left.\begin{array}{ll} \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1/4 \ (i = 1, 2, \ldots, n), \ \boldsymbol{E} \bullet \boldsymbol{X} = 0, \ \boldsymbol{X} \in \mathcal{S}^n_+ \end{array}\right\}. \tag{13}$$

Here $\boldsymbol{A}_0$ and $\boldsymbol{E}_{ii}$ $(i = 1, 2, \ldots, n)$ are the same matrices as defined previously, and $\boldsymbol{E}$ denotes the $n \times n$ matrix with all elements one. Although (13) involves a dense data matrix $\boldsymbol{E}$, we can obtain an equivalent SDP with sparse aggregate sparsity pattern applying an appropriate congruent transformation to it [8, section 6].

Table 7 compares the three methods for the transformed problems. As $k_1$ becomes large, the aggregate sparsity patterns remain sparse, though the extended sparsity patterns become dense for them.

Table 7: Numerical results on SDP relaxations of the graph partition problems.

| | standard | | conversion | | completion | | sparsity | |
|---|---|---|---|---|---|---|---|---|
| | CPU | memory | CPU | memory | CPU | memory | aggregate | sparsity |
| $k_1 \times k_2$ | (s) | (MB) | (s) | (MB) | (s) | (MB) | (%) | (%) |
| $2 \times 500$ | 3203.4 | 315 | 159.8 (2.9) | 40 | 440.3 | 18 | 0.70 | 0.70 |
| $4 \times 250$ | 3157.6 | 315 | 157.8 (5.4) | 46 | 563.5 | 23 | 1.05 | 1.10 |
| $5 \times 200$ | 3028.9 | 315 | 174.4 (6.6) | 51 | 668.2 | 27 | 1.06 | 1.30 |
| $8 \times 125$ | 3538.7 | 315 | 318.1 (14.8) | 75 | 807.3 | 25 | 1.07 | 2.39 |
| $10 \times 100$ | 3218.1 | 315 | 463.6 (24.9) | 98 | 893.2 | 27 | 1.07 | 2.94 |
| $20 \times 50$ | 3079.9 | 315 | 1136.5 (19.3) | 194 | 1323.8 | 32 | 1.08 | 4.97 |
| $25 \times 40$ | 3123.9 | 315 | 800.4 (71.1) | 179 | 1334.2 | 34 | 1.08 | 5.31 |

Problem size: $n$ =1000, $m$ =1001.

For the SDP relaxations of the graph partition problems over lattice graphs, the conversion method performed better than the other methods.

## 6.5 DIMACS implementation challenge problems

We have selected two SDPs from the 7th DIMACS implementation challenge problem library (semidefinite and related optimization problems). These problems are SDP relaxations of max-cut problems from the Ising model of spin glasses: *toruspm3-8.50* and *torusg3-8*. The aggregate sparsity patterns are sparse, but the extended sparsity patterns become dense for them. Although the other DIMACS implementation challenge problems are sparse SDPs, their aggregate sparsity patterns and/or their extended sparsity patterns become dense, and therefore, they become out of scope of our approach.

Table 8 compares the three methods for these two problems which have the same sparse aggregate sparsity patterns, and therefore the same extended sparsity patterns. Table 9 presents the termination error measure according to the DIMACS implementation challenge criteria.

$$\begin{array}{lcl} \text{duality gap} & \equiv & \max(0, \boldsymbol{A}_0 \bullet \boldsymbol{X} - \sum_{p=1}^m b_p z_p), \\ \text{primal feasibility error} & \equiv & \left(\sum_{p=1}^m (\boldsymbol{A}_p \bullet \boldsymbol{X} - b_p)^2\right)^{1/2} / (1 + \max_{p=1,2,\ldots,m} |b_p|), \\ \text{dual feasibility error} & \equiv & \| \sum_{p=1}^m \boldsymbol{A}_p z_p + \boldsymbol{Y} - \boldsymbol{A}_0 \|_F / (1 + \max_{i,j=1,2,\ldots,n} |[\boldsymbol{A}_0]_{ij}|). \end{array}$$

Table 8: Numerical results on DIMACS implementation challenge problems.

| | standard | | conversion | | completion | | sparsity | |
|---|---|---|---|---|---|---|---|---|
| Problem name | CPU (s) | memory (MB) | CPU (s) | memory (MB) | CPU (s) | memory (MB) | aggregate (%) | sparsity (%) |
| toruspm3-8-50 | 203.4 | 85 | 244.8 (8.9) | 86 | 230.0 | 21 | 1.37 | 13.92 |
| torusg3-8 | 294.0 | 85 | 347.0 (8.8) | 86 | 262.1 | 21 | 1.37 | 13.92 |

Problem size: $n$ =512, $m$ =512.

Table 9: Termination errors of the DIMACS implementation challenge problems.

| | standard | | | conversion | | | completion | | |
|---|---|---|---|---|---|---|---|---|---|
| Problem name | duality gap | primal feas. | dual feas. | duality gap | primal feas. | dual feas. | duality gap | primal feas. | dual feas. |
| toruspm3-8-50 | $2.02e{-}04$ | $3.02e{-}13$ | $1.80e{-}15$ | $1.96e{-}04$ | $5.44e{-}12$ | $1.22e{-}13$ | $2.88e{-}04$ | $2.52e{-}15$ | $3.07e{-}12$ |
| torusg3-8 | $3.82e{+}01$ | $2.88e{-}08$ | $2.03e{-}15$ | $3.66e{+}01$ | $2.45e{-}08$ | $1.03e{-}07$ | $3.08e{+}01$ | $5.20e{-}15$ | $1.38e{-}12$ |

The "standard" method performed better for the toruspm3-8-50 problem, though the completion method performed better for the torusg3-8 problem. Considering the used memory, the completion method required the smallest amount of memory, but the conversion method required a slightly large amount than the "standard" method.

These problems show that it is not advantageous to use the conversion method and/or the completion method when the extended sparsity patterns are not sparse.

One way to deal with this drawback can be by combining these three methods in a single software which selects the most appropriate method for each SDP by considering the sparsity structure of the problem, and by estimating the necessary flops for each method (see item (C) in Concluding remarks).

## 7 Concluding remarks

This article supplements Part I [8]. Here we have mostly focused on the implementation details of the two methods proposed in [8] to solve large scale and sparse SDPs exploiting their aggregate sparsity patterns over their data matrices via matrix completion.

In the *conversion method*, we have utilized the clique tree, which is related to the sparsity structure of a given SDP, to obtain a "good" conversion of the given SDP into an equivalent SDP having multiple but smaller size positive semidefinite matrix variables.

In the *completion method*, we have proposed new computational formulae which only involve sparse matrices and construct internally positive definite matrix completions of the primal matrix variables. These formulae are directly implemented inside the primal-dual path-following interior-point method to solve a given SDP without converting it.

Some numerical experiments have been given to show the advantage of using these two methods over the standard primal-dual path-following interior-point method for large scale and sparse SDPs.

Finally, we address some implementation oriented issues that can improve the conversion and completion methods.

(A) In the conversion method, we have proposed a very simple heuristic algorithm (Algorithm 4.4) to merge the maximal cliques in the clique tree. One can consider alternate strategies to merge the cliques like repeating successively Algorithm 4.4 for decreasing $\sigma$'s in (7) or sharping criterion (7) by estimating the flops of one iteration of the primal-dual interior-point method [7]. However, deriving a theoretical support for "good" conversions seems a further step due to the combinatorial nature and computational complexity of the problem.

(B) A further change in the data structure can improve the performance of the completion method through an efficient computation of the coefficient matrix $\boldsymbol{B}$ of the system of linear equations (8). In the SDPA-C, each nonzero element of $\boldsymbol{A}_p$ $(p = 0, 1, \ldots, m)$ is represented by its element and indices of its row and its column. Also the elements of each $\boldsymbol{A}_p$ are organized in increasing order of the column indices. Hence, we can efficiently extract the nonzero elements of each column of each $\boldsymbol{A}_p$, but not a specific $(i, j)$ nonzero element. Changing the data structure so that it permits the latter operations at low cost, we can carry out the computation of $\boldsymbol{B}$ more efficiently using Algorithm 7.1 or Algorithm 7.2 instead of Algorithm 5.1.

**Algorithm 7.1: Alternate algorithm 1 for the computation of $\boldsymbol{B}$**

```
Set $\boldsymbol{B} = \boldsymbol{O}$
for $k, l = 1, 2, \cdots, n$
    Compute $\boldsymbol{v}_1 = \boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{e}_k$ and $\boldsymbol{v}_2 = \boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{e}_l$
    for $p = 1, 2, \cdots, m$
        Compute $t = \boldsymbol{v}_2^T\boldsymbol{A}_p\boldsymbol{v}_1$
        for $q = 1, 2, \cdots, m$ with $[\boldsymbol{A}_q]_{lk} \neq 0$
            Compute $t[\boldsymbol{A}_q]_{lk}$ and add to $B_{pq}$
        end(for)
    end(for)
end(for)
```

Algorithm 7.1 is more efficient when the data matrices $\boldsymbol{A}_p$ $(p = 0, 1, \ldots, m)$ and the extended sparsity pattern $F$ are sparse. On the other hand, Algorithm 7.2 works efficiently even when the extended sparsity pattern becomes mildly dense.

(C) A general user who wants to solve a given SDP might be interested in a code which automatically selects the most efficient method to solve the SDP. Unfortunately, the

**Algorithm 7.2: Alternate algorithm 2 of computation of $\boldsymbol{B}$**

```
Set $\boldsymbol{B} = \boldsymbol{O}$
for $p = 1, 2, \cdots, m$
    for $k = 1, 2, \cdots, n$
        Compute $\boldsymbol{v} = \boldsymbol{N}^{-T}\boldsymbol{N}^{-1}\boldsymbol{A}_p\boldsymbol{W}^{-T}\boldsymbol{D}\boldsymbol{W}^{-1}\boldsymbol{e}_k$
        for $l = 1, 2, \cdots, n$
            for $q = 1, 2, \cdots, m$ with $[\boldsymbol{A}_q]_{lk} \neq 0$
                Compute $v_l[\boldsymbol{A}_q]_{lk}$ and add to $B_{pq}$
            end(for)
        end(for)
    end(for)
end(for)
```

performance of the conversion and completion methods changes depending on the sparsity of the data matrices $\boldsymbol{A}_p$ $(p = 0, 1, \ldots, m)$ and on the extended sparsity pattern $F$. Therefore, detailed estimations of the flops for the proposed methods become indispensable for this purpose. We observe also that the standard primal-dual path-following interior-point method can be viewed as a particular case of the conversion method for this analysis (if the merging of the cliques works successfully until the end).

(D) In these two series of papers, we have proposed methods to efficiently solve SDPs for large $n$ (matrix size). On the other hand, the conjugate gradient (CG) method or the conjugate residual (CR) method can be used to solve the system of linear equations (8) when $m$ (number of linear constraints) becomes large [4, 15, 17, 22]. In the CG and CR methods, we need to multiply the coefficient matrix $\boldsymbol{B}$ by a vector several times, instead of computing each element of $\boldsymbol{B}$ and storing them as we did here. Then, we can employ similar algorithms to the ones in subsection 5.2 which explore the sparse factorizations and the sparsity of the SDPs to multiply the coefficient matrix $\boldsymbol{B}$ by a vector. Combining the completion method, and the CG or CR methods, we can solve sparse SDPs with $n$ and $m$ large, though, it is known that the CG and CR methods might fail to converge in some cases.

## Acknowledgments

The authors thank Dr. John R. Gilbert of the Xerox Palo Alto Research Center for suggesting us the ordering code [12], and Mr. Satoshi Nakamura of Tokyo Institute of Technology for helping us with the numerical experiments.

## References

[1] C. Ashcraft, D. Pierce, D. K. Wah and J. Wu, The reference manual for SPOOLES, release 2.2: An object oriented software library for solving sparse linear systems of equations, Boeing Shared Services Group, P. O. Box 24346, Mail

Stop 7L-22, Seattle, WA 98124, January 1999; Available at http://netlib.bell-labs.com/netlib/linalg/spooles/spooles.

[2] J. R. S. Blair and B. Peyton, An introduction to chordal graphs and clique trees, in: A. George, J. R. Gilbert and J. W. H. Liu, eds., *Graph Theory and Sparse Matrix Computation* (Springer-Verlag, New York, 1993) 1–29.

[3] B. Borchers, CSDP 2.3 user's guide, *Optimization Methods & Software* 11 & 12 (1999) 597–611; Available at http://www.nmt.edu/~borchers/csdp.html.

[4] C. Choi and Y. Ye, Solving sparse semidefinite programs using the dual scaling algorithm with an iterative solver, Department of Management Sciences, The University of Iowa, Iowa City, IO 52242 March 2000.

[5] K. Fujisawa, M. Fukuda, M. Kojima and K. Nakata, Numerical evaluation of SDPA (SemiDefinite Programming Algorithm), in: H. Frenk, K. Roos, T. Terlaky and S. Zhang, eds., *High Performance Optimization* (Kluwer Academic Publishers, Dordrecht, 1999) 267–301.

[6] K. Fujisawa, M. Kojima and K. Nakata, SDPA (Semidefinite Programming Algorithm) — User's Manual —, Technical Report B-308, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552, Japan, December 1995 (revised August 1996); Available at ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA.

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

[8] 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.

[9] 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.

[10] C. Helmberg, F. Rendl, R. J. Vanderbei and H. Wolkowicz, An interior-point method for semidefinite programming, *SIAM Journal on Optimization* 6 (1996) 342–361.

[11] C. R. Johnson, Matrix completion problems: A survey, *Proceedings of Symposia in Applied Mathematics* 40 (1990) 171–198.

[12] G. Karypis and V. Kumar, METIS — A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0 —, Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis, MN 55455, September 1998; Available at http://www-users.cs.umn.edu/~karypis/metis/metis.

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

[14] J. G. Lewis, B. W. Peyton and A. Pothen, A fast algorithm for reordering sparse matrices for parallel factorization, *SIAM Journal on Scientific and Statistical Computing* 10 (1989) 1146–1173.

[15] C.-J. Lin and R. Saigal, An incomplete Cholesky factorization for dense symmetric positive definite matrices, *BIT* 40 (2000) 536–558.

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

[17] K. Nakata, K. Fujisawa and M. Kojima, Using the conjugate gradient method in interior-point methods for semidefinite programs (in Japanese), *Proceedings of the Institute of Statistical Mathematics* 46 (1998) 297–316.

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

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

[20] R. E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM Journal on Computing* 13 (1984) 566–579.

[21] M. J. Todd, A study of search directions in primal-dual interior-point methods for semidefinite programming, *Optimization Methods and Software* 11 & 12 (1999) 1–46.

[22] K. C. Toh and M. Kojima, Solving some large scale semidefinite programs via the conjugate residual method, *SIAM Journal on Optimization* to appear.

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

[24] L. Vandenberghe and S. Boyd, Semidefinite Programming, *SIAM Review* 38 (1996) 49–95.