A Fast Distributed Algorithm for Decomposable Semidefinite Programs

Abdulrahman Kalbat and Javad Lavaei
Department of Electrical Engineering, Columbia University

Abstract—This paper aims to develop a fast, parallelizable algorithm for an arbitrary decomposable semidefinite program (SDP). To formulate a decomposable SDP, we consider a multi-agent canonical form represented by a graph, where each agent (node) is in charge of computing its corresponding positive semidefinite matrix subject to local equality and inequality constraints as well as overlapping (consistency) constraints with regards to the agent’s neighbors. Based on the alternating direction method of multipliers, we design a numerical algorithm, which has a guaranteed convergence under very mild assumptions. Each iteration of this algorithm has a simple closed-form solution, consisting of matrix multiplications and eigenvalue decompositions performed by individual agents as well as information exchanges between neighboring agents. The cheap iterations of the proposed algorithm enable solving real-world large-scale conic optimization problems.

I. INTRODUCTION

Alternating direction method of multipliers (ADMM) is a first-order optimization algorithm proposed in the mid-1970s by [1] and [2]. This method has attracted much attention recently since it can be used for large-scale optimization problems and also be implemented in parallel and distributed computational environments [3], [4]. Compared to second-order methods that are able to achieve a high accuracy via expensive iterations, ADMM relies on low-complex iterations and can achieve a modest accuracy in tens of iterations. Inspired by Nesterov’s scheme for accelerating gradient methods [5], great effort has been devoted to accelerating ADMM and attaining a high accuracy in a reasonable number of iterations [6]. Since ADMM’s performance is affected by the condition number of the problem’s data, diagonal rescaling is proposed in [7] for a class of problems to improve the performance and achieve a linear rate of convergence. The $O(\frac{1}{n})$ worst-case convergence rate of ADMM is proven in [8], [9] under the assumptions of closed convex sets and convex functions (not necessarily smooth). In [10], the $O(\frac{1}{\sqrt{n}})$ convergence rate is obtained for an asynchronous ADMM algorithm. The recent paper [11] represents ADMM as a dynamical system and then reduces the problem of proving the linear convergence of ADMM to verifying the stability of a dynamical system [11].

Semidefinite programs (SDP) are attractive due in part to three reasons. First, positive semidefinite constraints appear in many applications [12]. Second, SDPs can be used to study and approximate hard combinatorial optimization problems [13]. Third, this class of convex optimization problems includes linear, quadratic, quadratically-constrained quadratic, and second-order cone programs. It is known that small- to medium-sized SDP problems can be solved efficiently by interior point methods in polynomial time up to any arbitrary precision [14]. However, these methods are less practical for large-scale SDPs due to computation time and memory issues. However, it is possible to somewhat reduce the complexity by exploiting any possible structure in the problem such as sparsity.

The pressing need for solving real-world large-scale optimization problems calls for the development of efficient, scalable, and parallel algorithms. Because of the scalability of ADMM, the main objective of this work is to design a distributed ADMM-based parallel algorithm for solving an arbitrary sparse large-scale SDP with a guaranteed convergence, under very mild assumptions. We consider a canonical form of decomposable SDPs, which is characterized by a graph of agents (nodes) and edges. Each agent needs to find the optimal value of its associated positive semidefinite matrix subject to local equality and inequality constraints as well as overlapping constraints with its neighbors (more precisely, the matrices of two neighboring agents may be subject to consistency constraints). The objective function of the overall SDP is the summation of individual objectives of all agents. From the computation perspective, each agent is treated as a processing unit and each edge of the graph specifies what agents can communicate. We propose a distributed algorithm, whose iterations comprise local matrix multiplications and eigenvalue decompositions performed by individual agents as well as information exchanges between neighboring agents.

This paper is organized as follows. An overview of ADMM is provided in Section [I]. The distributed multi-agent SDP problem is formalized in Section [II]. An ADMM-based parallel algorithm is developed in Section [IV] by first studying the 2-agent case and then investigating the general multi-agent case. Simulation results on randomly-generated large-scale SDPs with a few million variables are provided in Section [V]. Finally, some concluding remarks are drawn in Section [VI].

Notations: $\mathbb{R}^n$ and $\mathbb{S}^n$ denote the sets of $n \times 1$ real vectors and $n \times n$ symmetric matrices, respectively. Lower case letters (e.g., $x$) represent vectors, and upper case letters (e.g., $W$) represent matrices. $tr\{W\}$ denotes the trace of a matrix $W$ and the notation $W \succeq 0$ means that $W$ is symmetric and positive semidefinite. Given a matrix $W$, its $(l,m)$ entry is denoted as $W(l,m)$. The symbols $(\cdot)^T$, $\|\cdot\|_2$ and $\|\cdot\|_F$ denote the transpose, $\ell_2$-norm (for vectors) and Frobenius norm (for matrices) operators, respectively. The ordering operator $(a, b) \preceq$ returns $(a, b)$ if $a < b$ and returns $(b, a)$ if $a > b$. The notation $|X|$ represents the cardinality (or size) of the set $X$. The finite
sequence of variables \(x_1, \ldots, x_n\) is denoted by \(\{x_i\}_{i=1}^{n}\). For an \(m \times n\) matrix \(W\), the notation \(W(X, Y)\) denotes the submatrix of \(W\) whose rows and columns are chosen form \(X\) and \(Y\), respectively, for given index sets \(X \subseteq \{1, \ldots, m\}\) and \(Y \subseteq \{1, \ldots, n\}\).

The notation \(G = (V, E)\) defines a graph \(G\) with the vertex (or node) set \(V\) and the edge set \(E\). The set of neighbors of vertex \(i \in V\) is denoted as \(N(i)\). To orient the edges of \(G\), we define a new edge set \(\mathcal{E}^{+} = \{(i, j) \mid (i, j) \in E \text{ and } i < j\}\).

II. ALTERNATING DIRECTION METHOD OF MULTIPLIERS

Consider the optimization problem

\[
\min_{x \in \mathbb{R}^n, y \in \mathbb{R}^m} f(x) + g(y) \quad (1a)
\]

subject to \(Ax + By = c\) \(\quad (1b)\)

where \(f(x)\) and \(g(y)\) are convex functions, \(A, B\) are known matrices, and \(c\) is a given vector of appropriate dimension. The above optimization problem has a separable objective function and linear constraints. Before proceeding with the paper, three numerical methods for solving this problem will be reviewed.

The first method is dual decomposition, which uses the Lagrangian function

\[
\mathcal{L}(x, y, \lambda) = f(x) + g(y) + \lambda^T (Ax + By - c) = \begin{cases} f(x) + \lambda^T Ax + g(y) + \lambda^T By - \lambda^T c & \text{if } h_1(x, \lambda) + h_2(y, \lambda) \end{cases} \quad (2)
\]

where \(\lambda\) is the Lagrange multiplier corresponding to the constraint \((1b)\). The above Lagrangian function can be separated into two functions \(h_1(x, \lambda)\) and \(h_2(y, \lambda)\). Inspired by this separation, the dual decomposition method is based on updating \(x, y\) and \(\lambda\) separately. This leads to the iterations

\[
\begin{align*}
    x^{t+1} & := \arg\min_x h_1(x, \lambda^t) \quad (3a) \\
    y^{t+1} & := \arg\min_y h_2(y, \lambda^t) \quad (3b) \\
    \lambda^{t+1} & := \lambda^t + \alpha^t (Ax^{t+1} + By^{t+1} - c) \quad (3c)
\end{align*}
\]

for \(t = 0, 1, 2, \ldots\), with an arbitrary initialization \((x^0, y^0, \lambda^0)\), where \(\alpha^t\) is a step size. Note that “argmin” denotes any minimizer of the corresponding function.

Despite its decomposability, the dual decomposition method has robustness and convergence issues. The method of multipliers could be used to remedy these difficulties, which is based on the augmented lagrangian function

\[
\mathcal{L}_\mu(x, y, \lambda) = f(x) + g(y) + \lambda^T (Ax + By - c) + \frac{\mu}{2} \|Ax + By - c\|^2 \quad (4)
\]

where \(\mu\) is a nonnegative constant. Notice that \((4)\) is obtained by augmenting the Lagrangian function in \((2)\) with a quadratic term in order to increase the smallest eigenvalue of the Hessian of the Lagrangian with respect to \((x, y)\). However, this augmentation creates a coupling between \(x\) and \(y\). The iterations corresponding to the method of multipliers are

\[
\begin{align*}
    (x^{t+1}, y^{t+1}) & := \arg\min_{(x, y)} \mathcal{L}_\mu(x, y, \lambda^t) \quad (5a) \\
    \lambda^{t+1} & := \lambda^t + \mu (Ax^{t+1} + By^{t+1} - c) \quad (5b)
\end{align*}
\]

ADMM offers a distributed computation property, a high degree of robustness, and a guaranteed convergence under very mild assumptions. In the reminder of this paper, we will use this first-order method to solve large-scale decomposable SDP problems.

III. PROBLEM FORMULATION

Consider a simple, connected, and undirected graph \(G = (V, E)\) with the node set \(V := \{1, \ldots, n\}\) and the edge set \(E \subseteq V \times V\), as shown in Figure 1. In a physical context, each node could represent an agent (or a machine or a processor or a thread) and each edge represents a communication link between the agents. In the context of this paper, each agent is in charge of computing a positive semidefinite matrix variable \(W_i\), and each edge \((i, j) \in E\) specifies an overlap between the matrix variables \(W_i\) and \(W_j\) of agents \(i\) and \(j\). More precisely, each edge \((i, j)\) is accompanied by two arbitrary integer-valued index sets \(I_{ij}\) and \(I_{ji}\) to capture the overlap between \(W_i\) and \(W_j\) through the equation \(W_i(I_{ij}, I_{ji}) = W_j(I_{ij}, I_{ji})\). Figure 2 illustrates this specification through an example with three overlapping matrices, where every two neighboring submatrices with an identical color must take the same value at optimality. Another way of thinking about this setting is that Figure 1 represents the sparsity graph of an arbitrary sparse large-scale SDP with a single global matrix variable \(W\), which is then reformulated in terms of certain matrices of \(W\), named \(W_1, \ldots, W_n\), using the Chordal extension and matrix completion theorems [15]. The objective of this paper is to solve the decomposable SDP problem (interchangeably referred to as distributed multi-agent SDP) given below.
The formulation in (7) has three main ingredients: (7), by allowing each agent $i$ to collaborate with its neighbors $N(i)$ to find an optimal value for its positive semidefinite submatrix $W_i$ while meeting its own constraints as well as all overlapping constraints. This is accomplished by local computations performed by individual agents and local communication between neighboring agents for information exchange.

There are two scenarios in which (7) could be used. In the first scenario, it is assumed that the SDP problem of interest is associated with a multi-agent system and matches the formulation in (7) exactly. In the second scenario, we consider an arbitrary sparse SDP problem in the centralized standard form, i.e., an SDP with a single positive semidefinite matrix $W$, and then convert it into a distributed SDP with multiple but smaller positive semidefinite matrices $W_i$ to match the formulation in (7). (note that a dense SDP problem can be put in the form of (7) with $n=1$). The conversion from a standard SDP to a distributed SDP is possible using the idea of chordal decomposition of positive semidefinite cones in [16], which exploits the fact that a matrix $W$ has a positive semidefinite completion if and only if certain submatrices of $W$, denoted as $W_1,...,W_n$, are positive semidefinite [17].

In this work, we propose an iterative algorithm for solving the decomposable SDP problem (7) using the first-order ADMM method. We show that each iteration of this algorithm has a simple closed-form solution, which consists of matrix multiplication and eigenvalue decomposition over matrices of size $n_i$ for agent $i \in \mathcal{V}$. Our work improves upon some recent papers in this area. [18] is a special case of our work with $n=1$, which does not offer any parallelizable algorithm for sparse SDPs and may not be applicable to large-scale sparse SDP problems. [16] uses the clique-tree conversion method to decompose sparse SDPs with chordal sparsity pattern into smaller sized SDPs, which can then be solved by interior point methods but this approach is limited by the large number of consistency constraints for the overlapping parts. Recently, [19] solves the decomposed SDP created by [16] using a first-order splitting method, but it requires solving a quadratic program at every iteration, which again imposes some limitations on the scalability of the proposed algorithm. In contrast, the algorithm to be proposed here is parallelizable with low computations at every iteration, without requiring any initial feasible point unlike interior point methods.

IV. DISTRIBUTED ALGORITHM FOR DECOMPOSABLE SEMIDEFINITE PROGRAMS

In this section, we design an ADMM-based algorithm to solve (7). For the convenience of the reader, we first consider the case where there are only two overlapping matrices $W_1$ and $W_2$. Later on, we derive the iterations for the general case with an arbitrary graph $G$.

A. Two-Agent Case

Assume that there are two overlapping matrices $W_1$ and $W_2$ embedded in a global SDP matrix variable $W$ as shown in Figure 3, where "w" submatrices of $W$ are redundant (meaning...
that there is no explicit constraint on the entries of these parts. The SDP problem for this case can be put in the canonical form \[7\], by setting \( \mathcal{V} = \{1, 2\} \), \( \mathcal{E}^+ = \{(1, 2)\} \) and \( |\mathcal{V}| = 2 \):

\[
\min_{W_1 \in \mathbb{S}^{n_1}} \quad \frac{1}{2} \sum_{i=1}^n c_i^T \sigma_i \quad \text{subject to:}
\]

\[
W_1 W_2 \succeq 0
\]

\[
W_1 (I_{12}, I_{12}) = W_2 (I_{12}, I_{21})
\]

where the matrix variables \( W_1 \in \mathbb{S}^{n_1} \) and \( W_2 \in \mathbb{S}^{n_2} \) correspond to agent 1 and agent 2, respectively. The dual of the 2-agent SDP problem \[8\] can be expressed as

\[
\min \left\{ c_j^T z_j + d_1^T v_1 + d_2^T v_2 \right\}
\]

subject to:

\[
- \sum_{j=1}^{p_1} z_j^1 B_j^1 - \sum_{i=1}^{q_1} v_i^1 D_i^1 + R_1 - \begin{bmatrix} 0 & 0 \\ 0 & H_{1,2} \end{bmatrix} = A_1
\]

\[
- \sum_{j=1}^{p_2} z_j^2 B_j^2 - \sum_{i=1}^{q_2} v_i^2 D_i^2 + R_2 + \begin{bmatrix} H_{2,1} & 0 \\ 0 & 0 \end{bmatrix} = A_2
\]

\[
H_{1,2} = H_{2,1}
\]

\[
v_1, v_2 \geq 0
\]

\[
R_1, R_2 \geq 0
\]

with the variables \( z_1, z_2, v_1, v_2, R_1, R_2, H_{1,2}, H_{2,1} \), where \( z_1 \in \mathbb{R}^{p_1} \), \( z_2 \in \mathbb{R}^{p_2} \), \( v_1 \in \mathbb{R}^{q_1} \) and \( v_2 \in \mathbb{R}^{q_2} \) are the Lagrange multipliers corresponding to the equality and inequality constraints in \[8\] and \[8c\], respectively, and the dual matrix variables \( R_1 \in \mathbb{S}^{n_1} \) and \( R_2 \in \mathbb{S}^{n_2} \) are the Lagrange multipliers corresponding to the constraints \[8d\]. The dual matrix variable \( H_{1,2} \) is the Lagrange multiplier corresponding to the submatrix \( W_1 (I_{12}, I_{12}) \) of \( W_1 \), whereas \( H_{2,1} \) is the Lagrange multiplier corresponding to the submatrix \( W_2 (I_{12}, I_{21}) \) of \( W_2 \). Since the overlapping entries between \( W_1 \) and \( W_2 \) are equal, as reflected in constraint \[8g\], the corresponding Lagrange multipliers should be equal as well, leading to constraint \[9a\].

If we apply ADMM to \[9\], it becomes impossible to split the variables into two blocks of variables associated with agents 1 and 2. The reason is that the augmented Lagrangian function of \[9\] creates a coupling between \( H_{1,2} \) and \( H_{2,1} \), which then requires updating \( H_{1,2} \) and \( H_{2,1} \) jointly. This issue can be resolved by introducing a new auxiliary variable \( H^{(1,2)} \) in order to decompose the constraint \( H_{1,2} = H_{2,1} \) into two constraints \( H_{1,2} = H^{(1,2)} \) and \( H_{2,1} = H^{(1,2)} \). Similarly, to make the update of \( v_1 \) and \( v_2 \) easier, we do not impose positivity constraints directly on \( v_1 \) and \( v_2 \) as in \[9c\]. Instead, we impose the positivity on two new vectors \( u_1, u_2 \) \( \geq 0 \) and then add the additional constraints \( v_1 = u_1 \) and \( v_2 = u_2 \). By applying the previous modifications, \[9\] could be rewritten in the decomposable form

\[
\min \sum_{i=1}^2 \left( c_i^T z_i + d_i^T v_i + I_+ (R_i) + I_+ (v_i) \right)
\]

subject to:

\[
- \sum_{j=1}^{p_1} z_j^1 B_j^1 - \sum_{i=1}^{q_1} v_i^1 D_i^1 + R_1 - \begin{bmatrix} 0 & 0 \\ 0 & H_{1,2} \end{bmatrix} = A_1
\]

\[
- \sum_{j=1}^{p_2} z_j^2 B_j^2 - \sum_{i=1}^{q_2} v_i^2 D_i^2 + R_2 + \begin{bmatrix} H_{2,1} & 0 \\ 0 & 0 \end{bmatrix} = A_2
\]

\[
H_{1,2} = H^{(1,2)}
\]

\[
v_1 = u_1
\]

\[
v_2 = u_2
\]

with the variables \( z_1, z_2, v_1, u_1, v_2, u_2, R_1, R_2, H_{1,2}, H_{2,1}, H^{(1,2)} \), where \( I_+ (R_i) \) is equal to 0 if \( R_i \geq 0 \) and is \(+\infty\) otherwise, and \( I_+ (v_i) \) is equal to 0 if \( v_i \geq 0 \) and is \(+\infty\) otherwise. To streamline the presentation, define

\[
B_i^{\text{sum}} = \sum_{j=1}^{p_i} z_j^i B_j^i, \quad D_i^{\text{sum}} = \sum_{i=1}^{q_i} v_i^i D_i^i, \quad i = 1, 2
\]

and

\[
H_{1,2}^{\text{full}} = \begin{bmatrix} 0 & 0 \\ 0 & H_{1,2} \end{bmatrix}, \quad H_{2,1}^{\text{full}} = \begin{bmatrix} -H_{2,1} & 0 \\ 0 & 0 \end{bmatrix}
\]

Note that \( B_i^{\text{sum}}, D_i^{\text{sum}}, H_{1,2}^{\text{full}} \) and \( H_{2,1}^{\text{full}} \) are functions of the variables \( z_i, v_i, H_{1,2} \) and \( H_{2,1} \), respectively, but the arguments are dropped for notational simplicity. The augmented Lagrangian function for \[10\] can be obtained as

\[
L_{\mu} (F, \mathcal{M}) = \sum_{i=1}^2 \left( c_i^T z_i + d_i^T v_i + I_+ (R_i) + I_+ (v_i) \right)
\]

\[
+ \frac{\mu}{2} \left\| B_i^{\text{sum}} - D_i^{\text{sum}} + R_1 - H_{1,2}^{\text{full}} - A_1 + \frac{G_1}{\mu} \right\|_F^2
\]

\[
+ \frac{\mu}{2} \left\| -B_i^{\text{sum}} - D_i^{\text{sum}} + R_2 - H_{2,1}^{\text{full}} - A_2 + \frac{G_2}{\mu} \right\|_F^2
\]

\[
+ \frac{\mu}{2} \left\| H_{1,2} - H^{(1,2)} + \frac{G_{1,2}}{\mu} \right\|_F^2
\]

\[
+ \frac{\mu}{2} \left\| H_{2,1} - H^{(1,2)} + \frac{G_{2,1}}{\mu} \right\|_F^2
\]

\[
+ \frac{\mu}{2} \left\| v_1 - u_1 + \frac{\lambda_1}{\mu} \right\|_2^2 + \frac{\mu}{2} \left\| v_2 - u_2 + \frac{\lambda_2}{\mu} \right\|_2^2
\]

where \( F = (z_1, z_2, v_1, u_1, v_2, u_2, R_1, R_2, H_{1,2}, H_{2,1}, H^{(1,2)}) \) is the set of optimization variables and \( \mathcal{M} = (R_1, R_2, H_{1,2}, H_{2,1}, H^{(1,2)}) \) is the set of Lagrange multipliers.
In order to proceed, we need to split the set of optimization variables $\mathcal{X}$ into two blocks of variables. To this end, define $\mathcal{X} = \{u_1, u_2, R_1, R_2, H(1,2)\}$ and $\mathcal{Y} = \{z_1, z_2, v_1, v_2, H_{1,2}, H_{2,1}\}$. Using the method delineated in Section 1, the two-block ADMM iterations can be obtained as:

$$G_1^{t+1} = G_1^t + \mu \left( -B_1^{\text{sum}} D_1^{t+1} - D_1^{t+1} R_1^{t+1} - H_{1,2}^{t+1} - A_1 \right)$$  \hspace{1cm} (15c)$$
$$G_2^{t+1} = G_2^t + \mu \left( -B_2^{\text{sum}} D_2^{t+1} + D_2^{t+1} R_2^{t+1} - H_{2,1}^{t+1} - A_2 \right)$$  \hspace{1cm} (15d)$$
$$G_{1,2}^{t+1} = G_{1,2}^t + \mu \left( H_{1,2}^{t+1} - H(1,2) \right)$$  \hspace{1cm} (15e)$$
$$G_{2,1}^{t+1} = G_{2,1}^t + \mu \left( H_{2,1}^{t+1} - H(1,2) \right)$$  \hspace{1cm} (15f)$$
$$\lambda_1^{t+1} = \lambda_1^t + \mu (v_1^{t+1} - u_1^{t+1})$$  \hspace{1cm} (15g)$$
$$\lambda_2^{t+1} = \lambda_2^t + \mu (v_2^{t+1} - u_2^{t+1})$$  \hspace{1cm} (15h)

for $t = 0, 1, 2, ...$. The above updates are derived based on the fact that ADMM aims to find a saddle point of the augmented lagrangian function by alternatively performing one pass of Gauss Seidel over $\mathcal{X}$ and $\mathcal{Y}$ and then updating the Lagrange multipliers $\mathcal{M}$ through Gradient ascent.

It is straightforward to show that the optimization over $\mathcal{X}$ in Block 1 is fully decomposable and amounts to 5 separate optimization subproblems with respect to the individual variables $u_1, u_2, R_1, R_2, H(1,2)$. In addition, the optimization over $\mathcal{Y}$ in Block 2 is equivalent to 2 separate optimization subproblems with the variables $(z_1, v_1, H_{1,2})$ and $(z_2, v_2, H_{2,1})$, respectively. Interestingly, all these subproblems have closed-form solutions. The corresponding iterations that need to be taken by agents 1 and 2 are provided in (16) and (17) (given in the next page). Note that these agents need to perform local computation in every iteration according to (16) and (17) and then exchange the updated values of the pairs $(H_{1,2}, G_{1,2})$ and $(H_{2,1}, G_{2,1})$ with one another.

To elaborate on (16) and (17), the positive semidefinite matrices $R_1$ and $R_2$ are updated through the operator $(\cdot)_+$, where $X_+$ is defined as the projection of an arbitrary symmetric matrix $X$ onto the set of positive semidefinite matrices by replacing its negative eigenvalues with 0 in the eigenvalue decomposition [13]. The positive vectors $u_1$ and $u_2$ are also updated through the operator $(x)_+$, which replaces any negative entry in an arbitrary vector $x$ with 0 while keeping the nonnegative entries. Using the first-order optimality condition $\nabla_{H(1,2)} \mathcal{L}_\mu(\cdot) = 0$, one could easily find the closed-form solution for $H(1,2)$ as shown in (16a) and (16b). By combining the conditions $\nabla_{z_1} \mathcal{L}_\mu(\cdot) = 0$, $\nabla_{v_1} \mathcal{L}_\mu(\cdot) = 0$ and $\nabla_{H_{1,2}} \mathcal{L}_\mu(\cdot) = 0$, the updates of $(z_1, v_1, H_{1,2})$ and $(z_2, v_2, H_{2,1})$ reduce to a (not necessarily unique) linear map, denoted as Lin$(\cdot)$ in (16a) and (17b) (due to non-uniqueness, we may have multiple solutions, and any of them can be used in the updates). The Lagrange multipliers in $\mathcal{M}$ are updated through Gradient ascent, as specified in (16e)-(16g) for agent 1 and in (17e)-(17g) for agent 2.

### B. Multi-Agent Case

In this part, we will study the general distributed multi-agent SDP (7). The dual of this problem, after considering all modifications used to convert (9) to (10), can be expressed in the decomposable form:

$$\min\sum_{i \in \mathcal{V}} \left( c_i^T z_i + d_i^T v_i + I_+ (R_i) + I_+ (v_i) \right)$$  \hspace{1cm} (18a)

subject to:

$$-B_i^{\text{sum}} - D_i^{\text{sum}} + R_i - \sum_{k \in \epsilon(i)} H_{i,k}^{\text{full}} = A_i \quad \forall i \in \mathcal{V}$$  \hspace{1cm} (18b)$$
$$H_{i,j} = H(i,j) \quad \forall (i,j) \in \epsilon^+$$  \hspace{1cm} (18c)$$
$$v_i = u_i \quad \forall i \in \mathcal{V}$$  \hspace{1cm} (18e)$$

with the variables $z_i, v_i, u_i, R_i, H_{i,j}, H_{i,i}, H_{i,j}^*, H_{i,i}^*$ for every $i \in \mathcal{V}$ and $(i,j) \in \epsilon^+$, where $B_i^{\text{sum}} = \sum_{j=1}^n z_{i,j} B_{i,j}$, $D_i^{\text{sum}} = \sum_{k=1}^q r_{i,k} D_{i,k}$ and $H_i^{\text{sum}} = \sum_{k \in \epsilon(i)} H_{i,k}^{\text{full}}$. Note that $z_i \in \mathbb{R}^n$ and $v_i \in \mathbb{R}^q$ are the Lagrange multipliers corresponding to the equality and inequality constraints in (7b) and (7c), respectively, and that $R_i \in \mathbb{S}^n$ is the Lagrange multiplier corresponding to the constraint (7d). Each element $h_{i,k}^{\text{full}}(a,b)$ of $H_{i,k}^{\text{full}}$ is either zero or equal to the Lagrange multiplier corresponding to an overlapping element $W_i(a,b)$ between $W_i$ and $W_k$. For a better understanding of the difference between $H_{i,k}^{\text{full}}$, $H_{i,j}^*$ and $H_i^{\text{sum}}$, an example is given in Figure 1 for the case where agent 1 is overlapping with agents 2 and 3. The ADMM iterations for the general case can be derived similarly to the 2-agent case, which yields the local computation (19) for each agent $i \in \mathcal{V}$. Consider the parameters defined in (20) for every $i \in \mathcal{V}$, $(i,j) \in \epsilon^+$, and time $t \in \{1, 2, 3, ...\}$. Define $V^t$ as
Based on [21], we terminate the algorithm as soon as

\[ \text{convergence, we need a finite-time stopping rule.} \]

Since the proposed algorithm is iterative with an asymptotic convergence, we need a finite-time stopping rule. Based on [21], we terminate the algorithm as soon as max\{P_1, P_2, D_1, D_2, D_3, D_4, \text{Gap}\} becomes smaller than a pre-
Fig. 4: An illustration of the difference between $H_{i,j}^{\text{full}}$, $H_{i,j}^{\text{sum}}$ and $H_{i,j}^{\text{sum}}$. Agent 1 is overlapping with agents 2 and 3 at the entries specified by $I_{12}$ and $I_{23}$. The white squares in the left matrix $H_{i,j}^{\text{full}} + H_{i,j}^{\text{sum}}$ represent those entries with value 0, and the color squares carry Lagrange multipliers.

specified tolerance, where

$$
(P_1)_i = \frac{\|B_i^T W_i - c_i\|_2}{1 + \|c_i\|_2} + \max(\|D_i^T W_i - d_i, 0\|_2) \quad (22a)
$$

$$
(P_2)_{i,j} = \frac{\|W_i(I_{i,j}, I_{i,j}) - W_j(I_{i,j}, I_{i,j})\|_F}{1 + \|W_i(I_{i,j}, I_{i,j})\|_F + \|W_j(I_{i,j}, I_{i,j})\|_F} \quad (22b)
$$

$$
(D_1)_i = \frac{\|B_i^{\text{sum}} - B_i^{\text{full}} + R_i - H_i^{\text{sum}} - A_i\|_F}{1 + \|A_i\|_1} \quad (22c)
$$

$$
(D_2)_{i,j} = \frac{\|H_{i,j} - H^{(i,j)}\|_F}{1 + \|H_{i,j}\|_F + \|H^{(i,j)}\|_F} \quad (22d)
$$

$$
(D_3)_{i,j} = \frac{\|H_{i,j} - H^{(i,j)}\|_F}{1 + \|H_{i,j}\|_F + \|H^{(i,j)}\|_F} \quad (22e)
$$

$$
(D_4)_i = \frac{\|v_i - u_i\|_2}{1 + \|v_i\|_2 + \|u_i\|_2} \quad (22f)
$$

$$
\text{Gap} = \frac{\sum_{i \in V} (c_i^T z_i + d_i^T v_i - \text{tr}(A_i W_i))}{1 + \sum_{i \in V} (c_i^T z_i + d_i^T v_i) + |\sum_{i \in V} \text{tr}(A_i W_i)|} \quad (22g)
$$

for every $i \in V$ and $(i, j) \in E^+$, where

- the letters P and D refer to the primal and dual infeasibilities, respectively.
- $W_i$ is the vectorized version of $W_i$ obtained by stacking the columns of $W_i$ one under another to create a column vector.
- $B_i$ and $D_i$ are matrices whose columns are the vectorized versions of $B_j^{(i)}$ and $D_j^{(i)}$ for $j = 1, \ldots, p_i$ and $l = 1, \ldots, q_i$, respectively.

The stopping criteria in (22) are based on the primal and dual infeasibilities as well as the duality gap.

V. Simulations Results

The objective of this section is to elucidate the results of this work on randomly generated large-scale structured SDP problems. A prototype of the algorithm was implemented in MATLAB and all of the simulations below were run on a laptop with an Intel Core i7 quad-core 2.5 GHz CPU and 8 GB RAM.

For every $i \in V$, we generate a random instance of the problem as follows:

- Each matrix $A_i$ is chosen as $\Omega + \Omega^T + v_i I$, where the entries of $\Omega$ are uniformly chosen from the integer set $\{1, 2, 3, 4, 5\}$. This creates reasonably well-conditioned matrices $A_i$.
- Each matrix $B_j$ (or $D_j$) is chosen as $\Omega + \Omega^T$, where $\Omega$ is generated as before.
- Each matrix variable $W_i$ is assumed to be 40 by 40.
- The matrices $W_1, \ldots, W_n$ are assumed to overlap with each other in a banded structure, associated with a path graph $G$ with the edges $(1, 2), (2, 3), \ldots, (n - 1, n)$. One can regard $W_i$’s as submatrices of a full-scale matrix variable $W$ in the form of Figure 3 but with $n$ overlapping blocks, where 25% of the entries of every two neighboring matrices $W_i$ and $W_{i+1}$ (leading to a $10 \times 10$ submatrix) overlaps.

In order to demonstrate the proposed algorithm on large-scale SDPs, three different values will be considered for the total number of overlapping blocks (or agents): 1000, 2000 and 4000. To give the reader a sense of how large the simulated SDPs are, the total number of entries of $W_i$’s in the decomposed SDP problem ($N_{\text{Decomp}}$) and the total number of entries of $W$ in the corresponding full-SDP problem ($N_{\text{Full}}$) are listed below:

- 1000 agents: $N_{\text{Full}} = 0.9$ billion, $N_{\text{Decomp}} = 1.6$ million
- 2000 agents: $N_{\text{Full}} = 3.6$ billion, $N_{\text{Decomp}} = 3.2$ million
- 4000 agents: $N_{\text{Full}} = 14.4$ billion, $N_{\text{Decomp}} = 6.4$ million

The simulation results are provided in Table 1 with the following entries: $P_{\text{obj}}$ and $D_{\text{obj}}$ are the primal and dual objective values, “iter” denotes the number of iterations needed to achieve a desired tolerance, $t_{\text{CPU}}$ and $t_{\text{iter}}$ are the total CPU time (in seconds) and the time per iteration (in seconds per iteration), and “Optimality” (in percentage) is calculated as:

$$\text{Optimality Degree (\%)} = 100 - \frac{P_{\text{obj}} - D_{\text{obj}}}{P_{\text{obj}}} \times 100$$

As shown in Table 1, the simulations were run for three cases:

- $p_i = 5$ and $q_i = 0$: each agent has 5 equality constraints and no inequality constraints.
- $p_i = 0$ and $q_i = 5$: each agent has no equality constraints and 5 inequality constraints.
- $p_i = 5$ and $q_i = 5$: each agent has 5 equality constraints and 5 inequality constraints.

All solutions reported in Table 1 are based on the tolerance of $10^{-3}$ and an optimality degree of at least 99.9%. The aggregate residue $V^+$ is plotted in Figure 5 for the 4000-agent case with $p_i = q_i = 5$, which is a monotonically decreasing function. Note that the time per iteration is between 1.66 and 18.03 in a MATLAB implementation, which can be reduced significantly in C++. Efficient and computationally cheap preconditioning methods could dramatically reduce the number of iterations, but this is outside the scope of this paper.

VI. Conclusion

This paper develops a fast, parallelizable algorithm for an arbitrary decomposable semidefinite program (SDP). To formulate a decomposable SDP, we consider a multi-agent canonical form represented by a graph, where each agent (node) is
in charge of computing its corresponding positive semidefinite matrix. The main goal of each agent is to ensure that its matrix is optimal with respect to some measure and satisfies local equality and inequality constraints. In addition, the matrices of two neighboring agents may be subject to overlapping constraints. The objective function of the optimization is the sum of all objectives of individual agents. The motivation behind this formulation is that an arbitrary sparse SDP problem can be converted to a decomposable SDP by means of the Chordal extension and matrix completion theorems. Using the alternating direction method of multipliers, we develop a distributed algorithm to solve the underlying SDP problem. At every iteration, each agent performs simple computations (matrix multiplication and eigenvalue decomposition) without having to solve any optimization subproblem, and then communicates some information to its neighbors. By deriving a Lyapunov-type non-increasing function, it is shown that the proposed algorithm converges as long as Slater’s conditions hold. Simulations results on large-scale SDP problems with a few million variables are offered to elucidate the efficacy of this work.

<table>
<thead>
<tr>
<th>Cases</th>
<th>1000</th>
<th>2000</th>
<th>4000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i = 5$ $q_i = 0$</td>
<td>$P_{O_{2q}}$</td>
<td>4.896e+5</td>
<td>9.863e+5</td>
</tr>
<tr>
<td></td>
<td>$D_{O_{2q}}$</td>
<td>4.896e+5</td>
<td>9.863e+5</td>
</tr>
<tr>
<td></td>
<td>iter</td>
<td>257</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>$t_{CPU}$ (sec)</td>
<td>437.56</td>
<td>853.75</td>
</tr>
<tr>
<td>$t_{iter}$ (sec per iter)</td>
<td>1.66</td>
<td>4.32</td>
<td>15.04</td>
</tr>
<tr>
<td>Optimality</td>
<td>99.98%</td>
<td>99.98%</td>
<td>99.98%</td>
</tr>
<tr>
<td>$p_i = 0$ $q_i = 5$</td>
<td>$P_{O_{2q}}$</td>
<td>8.236e+5</td>
<td>1.64756e+6</td>
</tr>
<tr>
<td></td>
<td>$D_{O_{2q}}$</td>
<td>8.236e+5</td>
<td>1.64756e+6</td>
</tr>
<tr>
<td></td>
<td>iter</td>
<td>1081</td>
<td>1127</td>
</tr>
<tr>
<td></td>
<td>$t_{CPU}$ (sec)</td>
<td>1828.19</td>
<td>4353.49</td>
</tr>
<tr>
<td>$t_{iter}$ (sec per iter)</td>
<td>1.66</td>
<td>4.32</td>
<td>15.04</td>
</tr>
<tr>
<td>Optimality</td>
<td>99.98%</td>
<td>99.99%</td>
<td>99.99%</td>
</tr>
<tr>
<td>$p_i = 5$ $q_i = 5$</td>
<td>$P_{O_{2q}}$</td>
<td>5.313e+5</td>
<td>1.0673e+6</td>
</tr>
<tr>
<td></td>
<td>$D_{O_{2q}}$</td>
<td>5.313e+5</td>
<td>1.0673e+6</td>
</tr>
<tr>
<td></td>
<td>iter</td>
<td>291</td>
<td>409</td>
</tr>
<tr>
<td></td>
<td>$t_{CPU}$ (sec)</td>
<td>547.62</td>
<td>1416.85</td>
</tr>
<tr>
<td>$t_{iter}$ (sec per iter)</td>
<td>1.66</td>
<td>4.32</td>
<td>18.03</td>
</tr>
<tr>
<td>Optimality</td>
<td>99.98%</td>
<td>99.99%</td>
<td>99.99%</td>
</tr>
</tbody>
</table>

### REFERENCES


