Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 53-2 (2020) 7355–7361

A clique graph based merging strategy for decomposable SDPs

Michael Garstka* **Mark Cannon*** **Paul Goulart***

* *Department of Engineering Science, University of Oxford, UK*
Email: {michael.garstka, mark.cannon, paul.goulart}@eng.ox.ac.uk

Abstract: Chordal decomposition techniques are used to reduce large structured positive semidefinite matrix constraints in semidefinite programs (SDPs). The resulting equivalent problem contains multiple smaller constraints on the nonzero blocks (or cliques) of the original problem matrices. This usually leads to a significant reduction in the overall solve time. A further reduction is possible by remerging cliques with significant overlap. The degree of overlap for which this is effective is dependent on the particular solution algorithm and hardware to be employed. We propose a novel clique merging approach that utilizes the clique graph to identify suitable merge candidates and that is suitable for any SDP solver algorithm. We show its performance in combination with a first-order method by comparing it with two existing approaches on selected problems from a benchmark library. Our approach is implemented in the latest version of the conic ADMM-solver `COSMO`.

Keywords: convex optimisation, semidefinite programming, chordal decomposition, clique merging

We consider the primal-form semidefinite program (SDP):

$$\begin{aligned} &\text{minimize} && \langle C, X\rangle \\ &\text{subject to} && \langle A_i, X\rangle = b_i, \quad i = 1,\dots,m \\ &&& X \in \mathbb{S}^n_+, \end{aligned} \tag{1}$$

with variable X and coefficient matrices $A_i, C \in \mathbb{S}^n$. The corresponding dual problem is

$$\begin{aligned} &\text{maximize} && b^\top y \\ &\text{subject to} && \sum_{i=1}^m A_i y_i + S = C \\ &&& S \in \mathbb{S}^n_+, \end{aligned} \tag{2}$$

with dual variable $y \in \mathbb{R}^m$ and slack variable S. Semidefinite programming is used to solve problems that appear in a variety of applications such as portfolio optimisation, robust control, and optimal power flow problems. Algorithms to solve SDPs, most notably interior point methods, have been known since the 1980s (Nesterov and Nemirovsky, 1988). However, the recent trend to use models based on large quantities of data leads to SDPs whose dimensions challenge established solver algorithms.

Two main approaches are commonly used to deal with this challenge. The first approach is to use first-order methods (FOMs) as in O'Donoghue et al. (2016) or in Zheng et al. (2020). FOMs typically trade-off moderate accuracy solutions for a lower per-iteration computational cost and can therefore handle large problems more easily.

The second approach is to exploit sparsity in the problem data. Fukuda et al. (2001) showed that if the coefficient matrices A_i, C exhibit an *aggregate sparsity structure* represented by a *chordal graph* $G(V, E)$, then the original primal and dual forms in (1) and (2) can be decomposed. These equivalent problems involve only positive semidefinite constraints on the nonzero blocks of the sparsity pattern, which can lead to a significant reduction in the dimension of each constraint, thereby reducing solve time. The equivalent primal problem is given by

$$\begin{aligned} &\text{minimize} && \langle C, X\rangle \\ &\text{subject to} && \langle A_i, X\rangle = b_i, \quad i = 1,\dots,m \\ &&& X_\ell = T_\ell X T_\ell^\top, \ \ell = 1,\dots,p \\ &&& X_\ell \in \mathbb{S}^{|\mathcal{C}_\ell|}_+, \quad \ell = 1,\dots,p, \end{aligned} \tag{3}$$

where the blocks X_ℓ are represented by subgraphs, called cliques, denoted $\mathcal{C}_\ell$. Additional constraints using entry-selector matrices T_ℓ, see (6), enforce equality of the overlapping entries in X. Following Fukuda et al. (2001) we refer to this conversion as the *domain-space decomposition*. The dual of this problem can be obtained by applying the *range-space decomposition*:

$$\begin{aligned} &\text{maximize} && b^\top y \\ &\text{subject to} && \sum_{i=1}^m A_i y_i + \sum_{\ell=1}^p T_\ell^\top S_\ell T_\ell = C \\ &&& S_\ell \in \mathbb{S}^{|\mathcal{C}_\ell|}_+, \quad \ell = 1,\dots,p. \end{aligned} \tag{4}$$

Notice that the number and dimension of the block variables X_ℓ and S_ℓ depend only on the choice of cliques in the graph. Starting from an initial decomposition we can merge two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ into a single clique with dimension $|\mathcal{C}_i \cup \mathcal{C}_j|$. Consequently, merging blocks has two opposing effects. It increases the size of the blocks while decreasing the number of equality constraints. Therefore, to evaluate the effect of a merge on the per-iteration time of a solver algorithm one has to take into account both the overlap between the cliques and the main linear algebra operations involved in each iteration.

⋆ M. Garstka is supported by the Clarendon Scholarship.

10.1016/j.ifacol.2020.12.1255

Related work

Heuristic methods to merge cliques have been proposed for interior-point methods. Nakata et al. (2003) suggest traversing the *clique tree*, a subset of all clique pairs with overlapping entries. For each edge in the tree they merge corresponding cliques if the number of common entries relative to the cardinality of the individual cliques is higher than some threshold value, chosen heuristically to balance the block sizes and the number of additional equality constraints. The methods are implemented in the `SparseCoLO` package (Fujisawa et al., 2009).

Similarly, Sun et al. (2014) suggest to traverse the clique tree and merge cliques if the amount of fill-in and the cardinality of the supernodes are below certain thresholds. This approach is implemented in the `CHOMPACK` package (Andersen and Vandenberghe, 2015).

Molzahn et al. (2013) discuss clique merging in the context of a solver designed for large optimal power flow problems. For each pair of adjacent cliques in the clique tree they determine how a merge would affect the problem dimension, i.e. the change in total number of variables and linking constraints. They then greedily merge the blocks with the biggest reduction until the number of cliques decreases by a predefined percentage.

A limitation of existing methods is that they rely on heuristic parameters designed for a specific interior point implementation. Furthermore, they consider only pairs of cliques that are adjacent in the clique tree. We show in Section 2 that two cliques with an advantageous merge are not necessarily adjacent in the clique tree.

With this paper we make the following contributions:

1. We propose a clique merging strategy based on the clique intersection graph which considers all possible pair-wise merges and which can be tailored to the platform-specific matrix factorisation time.
2. We use a weighting function for each pair of overlapping cliques that can be tailored to the specific algorithm used to solve the SDP. Specifically, we propose a weighting function for first-order methods that leverages the simpler relationship between clique sizes and per-iteration time, compared to interior-point methods. Consequently, we achieve consistently lower per-iteration times than with existing merging strategies.
3. We provide a customisable implementation of our method in the latest version of the conic solver package `COSMO` (Garstka et al., 2019).

Outline

In Section 1 we define graph related concepts and describe how a graph-represented sparsity pattern can be used to decompose the primal and dual form of an SDP. In Section 2 we briefly outline two existing merging strategies based on the construction of the clique tree and describe our clique graph based approach. In Section 3 we then preprocess a number of benchmark problems using the different strategies and solve them with the same first order solver. Consequently, we compare the impact of each strategy on the number of iterations, the per-iteration time, and the total solve time of the algorithm. Section 4 concludes the paper.

1. GRAPH PRELIMINARIES

In the following we define graph related concepts and how they relate to the sparsity structure of a matrix. A good overview on this topic is provided by Vandenberghe and Andersen (2015). We consider the *undirected graph* $G(V, E)$ with vertex set V and edge set $E \subseteq V \times V$. Two vertices v_1, v_2 are *adjacent* if $\{v_1, v_2\} \in E$. A *cycle* is a path of edges (i.e. a sequence of distinct edges) joining a sequence of vertices in which only the first and last vertices are repeated. A graph is called *complete* if all vertices are pairwise adjacent. We follow the convention of Vandenberghe and Andersen (2015) by defining a *clique* as a subset of vertices $\mathcal{C} \subseteq V$ that induces a *maximal* complete subgraph of G.

The decomposition theory described in Section 1b relies on a subset of graphs that exhibit the important property of *chordality*. A graph is *chordal* (or *triangulated*) if every cycle of length greater than three has a *chord*, which is an edge between nonconsecutive vertices of the cycle. A non-chordal graph can always be made chordal by adding extra edges. An undirected graph with n vertices can be used to represent the sparsity pattern of a symmetric matrix $S \in \mathbb{S}^n$. Every nonzero entry $S_{ij} \neq 0$ in the lower (or upper) triangular part of the matrix introduces an edge $(i, j) \in E$. An example of a sparsity pattern and the associated graph is shown in Fig. 1(a–b).

For a given sparsity pattern $G(V, E)$, we define the following symmetric sparse matrix cones:

$$\mathbb{S}^n(E, 0) \coloneqq \{S \in \mathbb{S}^n \mid S_{ij} = S_{ji} = 0, \text{ if } i \neq j, \, (i, j) \notin E\},$$
$$\mathbb{S}^n_+(E, 0) \coloneqq \{S \in \mathbb{S}^n(E, 0) \mid S \succeq 0\}.$$

This means that for a matrix $S \in \mathbb{S}^n(E, 0)$ the diagonal entries S_{ii} and the off-diagonal entries S_{ij} with $(i, j) \in E$ may be zero or nonzero. Moreover, we define the cone of positive semidefinite completable matrices:

$$\mathbb{S}^n_+(E, ?) \coloneqq \left\{Y \mid \exists \hat{Y} \in \mathbb{S}^n_+, Y_{ij} = \hat{Y}_{ij}, \text{if } i = j \text{ or } (i, j) \in E\right\}.$$

For a matrix $Y \in \mathbb{S}^n_+(E, ?)$ we can find a positive semidefinite completion by choosing appropriate values for all entries $(i, j) \notin E$. An algorithm to find this completion is described in Vandenberghe and Andersen (2015).

An important structure conveying substantial information about the nonzero blocks of a matrix, or equivalently the cliques of a chordal graph, is the *clique tree* (or *junction tree*). For a chordal graph G let $\mathcal{B} = \{\mathcal{C}_1, \ldots, \mathcal{C}_p\}$ be the set of cliques. The clique tree $\mathcal{T}(\mathcal{B}, \mathcal{E})$ is formed by taking the cliques as vertices and by choosing edges from $\mathcal{E} \subseteq \mathcal{B} \times \mathcal{B}$ such that the tree satisfies the *running-intersection property*:

Definition 1. (Running intersection property).
For each pair of cliques $\mathcal{C}_i$, $\mathcal{C}_j \in \mathcal{B}$, the intersection $\mathcal{C}_i \cap \mathcal{C}_j$ is contained in all the cliques on the path in the clique tree connecting $\mathcal{C}_i$ and $\mathcal{C}_j$.

This property is also referred to as *clique-intersection property* in Nakata et al. (2003) and *induced subtree property* in Vandenberghe and Andersen (2015). For a

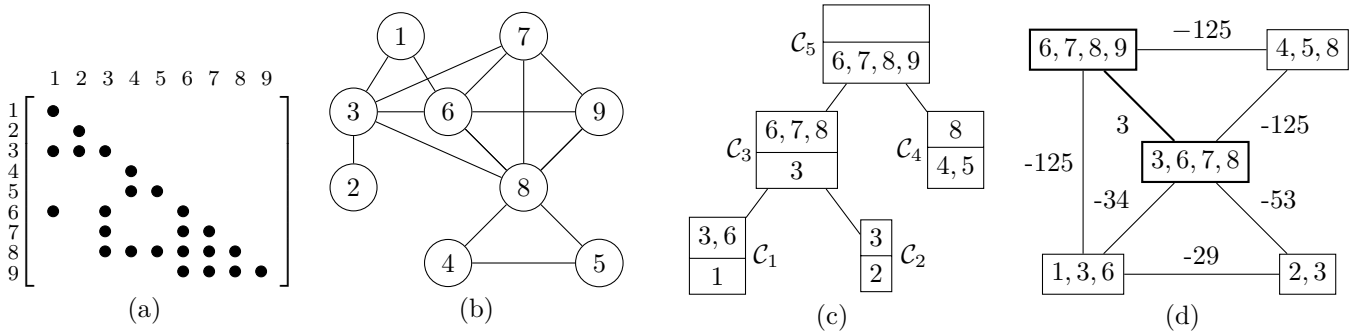

Fig. 1. (a) Aggregate sparsity pattern, (b) sparsity graph $G(V, E)$, (c) clique tree $\mathcal{T}(\mathcal{B}, \mathcal{E})$, and (d) clique graph $\mathcal{G}(\mathcal{B}, \xi)$ with edge weighting function $e(\mathcal{C}_i, \mathcal{C}_j) = |\mathcal{C}_i|^3 + |\mathcal{C}_j|^3 - |\mathcal{C}_i \cup \mathcal{C}_j|^3$.

given chordal graph, a clique tree can be computed using the algorithm described in Pothen and Sun (1990).

The clique tree for an example sparsity pattern is shown in Fig. 1(c). For a clique $\mathcal{C}_\ell$ we refer to the first clique encountered on the path to the root as its *parent clique* $\mathcal{C}_{\text{par}}$. Conversely $\mathcal{C}_\ell$ is called the *child* of $\mathcal{C}_{\text{par}}$. If two cliques have the same parent clique we refer to them as *siblings*. For each clique define the functions $\text{par}(\mathcal{C}_\ell)$ and $\text{ch}(\mathcal{C}_\ell)$ that return its parent clique and its set of child cliques. Note that each clique in Fig. 1(c) has been partitioned into two sets. The upper row represents the *separators* $\eta_\ell = \mathcal{C}_\ell \cap \text{par}(\mathcal{C}_\ell)$, i.e. all clique elements that are also contained in the parent clique. We call the sets of the remaining vertices shown in the lower rows the *clique residuals* or *supernodes* $\nu_\ell = \mathcal{C}_\ell \setminus \eta_\ell$. Keeping track of which vertices in a clique belong to the supernode and the separator is useful as the information is needed to perform a positive semidefinite completion. For a set of vertices V, the *power set* $\{W \mid W \subseteq V\}$ is denoted as 2^V.

1b. CHORDAL DECOMPOSITION

We next briefly describe how to apply chordal decomposition to an SDP. Let us assume that the problem matrices in (1) and (2) each have their own sparsity pattern

$$A_i \in \mathbb{S}^n(E_{A_i}, 0) \text{ and } C \in \mathbb{S}^n(E_C, 0).$$

The *aggregate sparsity* of the problem is given by the graph $G(V, E)$ with edge set

$$E = E_{A_1} \cup E_{A_2} \cup \cdots \cup E_{A_m} \cup E_C.$$

In general $G(V, E)$ will not be chordal, but a *chordal extension* can be found by adding edges to the graph. We denote the extended graph as $G(V, \bar{E})$. Finding the minimum number of edges to make the graph chordal is an NP-complete problem (Yannakakis, 1981). Consider a matrix M of ones corresponding to the edge set E. A commonly used heuristic method to find an extension is first to apply a reordering with approximate minimum fill-in (Amestoy et al., 1996). Afterwards, a symbolic Cholesky factorisation is applied to the reordered matrix. The Cholesky factor L then defines a chordal extension with edge set $\bar{E}$.

Given sparsity information of the problem we can modify the matrix constraints in (1) and (2) to the respective sparse positive semidefinite matrix spaces:

$$X \in \mathbb{S}^n_+(\bar{E}, ?) \text{ and } S \in \mathbb{S}^n_+(\bar{E}, 0). \tag{5}$$

We further define the entry-selector matrices $T_\ell \in \mathbb{R}^{|\mathcal{C}_\ell| \times n}$ for a clique $\mathcal{C}_\ell$:

$$(T_\ell)_{ij} \coloneqq \begin{cases} 1, & \text{if } \mathcal{C}_\ell(i) = j \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

where $\mathcal{C}_\ell(i)$ is the ith vertex of $\mathcal{C}_\ell$. We can express the constraints in (5) in terms of multiple smaller coupled constraints using the theorems by Grone et al. (1984) and Agler et al. (1988).

Theorem 2. (Grone's theorem). Let $G(V, \bar{E})$ be a chordal graph with a set of maximal cliques $\{\mathcal{C}_1, \ldots, \mathcal{C}_p\}$. Then $X \in \mathbb{S}^n_+(\bar{E}, ?)$ if and only if

$$X_\ell = T_\ell X T_\ell^\top \in \mathbb{S}^{|\mathcal{C}_\ell|}_+, \quad \forall \ell = 1, \ldots, p. \tag{7}$$

Applying this theorem to (1) while restricting X to the positive semidefinite completable matrix cone as in (5) yields the decomposed problem in (3). For the dual problem we utilise Agler's theorem, which is the dual to Thm. 2:

Theorem 3. (Agler's theorem). Let $G(V, \bar{E})$ be a chordal graph with a set of maximal cliques $\{\mathcal{C}_1, \ldots, \mathcal{C}_p\}$. Then $S \in \mathbb{S}^n_+(\bar{E}, 0)$ if and only if there exist matrices $S_\ell \in \mathbb{S}^{|\mathcal{C}_\ell|}_+$ for $\ell = 1, \ldots, p$ such that

$$S = \sum_{\ell=1}^{p} T_\ell^\top S_\ell T_\ell. \tag{8}$$

With this theorem, we transform the dual form SDP in (2) with the restriction on S in (5) to arrive at (4). Next, we show how to shape the sparsity pattern in the problem to reduce the per-iteration time of an SDP solver.

2. CLIQUE MERGING

Given an initial decomposition with edge set $\bar{E}$ and a set of cliques $\{\mathcal{C}_1, \ldots, \mathcal{C}_p\}$, we are free to merge any number of cliques back into larger blocks. This is equivalent to treating *structural* zeros in the problem as *numerical* zeros, which leads to additional edges in the graph. Looking at the decomposed problem in (3) and (4), the effects of merging two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ are twofold:

1. We replace two positive semidefinite matrix constraints of dimensions $|\mathcal{C}_i|$ and $|\mathcal{C}_j|$ with one constraint on a larger clique with dimension $|\mathcal{C}_i \cup \mathcal{C}_j|$, where the increase in dimension depends on the size of the overlap.
2. We remove consistency constraints for the overlapping entries between $\mathcal{C}_i$ and $\mathcal{C}_j$, thus reducing the size of the linear system of equality constraints.

When merging cliques these two factors have to be balanced. The correct balance depends foremost on the used solver algorithm. Nakata et al. (2003) and Sun et al. (2014) use the clique tree to search for favourable merge candidates. We will call these two approaches `SparseCoLO` and *parent-child* strategy in the following sections. Before describing these methods, we define a procedure in Alg. 1 that describes how to merge a set of cliques within the set $\mathcal{B}$ and update the edge set $\mathcal{E}$ accordingly.

Algorithm 1: Function mergeCliques($\mathcal{B}, \mathcal{E}, \mathcal{B}_m$).

Input : A set of cliques $\mathcal{B}$ with edge set $\mathcal{E}$, a subset of cliques $\mathcal{B}_m = \{\mathcal{C}_{m,1}, \mathcal{C}_{m,2}, \dots, \mathcal{C}_{m,r}\} \subseteq \mathcal{B}$ to be merged.

Output: A reduced set of cliques $\hat{\mathcal{B}}$ with edge set $\hat{\mathcal{E}}$ and the merged clique $\mathcal{C}_m$.

$\hat{\mathcal{E}} \leftarrow \mathcal{E}$;
$\mathcal{C}_m \leftarrow \mathcal{C}_{m,1} \cup \mathcal{C}_{m,2} \cup \cdots \cup \mathcal{C}_{m,r}$;
$\hat{\mathcal{B}} \leftarrow (\mathcal{B} \setminus \mathcal{B}_m) \cup \{\mathcal{C}_m\}$;
Remove edges $\{(\mathcal{C}_i, \mathcal{C}_j) \mid i \neq j,\ \mathcal{C}_i, \mathcal{C}_j \in \mathcal{B}_m\}$ in $\hat{\mathcal{E}}$;
Replace edges $\{(\mathcal{C}_i, \mathcal{C}_j) \mid \mathcal{C}_i \in \mathcal{B}_m, \mathcal{C}_j \notin \mathcal{B}_m\}$ with $(\mathcal{C}_m, \mathcal{C}_j)$ in $\hat{\mathcal{E}}$;

2.1 Existing clique tree-based strategies

The parent-child strategy described in Sun et al. (2014) traverses the clique tree in depth-first order and merges a clique $\mathcal{C}_\ell$ with its parent clique $\mathcal{C}_{\mathrm{par}(\ell)} \coloneqq \mathrm{par}(\mathcal{C}_\ell)$ if at least one of the two following conditions are met:

$$\left(\left|\mathcal{C}_{\mathrm{par}(\ell)}\right| - |\eta_\ell|\right)\left(|\mathcal{C}_\ell| - |\eta_\ell|\right) \leq t_{\mathrm{fill}}, \tag{9}$$

$$\max\left\{|\nu_\ell|, \left|\nu_{\mathrm{par}(\ell)}\right|\right\} \leq t_{\mathrm{size}}, \tag{10}$$

with heuristic parameters t_{fill} and t_{size}. The conditions keep the amount of extra fill-in and the supernode cardinalities below the specified thresholds. The `SparseCoLO` strategy described in Nakata et al. (2003) and Fujisawa et al. (2006) considers parent-child as well as sibling relationships. Given a parameter $\sigma > 0$, two cliques $\mathcal{C}_i, \mathcal{C}_j$ are merged if the following merge criterion holds

$$\min\left\{\frac{|\mathcal{C}_i \cap \mathcal{C}_j|}{|\mathcal{C}_i|}, \frac{|\mathcal{C}_i \cap \mathcal{C}_j|}{|\mathcal{C}_j|}\right\} \geq \sigma. \tag{11}$$

This approach traverses the clique tree depth-first, performing the following steps for each clique $\mathcal{C}_\ell$:

1. For each clique pair $\{(\mathcal{C}_i, \mathcal{C}_j) \mid \mathcal{C}_i, \mathcal{C}_j \in \mathrm{ch}(\mathcal{C}_\ell)\}$, check if (11) holds, then:
   - $\mathcal{C}_i$ and $\mathcal{C}_j$ are merged, or
   - if $(\mathcal{C}_i \cap \mathcal{C}_j) \supseteq \mathcal{C}_\ell$, then $\mathcal{C}_i$, $\mathcal{C}_j$, and $\mathcal{C}_\ell$ are merged.
2. For each clique pair $\{(\mathcal{C}_i, \mathcal{C}_\ell) \mid \mathcal{C}_i \in \mathrm{ch}(\mathcal{C}_\ell)\}$, merge $\mathcal{C}_i$ and $\mathcal{C}_\ell$ if (11) is satisfied.

We remark that the implementation of `SparseCoLO` follows the algorithm outlined here, but also employs a few additional heuristics.

An advantage of the two approaches is that the clique tree can be computed easily and the conditions are inexpensive to evaluate. However, a disadvantage is that choosing parameters that work well on a variety of problems and solver algorithms is difficult. Secondly, in some cases it is beneficial to merge cliques that are not directly related on the clique tree. To see this, consider a chordal graph $G(V, E)$ consisting of three connected subgraphs:

$$G_a(V_a, E_a), \text{ with } V_a = \{3, 4, \dots, m_a\},$$
$$G_b(V_b, E_b), \text{ with } V_b = \{m_a + 2, m_a + 3, \dots, m_b\},$$
$$G_c(V_c, E_c), \text{ with } V_c = \{m_b + 1, m_b + 2, \dots, m_c\},$$

and some additional vertices $\{1, 2, m_a + 1\}$. The graph is connected as shown in Fig. 2(a), where the complete subgraphs are represented as nodes V_a, V_b, V_c. A corresponding clique tree is shown in Fig. 2(b). By choosing the

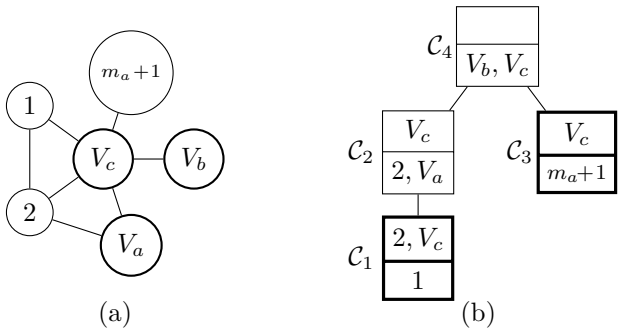

Fig. 2. Sparsity graph (a) that can lead to a clique tree (b) with an advantageous "nephew-uncle" merge between $\mathcal{C}_1$ and $\mathcal{C}_3$.

cardinality $|V_c|$, the overlap between cliques $\mathcal{C}_1 = \{1, 2\} \cup V_c$ and $\mathcal{C}_3 = \{m_a + 1\} \cup V_c$ can be made arbitrarily large while $|V_a|$, $|V_b|$ can be chosen so that any other merge is disadvantageous. However, neither the parent-child strategy nor `SparseCoLO` would consider merging $\mathcal{C}_1$ and $\mathcal{C}_3$ since they are in a "nephew-uncle" relationship.

2.2 A new clique graph-based strategy

To overcome the limitations of existing strategies we propose a merging strategy based on the *clique-(intersection) graph* $\mathcal{G}(\mathcal{B}, \xi)$, where the edge set ξ is defined as

$$\xi = \{(\mathcal{C}_i, \mathcal{C}_j) \mid i \neq j,\ \mathcal{C}_i, \mathcal{C}_j \in \mathcal{B},\ |\mathcal{C}_i \cap \mathcal{C}_j| > 0\}.$$

Let us further define an *edge weighting function* $e\colon 2^V \times 2^V \to \mathbb{R}$ that assigns a weight w_{ij} to each edge $(\mathcal{C}_i, \mathcal{C}_j) \in \xi$:

$$e(\mathcal{C}_i, \mathcal{C}_j) = w_{ij}.$$

This function is used to estimate the per-iteration computational savings of merging a pair of cliques depending on the targeted algorithm and hardware. It is chosen to evaluate to a positive number if a merge would reduce the per-iteration time and to a negative number otherwise. For a first-order method, whose per-iteration cost is dominated by an eigenvalue factorisation with complexity $\mathcal{O}\left(|\mathcal{C}|^3\right)$, a naive implementation would be:

$$e(\mathcal{C}_i, \mathcal{C}_j) = |\mathcal{C}_i|^3 + |\mathcal{C}_j|^3 - |\mathcal{C}_i \cup \mathcal{C}_j|^3. \tag{12}$$

More sophisticated weighting functions can be determined empirically; see Section 3. After a weight has been computed for each edge $(\mathcal{C}_i, \mathcal{C}_j)$ in the clique graph, we merge cliques as outlined in Alg. 2. Our strategy considers the edges in terms of their weights, starting with the clique pair $(\mathcal{C}_i, \mathcal{C}_j)$ with the highest weight w_{ij}. If the weight is positive, the two cliques are merged and the edge weights for all edges connected to the merged clique $\mathcal{C}_m = \mathcal{C}_i \cup \mathcal{C}_j$ are updated. This process continues until no edges with positive weights remain.

Algorithm 2: Clique graph-based merging strategy.

Input : A weighted clique graph $\mathcal{G}(\mathcal{B}, \xi)$.
Output: A merged clique graph $\mathcal{G}(\hat{\mathcal{B}}, \hat{\xi})$.

$\hat{\mathcal{B}} \leftarrow \mathcal{B}$ and $\hat{\xi} \leftarrow \xi$;
STOP $\leftarrow$false;
while *!STOP* **do**
  choose $(\mathcal{C}_i, \mathcal{C}_j)$ with maximum w_{ij};
  if $w_{ij} > 0$ **then**
    $\mathcal{B}_m \leftarrow \{\mathcal{C}_i, \mathcal{C}_j\}$;
    $\hat{\mathcal{B}}, \hat{\xi}, \mathcal{C}_m \leftarrow \text{mergeCliques}\left(\hat{\mathcal{B}}, \hat{\xi}, \mathcal{B}_m\right)$;
    for *each edge* $(\mathcal{C}_m, \mathcal{C}_\ell) \in \hat{\xi}$ **do**
      update $w_{m\ell} \leftarrow e(\mathcal{C}_m, \mathcal{C}_\ell)$;
  else
    STOP $\leftarrow$true;

The clique graph for the clique tree in Fig. 1(c) is shown in Fig. 1(d) with the edge weighting function in (12). Following Alg. 2 the edge with the largest weight is considered first and the corresponding cliques are merged, i.e. $\{3, 6, 7, 8\}$ and $\{6, 7, 8, 9\}$. The revised clique graph $\mathcal{G}(\hat{\mathcal{B}}, \hat{\xi})$ is shown in Fig. 3. Since no edges with positive weights remain, the algorithm stops.

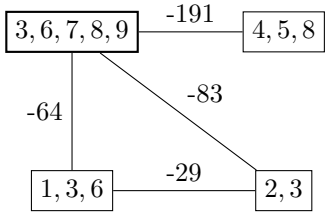

Fig. 3. Clique graph $\mathcal{G}(\hat{\mathcal{B}}, \hat{\xi})$ after merging the cliques $\{3, 6, 7, 8\}$ and $\{6, 7, 8, 9\}$ in Fig. 1(d) and updating edge weights.

After Alg. 2 has terminated, it is possible to recompute a valid clique tree from the revised clique graph. This can be done in two steps. First, the edge weights in $\mathcal{G}(\hat{\mathcal{B}}, \hat{\xi})$ are replaced with new weights:

$$\tilde{w}_{ij} = |\mathcal{C}_i \cap \mathcal{C}_j|, \text{ for all } (\mathcal{C}_i, \mathcal{C}_j) \in \hat{\xi}.$$

Second, a clique tree is then given by any *maximum weight spanning tree* of the newly weighted clique graph, which can be computed using e.g. the algorithm described in Kruskal (1956).

Our merging strategy has some advantages over competing approaches. Since the clique graph covers a wider range of merge candidates, it will consider edges that do not appear in clique tree-based approaches (such as the "nephew-uncle" example in Fig. 2. Moreover, the edge weighting function allows one to make a merge decision based on the particular solver algorithm and hardware used. One downside is that this approach is more computationally involved than the other methods. However, experiments show that the extra time spent on finding the clique graph, merging the cliques, and recomputing the clique tree is only a fraction of the total computational savings.

3. IMPLEMENTATION AND RESULTS

To compare the proposed merge approach with the clique tree-based strategies of Nakata et al. (2003) and Sun et al. (2014), all three methods were used to preprocess sparse SDPs from SDPLib, a collection of SDP benchmark problems (Borchers, 1999). Each strategy was given the same initial clique decomposition, and the resulting decomposed SDPs were solved using the first-order solver `COSMO` (Garstka et al., 2019). This section discusses how the different decompositions affect the per-iteration computation times of the solver.

For the strategy described in Nakata et al. (2003) we used the `SparseCoLO` package to decompose the problem. The parent-child method by Sun et al. (2014) and our clique graph-based method are available in the latest version of our conic solver `COSMO`. We further investigate the effect of using different edge weighting functions. Since `COSMO` is an ADMM-solver, the major operation affecting the per-iteration time is the projection step (see Garstka et al., 2019, for more details). This operation involves an eigenvalue decomposition of the matrices corresponding to the cliques. Since the eigenvalue decomposition of a symmetric matrix of dimension N has a complexity of $\mathcal{O}\left(N^3\right)$, we define a *nominal* edge weighting function as in (12). However, the exact relationship will be different because the projection function involves copying of data and is affected by hardware properties such as cache size. We therefore also consider an *estimated* edge weighting function. To determine the relationship between matrix size and projection time, the execution time of the relevant function inside `COSMO` was measured for different matrix sizes. We then approximated the relationship between projection time, t_{proj}, and matrix size, N, as a polynomial:

$$t_{\text{proj}}(N) = aN^3 + bN^2,$$

where a, b were estimated using least squares (Fig. 4). The estimated weighting function is then defined as

$$e(\mathcal{C}_i, \mathcal{C}_j) = t_{\text{proj}}\left(|\mathcal{C}_i|\right) + t_{\text{proj}}\left(|\mathcal{C}_j|\right) - t_{\text{proj}}\left(|\mathcal{C}_i \cup \mathcal{C}_j|\right). \quad (13)$$

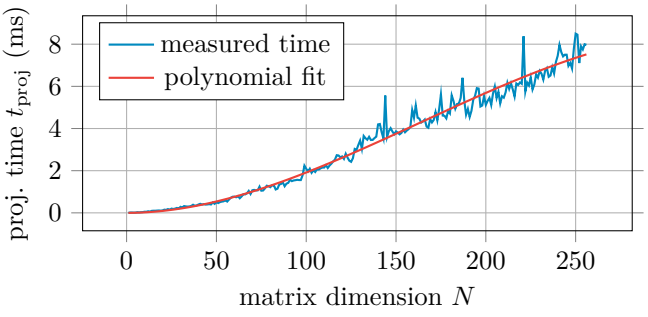

Fig. 4. Measured and estimated relationship between matrix size and execution time of the projection function in `COSMO`.

We emphasize that the same strategy could also be used for other solver algorithms. For an interior-point method this would mean replacing (13) with a (more complex) weighting function that takes into account both the impact of the merge on the clique sizes and the number of additional equality constraints.

The merging strategies were compared for large, sparse SDP problems with chordal sparsity patterns from the

Table 1. Benchmark results for different merging strategies.

problem	Solve time (s)						Projection time (ms)					
	NoDe[1]	NoMer[2]	SpCo[3]	ParCh[4]	CG1[5]	CG2[6]	NoDe	NoMer	SpCo	ParCh	CG1	CG2
maxG11	29.7	4.11	7.9	3.69	**2.72**	2.82	99.1	15.3	11.8	12.3	12.1	**9.2**
maxG32	320.98	21.12	27.08	13.09	**12.47**	15.79	1105.0	58.1	57.8	46.4	38.3	**34.5**
maxG51	29.12	28.04	19.86	9.59	**5.67**	8.25	171.4	182.9	191.9	89.6	54.3	**43.2**
mcp500-1	10.28	1.04	1.19	0.78	0.47	**0.37**	40.4	5.9	6.9	4.5	3.4	**2.7**
mcp500-2	8.9	10.25	7.61	5.97	2.08	**1.95**	35.2	37.7	34.5	18.9	11.6	**8.8**
mcp500-3	7.66	22.69	30.45	15.76	5.41	**4.35**	35.7	82.3	83.9	49.9	24.2	**16.3**
mcp500-4	11.63	51.37	60.52	21.92	**5.32**	8.74	39.6	180.9	132.3	93.8	36.8	**28.3**
qpG11	173.81	6.05	6.48	7.65	4.14	**3.87**	397.2	16.9	11.9	12.5	13.4	**10.9**
qpG51	607.61	138.38	155.04	150.14	113.87	**85.19**	749.9	201.6	185.6	99.7	58.8	**48.9**
thetaG11	225.89	8.28	37.16	10.24	9.01	**5.95**	292.4	20.9	15.6	14.8	15.6	**12.8**
thetaG51	505.33	82.48	587.79	103.47	**28.28**	78.08	193.4	199.5	204.6	93.5	58.8	**43.1**

problem	Iterations						Number of cliques / Maximum clique size					
	NoDe	NoMer	SpCo	ParCh	CG1	CG2	NoDe	NoMer	SpCo	ParCh	CG1	CG2
maxG11	280	240	640	280	**200**	280	1/800	598/24	13/80	207/32	473/28	411/38
maxG32	280	320	440	**240**	280	400	1/2000	1498/76	21/210	478/76	1164/92	468/126
maxG51	160	120	80	80	80	160	1/1000	674/326	181/322	172/326	448/362	256/422
mcp500-1	240	160	160	160	120	120	1/500	457/39	451/44	111/44	437/54	334/65
mcp500-2	240	240	200	280	**160**	200	1/500	363/138	144/138	111/140	316/156	223/177
mcp500-3	200	240	320	280	200	240	1/500	259/242	101/242	70/242	211/263	134/301
mcp500-4	280	240	400	200	**120**	280	1/500	161/340	63/346	52/341	105/368	85/413
qpG11	400	320	520	560	**280**	320	1/1600	1398/24	813/80	296/32	1273/28	1211/38
qpG51	760	**600**	720	1360	1800	1640	1/2000	1674/326	1182/304	284/326	1448/362	1256/422
thetaG11	760	**360**	2280	640	520	400	1/801	598/25	13/81	207/33	494/29	423/41
thetaG51	2500	**320**	2500	920	360	1560	1/1001	676/324	150/323	169/324	424/358	202/425

[1] no decomposition; [2] no merging; [3] `SparseCoLO` merging; [4] parent-child merging;
[5] clique graph with nominal edge weighting (12); [6] clique graph with estimated edge weighting (13)

SDPLib benchmark library. This problem set contains maximum cut problems, SDP relaxations of quadratic programs and Lovasz theta problems. Six different cases were considered: no decomposition (`NoDe`), no clique merging (`NoMer`), decomposition using `SparseCoLO` (`SpCo`), parent-child merging (`ParCh`), and the clique graph-based method with nominal edge weighting (`CG1`) and estimated edge weighting (`CG2`). All experiments were run on a MacBook with a 2.6 GHz Intel Core i5-8259U CPU and 8 GB of DDR3 RAM. `COSMO` was configured to terminate with accuracy $\epsilon_{\text{abs}} = \epsilon_{\text{rel}} = 5 \times 10^{-4}$. `SparseCoLO` was used with default parameters. Tab. 1 shows the total solve time, the mean projection time, the number of iterations, the number of cliques after merging, and the maximum clique size of the sparsity pattern. The minimum value of each row is highlighted.

Our clique graph-based methods lead to a reduction in overall solve time. The method with estimated edge weighting function `CG2` achieves the lowest average projection times. The geometric mean of the ratios of projection time of `CG2` compared to the best non-graph method is 0.613, with a minimum ratio of 0.458 for problem `mcp500-3`. Considering the number of cliques we see that `SparseCoLO` and `ParCh` merge more aggressively. Moreover, if the initial decomposition has a small maximum clique size, `SparseCoLO` seems to favor larger clique sizes. The merging strategies `ParCh`, `CG1` and `CG2` result in similar maximum clique sizes, with `CG1` being the most conservative in the number of merges.

4. CONCLUSION

A novel clique graph merging strategy to combine overlapping blocks in the aggregate sparsity pattern of structured SDPs is proposed. The method considers all possible pairwise merges and is customisable to the solver algorithm and hardware used. An extension to our method would include information about the number of available CPU threads in the edge weighting function. This would allow us to optimise the strategy for the parallel execution of the block-specific projection steps. Benchmark tests show that our approach is able to reduce the projection time and the solve time of our first-order solver significantly compared to existing clique merging methods.

ACKNOWLEDGEMENTS

We would like to thank Vidit Nanda and Heather Harrington for their helpful suggestions.

REFERENCES

Agler, J., Helton, W., McCullough, S., and Rodman, L. (1988). Positive semidefinite matrices with a given sparsity pattern. *Linear Algebra and its Applications*, 107, 101–149.

Amestoy, P.R., Davis, T.A., and Duff, I.S. (1996). An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4), 886–905.

Andersen, M. and Vandenberghe, L. (2015). CHOMPACK: A Python package for chordal matrix computations.

Borchers, B. (1999). SDPLIB 1.2, A library of semidefinite programming test problems. *Optimization Methods and Software*, 11(1-4), 683–690.

Fujisawa, K., Kim, S., Kojima, M., Okamoto, Y., and Yamashita, M. (2009). User's manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems. *Report B-453, Dept. of Math. and Comp. Sci. Japan, Tech. Rep.*, 152–8552.

Fujisawa, K., Fukuda, M., and Nakata, K. (2006). Preprocessing sparse semidefinite programs via matrix completion. *Optimization Methods and Software*, 21(1), 17–39.

Fukuda, M., Kojima, M., Murota, K., and Nakata, K. (2001). Exploiting sparsity in semidefinite programming via matrix completion I: General framework. *SIAM J. on Optimization*, 11(3), 647–674.

Garstka, M., Cannon, M., and Goulart, P. (2019). COSMO: A conic operator splitting method for large convex problems. In *European Control Conference*.

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

Kruskal, J.B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1), 48–50.

Molzahn, D.K., Holzer, J.T., Lesieutre, B.C., and DeMarco, C.L. (2013). Implementation of a large-scale optimal power flow solver based on semidefinite programming. *IEEE Trans. on Power Systems*, 28(4), 3987–3998.

Nakata, K., Fujisawa, K., Fukuda, M., Kojima, M., and Murota, K. (2003). Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results. *Math. Prog.*, 95(2), 303–327.

Nesterov, Y. and Nemirovsky, A. (1988). A general approach to polynomial-time algorithms design for convex programming. *Report, Central Economical and Mathematical Institute, USSR Academy of Sciences, Moscow*.

O'Donoghue, B., Chu, E., Parikh, N., and Boyd, S. (2016). Conic optimization via operator splitting and homogeneous self-dual embedding. *J. of Opt. Theory and Applications*, 169(3), 1042–1068.

Pothen, A. and Sun, C. (1990). Compact clique tree data structures in sparse matrix factorizations. *Large-Scale Numerical Optimization*, 180–204.

Sun, Y., Andersen, M.S., and Vandenberghe, L. (2014). Decomposition in conic optimization with partially separable structure. *SIAM J. on Optimization*, 24(2), 873–897.

Vandenberghe, L. and Andersen, M.S. (2015). Chordal graphs and semidefinite optimization. *Foundations and Trends® in Optimization*, 1(4), 241–433.

Yannakakis, M. (1981). Computing the minimum fill-in is NP-complete. *SIAM J. on Algebraic Discrete Methods*, 2(1), 77–79.

Zheng, Y., Fantuzzi, G., Papachristodoulou, A., Goulart, P., and Wynn, A. (2020). Chordal decomposition in operator-splitting methods for sparse semidefinite programs. *Mathematical Programming*, 180(1), 489–532.