Fast ADMM for Semidefinite Programs with Chordal Sparsity

Yang Zheng[†,1], Giovanni Fantuzzi[†,2], Antonis Papachristodoulou[1], Paul Goulart[1], Andrew Wynn[2]

***Abstract*— Many problems in control theory can be formulated as semidefinite programs (SDPs). For large-scale SDPs, it is important to exploit the inherent sparsity to improve the scalability. This paper develops efficient first-order methods to solve SDPs with chordal sparsity based on the alternating direction method of multipliers (ADMM). We show that chordal decomposition can be applied to either the primal or the dual standard form of a sparse SDP, resulting in scaled versions of ADMM algorithms with the same computational cost. Each iteration of our algorithms consists of a projection on the product of small positive semidefinite cones, followed by a projection on an affine set, both of which can be carried out efficiently. Our techniques are implemented in CDCS, an open source add-on to MATLAB. Numerical experiments on large-scale sparse problems in SDPLIB and random SDPs with block-arrow sparse patterns show speedups compared to some common state-of-the-art software packages.**

I. Introduction

Semidefinite programs (SDPs) are a type of convex optimization problems over the cone of positive semidefinite matrices. Given $b \in \mathbb{R}^m$, $C \in \mathbb{S}^n$, and matrices $A_1, \ldots, A_m \in \mathbb{S}^n$ that define the operators

$$\mathcal{A}(X) = \begin{bmatrix} \langle A_1, X\rangle \\ \vdots \\ \langle A_m, X\rangle \end{bmatrix}, \quad \mathcal{A}^*(y) = \sum_{i=1}^m A_i y_i,$$

SDPs are typically written in the *standard primal form*

$$\begin{aligned} \min_{X} \quad & \langle C, X\rangle \\ \text{subject to} \quad & \mathcal{A}(X) = b, \\ & X \in \mathbb{S}^n_+, \end{aligned} \tag{1}$$

or in the *standard dual form*

$$\begin{aligned} \max_{y,Z} \quad & \langle b, y\rangle \\ \text{subject to} \quad & \mathcal{A}^*(y) + Z = C, \\ & Z \in \mathbb{S}^n_+. \end{aligned} \tag{2}$$

In the above and throughout this work, $\mathbb{R}^m$ is the usual m-dimensional Euclidean space, $\mathbb{S}^n$ is the space of $n \times n$ symmetric matrices, $\mathbb{S}^n_+$ is the subspace of positive semidefinite matrices, and $\langle \cdot, \cdot \rangle$ denotes the inner product in the appropriate space, *i.e.*, $\langle x, y\rangle = x^T y$ for $x, y \in \mathbb{R}^m$ and $\langle X, Y\rangle = \mathrm{trace}(XY)$ for $X, Y \in \mathbb{S}^n$.

SDPs have found applications in a wide range of fields, including control theory, machine learning, combinatorics, and operations research [1]. Moreover, other common types of optimization problems, *e.g.*, linear, quadratic, and second-order-cone programs, are particular instances of SDPs [2]. Small and medium-sized SDPs can be solved up to any arbitrary precision in polynomial time [3] using efficient second-order interior-point methods [4], [5]. However, many problems of practical interest are too large to be addressed by the current state-of-the-art interior-point algorithms, largely due to memory and CPU time constraints.

One of the main approaches to address this shortcoming is to abandon interior-point methods, in favour of faster, first-order methods with modest accuracy. For instance, Wen *et al.* proposed an alternating direction augmented Lagrangian method for large-scale SDPs in the dual standard form [6]. More recently, O'Donoghue *et al.* developed a first-order operator-splitting method to solve the homogeneous self-dual embedding of the primal-dual pair of a conic program, which has the advantage of being able to provide primal or dual certificates of infeasibility [7]. An implementation of this method is available in the C package SCS [8].

The second major approach is based on the observation that the large-scale SDPs encountered in applications are often structured and/or sparse [1]. Exploiting sparsity in SDPs is an active and challenging area of research [9], one main difficulty being that the optimal solution is typically dense despite the sparsity of the problem data. If, however, the aggregate sparsity pattern of the data is *chordal* (or has efficient *chordal extensions*), Grone's theorem [10] and Agler's theorem [11] allow us to replace the positive semidefinite constraint with a set of smaller semidefinite constraints, plus an additional set of equality constraints. Having reduced the size of the semidefinite variables, the converted SDP can be solved more efficiently than the original problem in some cases. These ideas underly the *domain-space* and the *range-space* conversion techniques of [12], [13], implemented in the MATLAB package SparseCoLO [14].

However, the addition of equality constraints to the SDP often offsets the benefit of working with smaller semidefinite cones. One possible solution is to exploit the properties of chordal sparsity pattern directly in the interior-point methods: Fukuda *et al.* used Grone's positive definite completion theorem [10] to develop a primal-dual path-following method for SDPs [12]; Burer proposed a nonsymmetric primal-dual interior-point method using Cholesky factors of the dual variable and maximum determinant completion of the primal

[†]Y. Zheng and G. Fantuzzi contributed equally to this work. Y. Zheng is supported by the Clarendon Scholarship and the Jason Hu Scholarship. G. Fantuzzi was partially supported by the EPSRC grant EP/J010537/1.

[1]Department of Engineering Science, University of Oxford, Parks Road, Oxford, OX1 3PJ, United Kingdom (e-mail: yang.zheng@eng.ox.ac.uk; paul.goulart@eng.ox.ac.uk; antonis@eng.ox.ac.uk).

[2]Department of Aeronautics, Imperial College London, South Kensington Campus, London, SW7 2AZ, United Kingdom (e-mail: gf910@ic.ac.uk; a.wynn@imperial.ac.uk).

variable [15]; and Andersen *et al.* developed fast recursive algorithms for SDP problems with chordal sparsity [16]. Another option is to solve the decomposable SDP via first-order methods: Sun *et al.* proposed a first-order splitting algorithm for decomposable conic programs, including SDPs with chordal sparsity [17]; Kalbat & Lavaei applied the alternating direction method of multipliers (ADMM) to solve a special class of SDPs with fully decomposable constraints [18]; Madani *et al.* developed a highly-parallelizable ADMM algorithm for sparse SDPs with inequality constraints with optimal power flow applications [19].

In this work, we adopt the strategy of exploiting sparsity using first-order algorithms in the spirit of [17], [18], [19], and develop efficient ADMM algorithms to solve large-scale sparse SDPs in either primal or dual standard form. The contributions of this paper are three-fold:

1) We show that the ADMM and chordal decomposition techniques can be combined to solve sparse SDPs in either primal or dual standard form. The resulting primal and dual algorithms are scaled versions of each other. This provides a conversion framework suitable for the application of first-order methods, analogous to that of [12], [13] for interior-point methods.
2) In each iteration, the positive semidefinite (PSD) constraint $X \in \mathbb{S}^n_+$ (or $Z \in \mathbb{S}^n_+$) is enforced via parallel projections onto small PSD cones. The affine constraints $\mathcal{A}(X) = b$ (or $\mathcal{A}^*(y) + Z = C$) are enforced via an equality-constrained quadratic program, and the corresponding KKT system matrix only depends on the problem data. The preferred matrix factorization can thus be cached before iterating the ADMM algorithm.
3) We implement our techniques in the open source MATLAB solver CDCS (Cone Decomposition Conic Solver). We ran numerical simulations on random SDPs with block-arrow sparse patterns, and on four large-scale sparse problems in SDPLIB [20]. The results demonstrate the efficiency of our algorithms compared to other commonly used software packages.

The rest of this paper is organized as follows. Section II reviews some key notions regarding chordal sparsity and decomposition techniques. We show how to apply the ADMM to primal and dual standard-form SDPs in Sections III and IV, respectively, and report some numerical experiments in Section V. Finally, Section VI offers concluding remarks.

II. Preliminaries: Chordal Decomposition and the ADMM Algorithm

A. Chordal graphs

Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be an undirected graph with vertices $\mathcal{V} = \{1, 2, \ldots, n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A subset of vertices $\mathcal{C} \subseteq \mathcal{V}$ such that $(i, j) \in \mathcal{E}$ for any distinct vertices $i, j \in \mathcal{C}$ is called a *clique*, and the number of vertices in $\mathcal{C}$ is denoted by $|\mathcal{C}|$. If $\mathcal{C}$ is not a subset of any other clique, then it is referred to as a *maximal clique*.

A cycle of length k in $\mathcal{G}$ is a set of pairwise distinct vertices $\{v_1, v_2, \ldots, v_k\} \subset \mathcal{V}$ such that $(v_k, v_1) \in \mathcal{E}$ and $(v_i, v_{i+1}) \in \mathcal{E}$ for $i = 1, \ldots, k-1$. A chord is an edge joining two non-adjacent vertices in a cycle. An undirected graph $\mathcal{G}$ is called *chordal* if every cycle of length greater than or equal to four has at least one chord. Note that if $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is not chordal, it can be *chordal extended*, *i.e.*, we can construct a chordal graph $\mathcal{G}'(\mathcal{V}, \mathcal{E}')$ by adding additional edges to $\mathcal{E}$ such that $\mathcal{G}'$ is chordal. Although finding the chordal extension with the minimum number of additional edges is an NP-complete problem [21], good chordal extensions can be computed efficiently using several heuristics; see *e.g.*, [22].

B. Sparse matrices defined by graphs

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph, and assume that $(i, i) \in \mathcal{E}$, *i.e.*, each node has a self-loop. We say that X is a sparse symmetric matrix defined by $\mathcal{G}$ if $X_{ij} = X_{ji} = 0$ whenever $(i, j) \notin \mathcal{E}$. The spaces of sparse and positive semidefinite sparse symmetric matrices defined by $\mathcal{G}$ are

$$\begin{aligned}\mathbb{S}^n(\mathcal{E}, 0) =& \{X \in \mathbb{S}^n \mid X_{ij} = X_{ji} = 0 \text{ if } (i, j) \notin \mathcal{E}\},\\ \mathbb{S}^n_+(\mathcal{E}, 0) =& \{X \in \mathbb{S}^n(\mathcal{E}, 0) \mid X \succeq 0\}.\end{aligned}$$

Similarly, we say that X is a partial symmetric matrix defined by $\mathcal{G}$ if $X_{ij} = X_{ji}$ are given when $(i, j) \in \mathcal{E}$, and arbitrary otherwise. Moreover, we say that M is a positive semidefinite completion of the partial symmetric matrix X if there exists $M \succeq 0$ such that $M_{ij} = X_{ij}$ when $(i, j) \in \mathcal{E}$. We can then define the spaces

$$\begin{aligned}\mathbb{S}^n(\mathcal{E}, ?) =& \{X \in \mathbb{S}^n \mid X_{ij} = X_{ji} \text{ given if } (i, j) \in \mathcal{E}\},\\ \mathbb{S}^n_+(\mathcal{E}, ?) =& \{X \in \mathbb{S}^n(\mathcal{E}, ?) \mid\\ & \qquad \exists M \succeq 0,\, M_{ij} = X_{ij}, \forall (i, j) \in \mathcal{E}\}.\end{aligned}$$

Finally, given a clique $\mathcal{C}_k$ of $\mathcal{G}$, we let $E_k \in \mathbb{R}^{|\mathcal{C}_k| \times n}$ be the matrix with $(E_k)_{ij} = 1$ if $\mathcal{C}_k(i) = j$ and zero otherwise, where $\mathcal{C}_k(i)$ is the i-th vertex in $\mathcal{C}_k$, sorted in the natural ordering. Given $X \in \mathbb{S}^n$, the submatrix of X defined by the clique $\mathcal{C}_k$ is given by $E_k X E_k^T \in \mathbb{S}^{|\mathcal{C}_k|}$.

C. Chordal decomposition of positive semidefinite matrices

Given any undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the spaces $\mathbb{S}^n_+(\mathcal{E}, ?)$ and $\mathbb{S}^n_+(\mathcal{E}, 0)$ are cones, and they are dual to each other [16], [22]. If $\mathcal{G}$ is chordal, then $\mathbb{S}^n_+(\mathcal{E}, ?)$ and $\mathbb{S}^n_+(\mathcal{E}, 0)$ can be expressed in terms of several coupled smaller convex cones according to the following theorems:

Theorem 1 (Grone's theorem [10]): Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be a chordal graph, and let $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_p\}$ be the set of its maximal cliques. Then, $X \in \mathbb{S}^n_+(\mathcal{E}, ?)$ if and only if

$$X_k := E_k X E_k^T \in \mathbb{S}^{|\mathcal{C}_k|}_+$$

for all $k = 1, \ldots, p$.

Theorem 2 (Agler's theorem [11]): Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be a chordal graph, and let $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_p\}$ be the set of its maximal cliques. Then, $Z \in \mathbb{S}^n_+(\mathcal{E}, 0)$ if and only if there exist matrices $Z_k \in \mathbb{S}^{|\mathcal{C}_k|}_+$ for $k = 1, \ldots, p$ such that

$$Z = \sum_{k=1}^{p} E_k^T Z_k E_k.$$

Note that these results can be proven individually, but can also can be derived from each other using the duality of the cones $\mathbb{S}^n_+(\mathcal{E}, ?)$ and $\mathbb{S}^n_+(\mathcal{E}, 0)$ [22].

D. ADMM algorithm

The ADMM algorithm solves the optimization problem

$$\begin{array}{ll}\min & f(x)+g(y)\\ \text{subject to} & Ax+By=c,\end{array}$$

where f and g are convex functions, $x\in\mathbb{R}^{n_x}, y\in\mathbb{R}^{n_y}, A\in\mathbb{R}^{n_c\times n_x}, B\in\mathbb{R}^{n_c\times n_y}$ and $c\in\mathbb{R}^{n_c}$. Given a penalty parameter $\rho>0$ and a dual multiplier $z\in\mathbb{R}^{n_c}$, the ADMM algorithm minimizes the augmented Lagrangian

$$L_\rho(x,y,z)=f(x)+g(y)+\frac{\rho}{2}\left\|Ax+By-c+\frac{1}{\rho}z\right\|^2$$

with respect to the variables x and y separately, followed by a dual variable update:

$$x^{(n+1)}=\arg\min_x L_\rho(x,y^{(n)},z^{(n)}), \tag{3a}$$

$$y^{(n+1)}=\arg\min_y L_\rho(x^{(n+1)},y,z^{(n)}), \tag{3b}$$

$$z^{(n+1)}=z^{(n)}+\rho(Ax^{(n+1)}+By^{(n+1)}-c). \tag{3c}$$

The superscript (n) indicates that a variable is fixed to its value at the n-th iteration. ADMM is particularly suitable when the minimization with respect to each of the variables x and y in (3a) and (3b) can be carried out efficiently through closed-form expressions.

III. ADMM for Sparse Primal-form SDPs

This section introduces the ADMM algorithm for the primal-standard-form SDP (1), in which Grone's theorem is used to decompose the PSD constraint. We assume that (1) is sparse with an *aggregate sparsity pattern* described by the graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, meaning that $(i,j)\in\mathcal{E}$ if and only if the entry ij of at least one of the data matrices $C, A_0, \ldots, A_m$, is nonzero. We also assume that $\mathcal{G}$ is chordal (otherwise, it can be chordal extended), and that its maximal cliques $\mathcal{C}_1,\ldots,\mathcal{C}_p$ are small.

A. Reformulation and decomposition of the PSD constraint

In (1), only the entries of the matrix variable X corresponding to the graph edges $\mathcal{E}$ appear in the cost and constraint functions, so the constraint $X\in\mathbb{S}^n_+$ can be replaced by $X\in\mathbb{S}^n_+(\mathcal{E},?)$. Using Theorem 1, we can then reformulate (1) as

$$\begin{array}{ll}\min\limits_{X,X_1,\ldots,X_p} & \langle C,X\rangle\\ \text{subject to} & \mathcal{A}(X)=b,\\ & X_k-E_kXE_k^T=0, \quad k=1,\ldots,p,\\ & X_k\in\mathbb{S}^{|\mathcal{C}_k|}_+, \qquad\quad k=1,\ldots,p.\end{array} \tag{4}$$

In other words, we can decompose the original large semidefinite cone into multiple smaller cones, at the expense of introducing a set of consensus constraints between the variables.

To ease the exposition, we rewrite (4) in a vectorized form. Letting $\mathrm{vec}:\mathbb{S}^n\to\mathbb{R}^{n^2}$ be the usual operator mapping a matrix to the stack of its column, define the vectorized data

$$\begin{aligned}c&:=\mathrm{vec}(C),\\ A&:=\begin{bmatrix}\mathrm{vec}(A_0) & \ldots & \mathrm{vec}(A_m)\end{bmatrix}^T,\end{aligned}$$

the vectorized variables

$$\begin{aligned}x&:=\mathrm{vec}(X),\\ x_k&:=\mathrm{vec}(x_k), \quad k=1,\ldots,p,\end{aligned}$$

and the matrices

$$H_k:=E_k\otimes E_k, \tag{5}$$

such that

$$x_k=\mathrm{vec}(X_k)=\mathrm{vec}(E_kXE_k^T)=H_kx.$$

In other words, the matrices $H_1,\ldots,H_p$ are "entry-selector" matrices of 1's and 0's, whose rows are orthonormal, that project x onto the subvectors $x_1,\ldots,x_p$, respectively. If we denote the constraints $X_k\in\mathbb{S}^{|\mathcal{C}_k|}_+$ by $x_k\in\mathcal{S}_k$, we can rewrite (4) as

$$\begin{array}{ll}\min\limits_{x,x_1,\ldots,x_p} & \langle c,x\rangle\\ \text{subject to} & Ax=b,\\ & x_k=H_kx, \quad k=1,\ldots,p,\\ & x_k\in\mathcal{S}_k, \quad k=1,\ldots,p.\end{array} \tag{6}$$

B. The ADMM algorithm for primal SDPs

We start by moving the constraints $Ax=b$ and $x_k\in\mathcal{S}_k$ in (6) to the objective using the indicator functions $\delta_0(\cdot)$ and $\delta_{\mathcal{S}_k}(\cdot)$, respectively, *i.e.*, we write

$$\begin{array}{ll}\min\limits_{x,x_1,\ldots,x_p} & \langle c,x\rangle+\delta_0\left(Ax-b\right)+\sum\limits_{k=1}^p\delta_{\mathcal{S}_k}(x_k)\\ \text{subject to} & x_k=H_kx, \quad k=1,\ldots,p.\end{array} \tag{7}$$

This problem is in the standard form for the application of ADMM. Given a penalty parameter $\rho>0$ and a Lagrange multiplier λ_k for each constraint $x_k=H_kx$, we define the augmented Lagrangian

$$\begin{aligned}\mathcal{L}:=&\langle c,x\rangle+\delta_0\left(Ax-b\right)\\ &+\sum_{k=1}^p\left[\delta_{\mathcal{S}_k}(x_k)+\frac{\rho}{2}\left\|x_k-H_kx+\frac{1}{\rho}\lambda_k\right\|^2\right],\end{aligned} \tag{8}$$

and group the variables as

$$\begin{aligned}\mathcal{X}&:=\{x\},\\ \mathcal{Y}&:=\{x_1,\ldots,x_p\},\\ \mathcal{Z}&:=\{\lambda_1,\ldots,\lambda_p\}.\end{aligned}$$

Then, according to (3), each iteration of the ADMM consists of an $\mathcal{X}$-minimization step, a $\mathcal{Y}$-minimization step and an update of multipliers $\mathcal{Z}$.

*1) **Minimization over** $\mathcal{X}$:* Minimizing (8) over $\mathcal{X}$ is equivalent to the equality-constrained quadratic program

$$\begin{array}{ll}\min\limits_{x} & \langle c,x\rangle+\frac{\rho}{2}\sum\limits_{k=1}^p\left\|x_k^{(n)}-H_kx+\frac{1}{\rho}\lambda_k^{(n)}\right\|^2\\ \text{subject to} & Ax=b.\end{array} \tag{9}$$

Letting ρy be the multiplier for the equality constraint, and defining

$$D := \sum_{k=1}^{p} H_k^T H_k, \tag{10}$$

the optimality conditions for (9) can be written as the KKT system

$$\begin{bmatrix} D & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{p} H_k^T \left(x_k^{(n)} + \rho^{-1} \lambda_k^{(n)} \right) - \rho^{-1} c \\ b \end{bmatrix}. \tag{11}$$

Note that D is a diagonal matrix, because the rows of each matrix H_k are orthonormal unit vectors of 1's and 0's. Consequently, (11) can be solved efficiently, *e.g.*, by block elimination. Moreover, the coefficient matrix is the same at every iteration, which means its Cholesky (or LDL) factors can be pre-computed and cached before starting the ADMM iterations.

*2) **Minimization over** $\mathcal{Y}$:* Minimizing (8) over $\mathcal{Y}$ is equivalent to the p independent problems

$$\begin{array}{ll} \min\limits_{x_k} & \left\| x_k - H_k x^{(n+1)} + \rho^{-1} \lambda_k^{(n)} \right\|^2 \\ \text{subject to} & x_k \in \mathcal{S}_k. \end{array} \tag{12}$$

In terms of the original matrix variables $X_1, \ldots, X_p$, this amounts to a projection on the positive semidefinite cone. More precisely, if $\mathbb{P}_k$ denotes the projection onto $\mathbb{S}_+^{|\mathcal{C}_k|}$ and $\mathrm{mat}(\cdot) = \mathrm{vec}^{-1}(\cdot)$ we have

$$x_k^{(n+1)} = \mathrm{vec}\left\{ \mathbb{P}_k \left[\mathrm{mat}\left(H_k x^{(n+1)} - \rho^{-1} \lambda_k^{(n)} \right) \right] \right\}. \tag{13}$$

Since the projection $\mathbb{P}_k$ can be computed with an eigenvalue decomposition, and each cone $\mathbb{S}_+^{|\mathcal{C}_k|}$ is assumed to be small, the new variables $x_1^{(n+1)}, \ldots, x_p^{(n+1)}$ can be computed efficiently and in parallel.

Remark 1: The use of a global variable x to enforce the consensus constraints between the entries of the subvectors $x_1, \ldots, x_p$ (*i.e.*, $x_k = H_k x$) is fundamental because it allows to separate the conic constraints from affine constraint in (1), which makes the minimization over $\mathcal{Y}$ easy to compute and parallelizable.

*3) **Updating the multipliers** $\mathcal{Z}$:* The final step in the $(n+1)$-th ADMM iteration is to update the multipliers $\lambda_1 \ldots, \lambda_p$ with the usual gradient ascent rule: for each $k = 1, \ldots, p$,

$$\lambda_k^{(n+1)} = \lambda_k^{(n)} + \rho \left(x_k^{(n+1)} - H_k x^{(n+1)} \right). \tag{14}$$

This computation is cheap, and can be parallelized.

C. Summary & Stopping conditions

The ADMM algorithm is stopped after the n-th iteration if the relative primal/dual error measures

$$\epsilon_{\mathrm{p}} = \frac{\left(\sum_{k=1}^{p} \left\| x_k^{(n)} - H_k x^{(n)} \right\|^2 \right)^{1/2}}{\max\left\{ \left(\sum_{k=1}^{p} \left\| x_k^{(n)} \right\|^2 \right)^{1/2}, \left(\sum_{k=1}^{p} \left\| H_k x^{(n)} \right\|^2 \right)^{1/2} \right\}},$$

$$\epsilon_{\mathrm{d}} = \rho \times \frac{\left(\sum_{k=1}^{p} \left\| x_k^{(n)} - x_k^{(n-1)} \right\|^2 \right)^{1/2}}{\left(\sum_{k=1}^{p} \left\| \lambda_k^{(n)} \right\|^2 \right)^{1/2}},$$

are smaller than a specified tolerance, ϵ_{tol}; see [23] for more details on stopping conditions for a generic ADMM algorithm. In conclusion, the decomposed primal-standard-form SDP (6) can be solved using the steps summarized in Algorithm 1.

Algorithm 1 ADMM for decomposed primal form SDPs

1: Given $\rho > 0$, $\epsilon_{\mathrm{tol}} > 0$ and an initial guess $x^{(0)}$, $x_1^{(0)}, \ldots, x_p^{(0)}$, $\lambda_1^{(0)}, \ldots, \lambda_p^{(0)}$
2: Data preprocessing: chordal extension, chordal decomposition and factorizing the KKT system (11).
3: **while** $\max(\epsilon_{\mathrm{p}}, \epsilon_{\mathrm{d}}) \geq \epsilon_{\mathrm{tol}}$ **do**
4: Compute $x^{(n)}$ with (11).
5: **for** $k = 1, \ldots, p$ **do**
6: Compute $x_k^{(n)}$ with (13).
7: **end for**
8: **for** $k = 1, \ldots, p$ **do**
9: Compute $\lambda_k^{(n)}$ with (14).
10: **end for**
11: Update the residuals $\epsilon_{\mathrm{p}}, \epsilon_{\mathrm{d}}$.
12: **end while**

IV. ADMM for Sparse Dual-form SDPs

We now develop a similar ADMM algorithm for the dual-standard-form SDP (2), using Agler's theorem to decompose the positive semidefinite cone. As in Section III, we assume that the aggregate sparsity pattern of the problem data is described by the chordal graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$.

A. Reformulation of decomposition of the PSD constraint

The equality constraint in (2) implies that the semidefinite variable Z has the same sparsity pattern as the aggregate sparsity pattern of the problem data, *i.e.*, $Z \in \mathbb{S}_+^n(\mathcal{E}, 0)$. Applying Theorem 2, we then rewrite (2) as

$$\begin{array}{ll} \min\limits_{y, Z_1, \ldots, Z_p} & -\langle b, y \rangle \\ \text{subject to} & \mathcal{A}^*(y) + \sum\limits_{k=1}^{p} E_k^T Z_k E_k = C, \\ & Z_k \in \mathbb{S}_+^{|\mathcal{C}_k|}, \quad k = 1, \ldots, p. \end{array} \tag{15}$$

In (15), the original PSD constraint has been replaced by multiple smaller PSD constraints. However, it is not convenient to apply ADMM to this problem directly, because the positive semidefinite variables $Z_1, \ldots, Z_k$ in the equality constraint are weighted by the matrices E_k. Instead, we introduce a set of slack variables $V_1, \ldots, V_p$ such that $Z_k =$

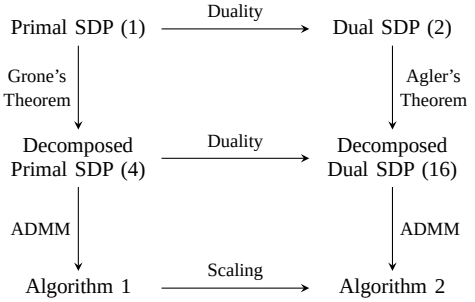

Fig. 1. Duality relationships between primal and dual SDPs, and the decomposed primal and dual SDPs.

$V_k, k = 1, \ldots, p$, and reformulate (15) as

$$\begin{aligned} \min_{y, Z_1, \ldots, Z_p, V_1, \ldots, V_p} \quad & -\langle b, y\rangle \\ \text{subject to} \quad & \mathcal{A}^*(y) + \sum_{k=1}^{p} E_k^T V_k E_k = C, \\ & Z_k - V_k = 0, \quad k = 1, \ldots, p, \\ & Z_k \in \mathbb{S}_+^{|\mathcal{C}_k|}, \quad k = 1, \ldots, p. \end{aligned} \tag{16}$$

Finally, we define $z_k := \text{vec}(Z_k)$ and $v_k := \text{vec}(V_k)$ for all $k = 1, \ldots, p$, and use the same vectorized notation as in Section III to rewrite (16) in the vectorized form

$$\begin{aligned} \min_{y, z_1, \ldots, z_p, v_1, \ldots, v_p} \quad & -\langle b, y\rangle \\ \text{subject to} \quad & A^T y + \sum_{k=1}^{p} H_k^T v_k = c, \\ & z_k - v_k = 0, \quad k = 1, \ldots, p, \\ & z_k \in \mathcal{S}_k, \quad k = 1, \ldots, p. \end{aligned} \tag{17}$$

Remark 2: Similar to the primal case, the use of a set of consensus equality constraints ($z_k - v_k = 0$) is also essential to the development of dual algorithm, so that the update of the conic variables in our ADMM algorithm are reduced to simple conic projections.

Remark 3: Although we have derived (17) by applying Theorem 2, it is not difficult to check that problem (17) is exactly the dual of the decomposed primal SDP (6). Consequently, our analysis provides a decomposition framework suitable for the application of first-order methods analogous of the conversion techniques for interior-point methods of [12], [13]. This elegant picture, in which the duality between the primal and dual SDP is inherited by the decomposed problems by virtue of the duality between Grone's and Agler's theorems, is shown in Figure 1.

B. The ADMM algorithm for dual SDPs

As in Section III, we start by moving all but the consensus equality constraints $z_k = v_k$, $k = 1, \ldots, p$, to the objective using indicator functions. This leads to

$$\begin{aligned} \min \ & -\langle b, y\rangle + \delta_0\left(c - A^T y - \sum_{k=1}^{p} H_k^T v_k\right) + \sum_{k=1}^{p} \delta_{\mathcal{S}_k}(z_k) \\ \text{subject to} \quad & z_k = v_k, \quad k = 1, \ldots, p. \end{aligned} \tag{18}$$

Given a penalty parameter $\rho > 0$ and a Lagrange multiplier λ_k for each of the constraints $z_k = v_k$, $k = 1, \ldots, p$, we define the augmented Lagrangian

$$\begin{aligned} \mathcal{L} := & -\langle b, y\rangle + \delta_0\left(c - A^T y - \sum_{k=1}^{p} H_k^T v_k\right) \\ & + \sum_{k=1}^{p}\left[\delta_{\mathcal{S}_k}(z_k) + \frac{\rho}{2}\left\| z_k - v_k + \frac{1}{\rho}\lambda_k \right\|^2\right], \end{aligned} \tag{19}$$

and group the variables as

$$\begin{aligned} \mathcal{X} &:= \{y, v_1, \ldots, v_p\}, \\ \mathcal{Y} &:= \{z_1, \ldots, z_p\}, \\ \mathcal{Z} &:= \{\lambda_1, \ldots, \lambda_p\}. \end{aligned}$$

*1) **Minimization over** $\mathcal{X}$:* Minimizing (19) over block $\mathcal{X}$ is equivalent to the equality-constrained quadratic program

$$\begin{aligned} \min_{y, v_1, \ldots, v_p} \quad & -\langle b, y\rangle + \frac{\rho}{2}\sum_{k=0}^{p}\left\| z_k^{(n)} - v_k + \frac{1}{\rho}\lambda_k^{(n)} \right\|^2 \\ \text{subject to} \quad & c - A^T y - \sum_{k=1}^{p} H_k^T v_k = 0. \end{aligned} \tag{20}$$

Let ρx be the multiplier for the equality constraint. After some algebra, the optimality conditions for (20) can be written as the KKT system

$$\begin{bmatrix} D & A^T \\ A & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c - \sum_{k=1}^{p} H_k^T\left(z_k^{(n)} + \rho^{-1}\lambda_k^{(n)}\right) \\ -\rho^{-1} b \end{bmatrix}, \tag{21}$$

plus a set of p uncoupled equations for the variables v_k,

$$v_k = z_k^{(n)} + \frac{1}{\rho}\lambda_k^{(n)} + H_k x, \quad k = 1, \ldots, p. \tag{22}$$

The KKT system (21) is the same as (11) after rescaling $x \mapsto -x$, $y \mapsto -y$, $c \mapsto \rho^{-1}c$ and $b \mapsto \rho b$. Consequently, the numerical cost of these operation is the same as in Section III-B.1, plus the cost of (22), which is cheap and can be parallelized. Moreover, as in Section III-B.1, the factors of the coefficient matrix required to solve the KKT system (21) can be pre-computed and cached before iterating the ADMM algorithm.

*2) **Minimization over** $\mathcal{Y}$:* Similarly to Section III-B.2, the variables $z_1, \ldots, z_p$ are updated with p independent projections,

$$z_k^{(n+1)} = \text{vec}\left\{\mathbb{P}_k\left[\text{mat}\left(v_k^{(n+1)} - \rho^{-1}\lambda_k^{(n)}\right)\right]\right\}, \tag{23}$$

Again, these projections can be computed efficiently and in parallel.

*3) **Updating the multipliers** $\mathcal{Z}$:* The multipliers λ_k, $k = 1, \ldots, p$, are updated with the usual gradient ascent rule

$$\lambda_k^{(n+1)} = \lambda_k^{(n)} + \rho\left(z_k^{(n+1)} - v_k^{(n+1)}\right). \tag{24}$$

Remark 4: Since the computational cost of (22) is the same as (14), all ADMM iterations for the decomposed dual-standard-form SDP (16) have the same cost as the

ADMM iterations for the decomposed primal-standard-form SDP (6), plus the cost of (24). However, if one minimizes the dual augmented Lagrangian (19) over $z_1, \ldots, z_p$ *before* minimizing it over $y, v_1, \ldots, v_p$, then (22) can be used to simplify the multiplier update equations to

$$\lambda_k^{(n+1)} = \rho H_k x^{(n+1)}, \quad k = 1, \ldots, p. \tag{25}$$

Given that the products $H_1 x, \ldots, H_p x$ have already been computed to update $v_1, \ldots, v_p$, updating the multipliers $\lambda_1, \ldots, \lambda_p$ requires only a scaling operation. Consequently, if one swaps the order of the minimization, the ADMM algorithms for the primal and dual standard form SDPs can be considered as scaled versions to each other, which have the same leading-order computational cost at each iteration.

C. Summary & Stopping conditions

Similarly to Section III-C, we stop our ADMM algorithm after the n-th iteration if the relative primal/dual error measures

$$\epsilon_{\mathrm{p}} := \frac{\left(\sum_{k=1}^{p} \left\| z_k^{(n)} - v_k^{(n)} \right\|^2\right)^{1/2}}{\max\left\{ \left(\sum_{k=1}^{p} \left\| z_k^{(n)} \right\|^2\right)^{1/2}, \left(\sum_{k=1}^{p} \left\| v_k^{(n)} \right\|^2\right)^{1/2} \right\}},$$

$$\epsilon_{\mathrm{d}} := \rho \times \frac{\left(\sum_{k=1}^{p} \left\| z_k^{(n)} - z_k^{(n-1)} \right\|^2\right)^{1/2}}{\left(\sum_{k=1}^{p} \left\| \lambda_k^{(n)} \right\|^2\right)^{1/2}},$$

are smaller than a specified tolerance, ϵ_{tol}. The complete ADMM algorithm to solve the decomposed dual-standard-form SDP (16) is summarized in Algorithm 2.

Algorithm 2 ADMM for decomposed dual form SDPs

Given $\rho > 0$, $\epsilon_{\mathrm{tol}} > 0$ and an initial guess $y^{(0)}$, $z_1^{(0)}, \ldots, z_p^{(0)}$,$\lambda_1^{(0)}, \ldots, \lambda_p^{(0)}$
Data preprocessing: chordal extension, chordal decomposition and factorizing the KKT system (21).
while $\max(\epsilon_{\mathrm{p}}, \epsilon_{\mathrm{d}}) \geq \epsilon_{\mathrm{tol}}$ **do**
  for $k = 1, \ldots, p$ **do**
    Compute $z_k^{(n)}$ with (23).
  end for
  Compute $y^{(n)}, x$ with (20).
  for $k = 1, \ldots, p$ **do**
    Compute $v_k^{(n)}$ with (22)
    Compute $\lambda_k^{(n)}$ with (25) (no cost).
  end for
  Update the residuals ϵ_{p} and ϵ_{d}.
end while

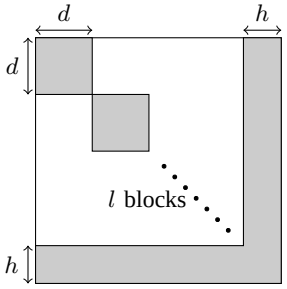

Fig. 2. Block-arrow sparsity pattern: the number of blocks, l; block size, d; the size of the arrow head, h.

V. Numerical Simulations

We have implemented our techniques in CDCS (Cone Decomposition Conic Solver), an open source MATLAB solver for partially decomposable conic programs. CDCS supports cartesian products of the following cones: $\mathbb{R}^n$, non-negative orthant, second-order cone, and the positive semidefinite cone. Currently, only chordal decomposition techniques for semidefinite cones are implemented, while the other supported cone types are not decomposed. Although some steps in Algorithms 1 and 2 are parallelizable, our current implementation is sequential. Moreover, we do not use the clique combination techniques suggested in [12], [24] for simplicity. Finally, CDCS scales the problem before the chordal decomposition step using the methods described in [7] and implemented in SCS [8]. Our codes can be downloaded from

`https://github.com/giofantuzzi/CDCS`.

We tested CDCS on four large-scale sparse problems ($n \geq 1000, m \geq 1000$) in SDPLIB [20], as well as on a series of randomly generated SDPs with the block-arrow sparse pattern, used as a benchmark in [17]. The performance of our method is compared to that of the interior-point solver SeDuMi [25] and of the first-order solver SCS [8]; both these solvers are used on the full problem (without decomposition) or the decomposed problem returned by SparseColO [14].

The comparison has two purposes: 1) the solution returned by SeDuMi is of high accuracy, so we can use it to assess the quality of the solution computed by CDCS; 2) SCS is a high performance first-order solver for general conic programs, so we can assess the unique features of our techniques in terms of chordal decomposition. We remark that the CPU time required by SeDuMi should not be compared to the other solvers, because the latter only aim to achieve moderate accuracy. In the experiments reported below, the termination tolerance for CDCS and SCS was set as $\epsilon_{\mathrm{tol}} = 10^{-3}$, and the maximum number of iterations was set to 2×10^3. All the experiments were carried out on a computer with an Intel(R) Core(TM) i7 CPU, 2.8 GHz processor and 8GB of RAM.

A. SDPs with block-arrow pattern

We consider randomly generated SDPs with block-arrow patterns (see Figure 2), which is used as a benchmark case in [17]. Such sparsity pattern is chordal, and its parameters are the number of blocks, l; block size, d; the size of the

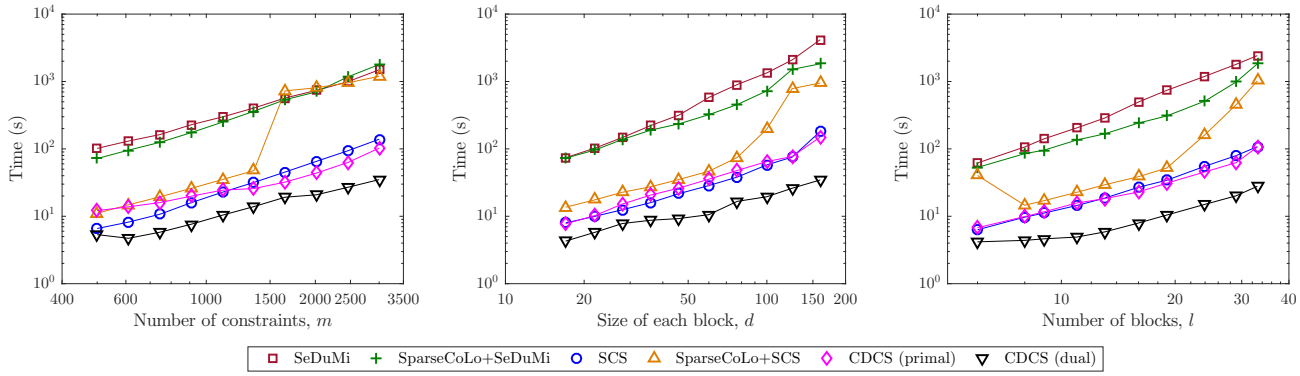

Fig. 3. CPU time for SDPs with block-arrow patterns. Left to right: varying number of constraints; varying number of blocks; varying block size.

TABLE I
PROBLEM STATISTICS FOR SDPLIB PROBLEMS

	maxG32	maxG51	thetaG51	qpG51
Affine constraints, m	2000	1000	6910	1000
Original cone size, n	2000	1000	1001	2000
Number of cliques, p	1499	674	674	1675
Maximum clique size	60	322	323	304
Minimum clique size	5	6	7	1

arrow head, h; and the number of constraints, m. Here, we consider the following cases:

1) Fix $l = 40$, $d = 10$, $h = 20$, vary m;
2) Fix $m = 1000$, $d = 10$, $h = 20$, vary l;
3) Fix $l = 40$, $h = 10$, $m = 1000$, vary d.

The CPU times for different combinations of solvers, averaged over five random problem instances, are shown in Figure 3. CDCS is approximately 10 times faster than SeDuMi and the combination SparseCoLO+SeDuMi, and our implementation of Algorithm 2 is the fastest of all methods we tested. Besides, the optimal value returned by CDCS was always within 0.02% of the high-accuracy value returned by SeDuMi.

B. Sparse SDPs from SDPLIB

Our second experiment is based on a set of sparse SDPs from SDPLIB [20]. We consider two max-cut problems (maxG32 and maxG51), a Lovász theta problem (thetaG51), and a box-constrained quadratic problem (qpG51), all of which are large, sparse SDPs. Table I reports the dimensions and some chordal decomposition details of these problems, while Figure 4 illustrates their aggregate sparsity patterns.

Table II summarizes our numerical results; problems maxG51, thetaG51 and qpG51 could not be solved using SeDuMi after being decomposed by SparseCoLO due to memory overflow caused by the large number of consensus constraints in the decomposed problem. For all four problems, CDCS (both primal and dual) is faster than SeDuMi, and can give speedups compared to either SCS and SparseCoLO+SCS in terms of CPU time and iterations.

It should be kept in mind that the most time-consuming step in CDCS is the projection onto a large number of semidefinite cones, which is currently implemented sequentially; CPU times are likely to reduce with a parallel implementation. Also note that although Algorithms 1 and 2 are scaled versions of each other and have the same iteration cost, they behave slightly differently; this could be expected, since ADMM algorithms are sensitive to data scaling.

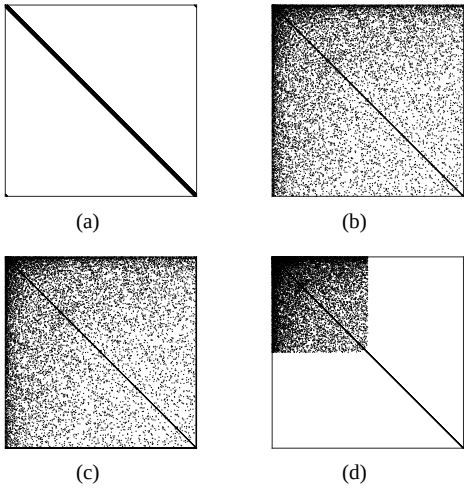

Fig. 4. Aggregate sparsity pattern of the SDPLIB problems considered: (a) maxG32 (b) maxG51 (c) thetaG51, (d) qpG51.

Finally, we remark that the stopping objective value from CDCS is within 2% of the optimal value returned by SeDuMi (which is highly accurate, and can be considered exact) in all four cases, and within 0.08% for the max-cut problems maxG32 and maxG51 — a negligible difference in applications. Of course, the accuracy can be improved by setting tighter stopping tolerances.

VI. CONCLUSION

In this paper, we proposed a conversion framework for large SDPs characterized by chordal sparsity suitable for the application of first-order methods, analogous to the conversion techniques for interior-point methods of [12], [13]. We also developed efficient ADMM algorithms for sparse SDPs in either primal or dual standard form, which are implemented in the conic solver CDCS. Our numerical simulations on random SDPs with block-arrow sparsity patterns and on large sparse problems in SDPLIB [20] show that our methods can provide speedups compared to interior-point solvers

TABLE II

RESULTS FOR THE PROBLEM INSTANCES IN SDPLIB

		SeDuMi	SparseCoLO+ SeDuMi	SCS	SparseCoLO+ SCS	CDCS (primal)	CDCS (dual)
maxG32	Total time (s)	974.6	355.2	2.553×10^3	65.1	88.6	53.1
	Pre-processing time (s)	0	3.18	0.43	3.24	21.2	21.4
	Objective value	1.568×10^3	1.568×10^3	1.568×10^3	1.566×10^3	1.569×10^3	1.568×10^3
	Iterations	14	15	2000	960	238	127
maxG51	Total time (s)	134.5	–	87.9	1.201×10^3	110.9	75.9
	Pre-processing time (s)	0	–	0.11	2.87	3.30	3.20
	Objective value	4.006×10^3	–	4.006×10^3	3.977×10^3	4.005×10^3	4.006×10^3
	Iterations	16	–	540	2000	235	157
thetaG51	Total time (s)	2.218×10^3	–	424.2	1.346×10^3	471.2	735.1
	Pre-processing time (s)	0	–	0.30	5.30	25.1	25.0
	Objective value	349	–	350.6	341.3	354.5	355.9
	Iterations	20	–	2000	2000	394	646
qpG51	Total time (s)	1.407×10^3	–	2.330×10^3	985.8	727.1	606.2
	Pre-processing time (s)	0	–	0.47	190.2	12.3	12.3
	Objective value	1.182×10^3	–	1.288×10^3	1.174×10^3	1.195×10^3	1.194×10^3
	Iterations	22	–	2000	2000	1287	1048

such as SeDuMi [25] — even when the chordal sparsity is exploited using SparseCoLO [14] — and also compared to the state-of-the-art first-order solver SCS [8]. Looking ahead, it would be desirable to exploit chordal sparsity in a first-order self-dual embedding formulation similar to that of [7], because a self-dual embedding can detect infeasibility. The development of improved software packages that take full advantage of parallel/distributed computer architectures is also of interest.

REFERENCES

[1] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics, 1994.

[2] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[3] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[4] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, "Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 746–768, 1998.

[5] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization*, vol. 6, no. 2, pp. 342–361, 1996.

[6] Z. Wen, D. Goldfarb, and W. Yin, "Alternating direction augmented lagrangian methods for semidefinite programming," *Mathematical Programming Computation*, vol. 2, no. 3-4, pp. 203–230, 2010.

[7] B. O'Donoghue, E. Chu, and S. Parikh, Nealand Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, 2016.

[8] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "SCS: Splitting conic solver, version 1.2.6," https://github.com/cvxgrp/scs, Apr. 2016.

[9] M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe, "Interior-point methods for large-scale cone programming," *Optimization for machine learning*, pp. 55–83, 2011.

[10] R. Grone, C. R. Johnson, E. M. Sá, and H. Wolkowicz, "Positive definite completions of partial hermitian matrices," *Linear algebra and its applications*, vol. 58, pp. 109–124, 1984.

[11] J. Agler, W. Helton, S. McCullough, and L. Rodman, "Positive semidefinite matrices with a given sparsity pattern," *Linear algebra and its applications*, vol. 107, pp. 101–149, 1988.

[12] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, "Exploiting sparsity in semidefinite programming via matrix completion i: General framework," *SIAM Journal on Optimization*, vol. 11, no. 3, pp. 647–674, 2001.

[13] S. Kim, M. Kojima, M. Mevissen, and M. Yamashita, "Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion," *Mathematical programming*, vol. 129, no. 1, pp. 33–68, 2011.

[14] K. Fujisawa, S. Kim, M. Kojima, Y. Okamoto, and M. Yamashita, "Users manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems," Research Report B-453, Dept. of Math. and Comp. Sci., Tokyo Institute of Technology, Tokyo 152-8552, Japan, Tech. Rep., 2009.

[15] S. Burer, "Semidefinite programming in the space of partial positive semidefinite matrices," *SIAM Journal on Optimization*, vol. 14, no. 1, pp. 139–172, 2003.

[16] M. S. Andersen, J. Dahl, and L. Vandenberghe, "Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones," *Mathematical Programming Computation*, vol. 2, no. 3-4, pp. 167–201, 2010.

[17] Y. Sun, M. S. Andersen, and L. Vandenberghe, "Decomposition in conic optimization with partially separable structure," *SIAM Journal on Optimization*, vol. 24, no. 2, pp. 873–897, 2014.

[18] A. Kalbat and J. Lavaei, "A fast distributed algorithm for decomposable semidefinite programs," in *2015 54th IEEE Conference on Decision and Control (CDC)*, Dec 2015, pp. 1742–1749.

[19] R. Madani, A. Kalbat, and J. Lavaei, "ADMM for sparse semidefinite programming with applications to optimal power flow problem," in *54th IEEE Conference on Decision and Control (CDC)*, Dec 2015, pp. 5932–5939.

[20] B. Borchers, "SDPLIB 1.2, a library of semidefinite programming test problems," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 683–690, 1999.

[21] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, pp. 77–79, 1981.

[22] L. Vandenberghe and M. S. Andersen, "Chordal graphs and semidefinite optimization," *Foundations and Trends® in Optimization*, vol. 1, no. 4, pp. 241–433, 2014.

[23] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[24] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima, and K. Murota, "Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results," *Mathematical Programming*, vol. 95, no. 2, pp. 303–327, 2003.

[25] J. F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization methods and software*, vol. 11, no. 1-4, pp. 625–653, 1999.