Journal of Optimization Theory and Applications (2021) 190:779–810
https://doi.org/10.1007/s10957-021-01896-x

COSMO: A Conic Operator Splitting Method for Convex Conic Problems

Michael Garstka[1] · **Mark Cannon**[1] · **Paul Goulart**[1]

Received: 8 January 2021 / Accepted: 16 June 2021 / Published online: 29 August 2021
© Crown 2021

Abstract

This paper describes the conic operator splitting method (COSMO) solver, an operator splitting algorithm and associated software package for convex optimisation problems with quadratic objective function and conic constraints. At each step, the algorithm alternates between solving a quasi-definite linear system with a constant coefficient matrix and a projection onto convex sets. The low per-iteration computational cost makes the method particularly efficient for large problems, e.g. semidefinite programs that arise in portfolio optimisation, graph theory, and robust control. Moreover, the solver uses chordal decomposition techniques and a new clique merging algorithm to effectively exploit sparsity in large, structured semidefinite programs. Numerical comparisons with other state-of-the-art solvers for a variety of benchmark problems show the effectiveness of our approach. Our Julia implementation is open source, designed to be extended and customised by the user, and is integrated into the Julia optimisation ecosystem.

Keywords Conic programming · ADMM · Chordal decomposition · Clique merging

Mathematics Subject Classification 49J53 · 49K99 · More

Communicated by Suliman Saleh Al-Homidan.

Preliminary versions of this work appeared in [26,27]. MG is supported by the Clarendon Scholarship.

✉ Michael Garstka
michael.garstka@eng.ox.ac.uk

Mark Cannon
mark.cannon@eng.ox.ac.uk

Paul Goulart
paul.goulart@eng.ox.ac.uk

[1] University of Oxford, Parks Road, Oxford OX1 3PJ, UK

1 Introduction

We consider convex optimisation problems in the form

$$\begin{array}{ll} \text{minimise} & f(x) \\ \text{subject to} & g_i(x) \preceq_{\mathcal{K}_i} 0, \; i = 1, \ldots, l \\ & h_i(x) = 0, \quad i = 1, \ldots, k, \end{array} \tag{1}$$

with proper convex cones $\mathcal{K}_i \subseteq \mathbb{R}^{m_i}$ and where we assume that both the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the inequality constraint functions $g_i : \mathbb{R}^{m_i} \to \mathbb{R}$ are convex, and that the equality constraints $h_i(x) := a_i^\top x - b_i$ are affine. Convex optimisation problems feature heavily in a wide range of research areas and industries, including problems in machine learning [16], finance [14], optimal control [12], and operations research [11]. Concrete examples of problems fitting the general form (1) include linear programming (LP), convex quadratic programming (QP), second-order cone programming (SOCP), and semidefinite programming (SDP) problems. Methods to solve each of these standard problem classes are well known and a number of open- and closed-source solvers are widely available. However, the trend for data and training sets of increasing size in decision making problems and machine learning poses a challenge for state-of-the-art software.

The most common approach taken by modern convex solvers is the *interior-point method* (IPM) [63], which stems from Karmarkar's original projective algorithm [35], and is able to solve both LPs and QPs in polynomial time. IPMs have since been extended to problems with positive semidefinite (PSD) constraints in [3,30]. The primal-dual IPMs apply variants of Newton's method to iteratively find a solution to a set of optimality KKT conditions. At each iteration, the algorithm alternates between a Newton step that involves factoring a Jacobian matrix and a line search to determine the magnitude of the step to ensure a feasible iterate. Most notably, the Mehrotra predictor–corrector method in [39] forms the basis of several implementations because of its strong practical performance [62]. However, IPMs typically do not scale well for very large problems since the Jacobian matrix must be re-calculated and re-factored at each step.

There are two main approaches to overcome this limitation, both of which are areas of active research. The first of these is a renewed focus on first-order methods with computationally cheaper per-iteration-cost. First-order methods are known to handle larger problems well at the expense of reduced accuracy compared to IPMs. In the 1960s, Everett [21] proposed a dual decomposition method that allows one to decompose a separable objective function, making each iteration cheaper to compute. Augmented Lagrangian methods by Miele [40–42], Hestenes [31], and Powell [49] are more robust and helped to remove the strict convexity conditions on problems, while losing the decomposition property. By splitting the objective function, the alternating direction method of multipliers (ADMM), first described in [25], allowed the advantages of dual decomposition to be combined with the superior convergence and robustness of augmented Lagrangian methods. Subsequently, it was shown that ADMM can be analysed from the perspective of monotone operators as a special case of Douglas–Rachford splitting [20] as well as of the proximal point algorithm in

[52], which allowed further insight into the method. ADMM methods are simple to implement and computationally cheap, even for large problems. However, they tend to converge slowly to a high accuracy solution and the detection of infeasibility is more involved. Thus, they are typically used in applications where a modestly accurate solution is sufficient [48].

The second area of active research has been the exploitation of sparsity in the problem data. A method of exploiting the sparsity pattern of PSD constraints in an IPM was developed in [24]. This work showed that if the coefficient matrices of an SDP exhibit an aggregate sparsity pattern represented by a chordal graph, then both primal and dual problems can be decomposed into a problem with many smaller PSD constraints on the nonzero blocks of the original matrix variable. These blocks are associated with subsets, called *cliques*, of graph vertices. Moreover, it can be advantageous to merge some of these blocks, for example, if they overlap significantly. The optimal way to merge the blocks, or equivalently the corresponding graph cliques, after the initial decomposition depends on the solver algorithm and is still an open question. Sparsity in SDPs has also been studied for problems with underlying graph structure, e.g. optimal power-flow problems in [44] and graph optimisation problems in [2].

Related Work Widely used solvers for conic problems, especially SDPs, include SeDuMi [57], SDPT3 [59] (both open source, MATLAB), and MOSEK [45] (commercial, C) among others. All of these solvers implement primal-dual IPMs. Both Fukuda [24] and Sun [58] developed interior-point solvers that exploit chordal sparsity patterns in PSD constraints. Some heuristic methods to merge cliques have been proposed for IPMs in [44] and been implemented in the SparseCoLO package [22] and the CHOMPACK package [4]. Several solvers based on the ADMM method have been released recently. The solver OSQP [56] (C) detects infeasibility based on the differences of the iterates [7], but only solves LPs and QPs. The C-based SCS [47] implements an operator splitting method that solves the primal-dual pair of conic programs in order to provide infeasibility certificates. The underlying homogeneous self-dual embedding method has been extended by [66] to exploit sparsity and implemented in the MATLAB solver CDCS. The conic solvers SCS and CDCS are unable to handle quadratic cost functions directly. Instead, they are forced to reformulate problem with quadratic objective functions by adding a second-order cone constraint, which increases the problem size and removes any favourable sparsity. Moreover, they rely on self-dual embedding formulations to detect infeasibility.

Outline In Sect. 2, we define the general conic problem format of interest and describe the ADMM algorithm used by COSMO. Section 3 explains how to apply chordal decomposition to large sparse SDPs. In Sect. 4, we describe a new clique merging strategy for SDPs and compare it to existing approaches. Implementation details and code-related design choices are discussed in Sect. 5. Section 6 shows benchmark results of COSMO vs. other state-of-the art solvers on a number of test problems. Section 7 concludes the paper.

Contributions In this paper we make the following contributions:

1. We describe a first-order method for large conic problems that directly supports quadratic objective functions, thus reducing overheads for applications with both

quadratic objective function and PSD constraints. This also avoids a major disadvantage of conic solvers compared to native QP solvers, i.e. no additional matrix factorisation for the conversion is needed and favourable sparsity in the objective can be maintained. These benefits are demonstrated using benchmark sets of QPs and SDPs with norm constraints.
2. For large SDPs with complex sparsity patterns, we show that substantial performance improvements can be achieved relative to standard chordal decomposition methods by recombining some of the sub-blocks (i.e. the cliques) of the initial decomposition in an optimal way. We show that our new clique graph-based merging strategy reduces the projection time of the algorithm by up to 60% in benchmark tests. The combination of this decomposition strategy with a multithreaded projection step allows us to solve very large sparse and structured problems orders of magnitudes faster than competing solvers.
3. We describe our open-source solver COSMO written in the fast and flexible programming language Julia. The design of this package is highly modular, allowing users to extend the solver by specifying a custom linear system solvers and by defining their own convex cones and custom projection methods. We show how a custom projection method can speed up the algorithm by a factor of 5 for large QPs.

Notation The following notation and definitions will be used throughout this paper. Denote the space of real numbers $\mathbb{R}$, the n-dimensional real space $\mathbb{R}^n$, the n-dimensional zero cone $\{0\}^n$, the nonnegative orthant $\mathbb{R}^n_+$, the space of symmetric matrices $\mathbb{S}^n$, and the set of positive semidefinite matrices $\mathbb{S}^n_+$. Denote the vectorisation of a matrix X by stacking it columns as $x = \text{vec}(X)$ and the inverse operation as $\text{vec}^{-1}(X) = \text{mat}(x)$. We denote the space of vectorised symmetric positive semidefinite matrices as $\mathcal{S}^n_+$. For a convex cone $\mathcal{K}$, denote the *polar cone* by $\mathcal{K}^\circ = \left\{ y \in \mathbb{R}^n \mid \sup_{x \in \mathcal{K}} \langle x, y \rangle \leq 0 \right\}$ and, following [51], the *recession cone* of $\mathcal{K}$ by $\mathcal{K}^\infty = \{y \in \mathbb{R}^n \mid x + ay \in \mathcal{K},\ \forall x \in \mathcal{K}, \forall a \geq 0\}$. We denote the *indicator function* of a nonempty, closed convex set $\mathcal{C} \subseteq \mathbb{R}^n$ by $I_\mathcal{C}$ and the *projection* of $x \in \mathbb{R}^n$ onto $\mathcal{C}$ by $\Pi_\mathcal{C}(x) = \operatorname{argmin}_{y \in \mathcal{C}} \|x - y\|_2^2$.

2 Problem Description and Core Algorithm

We address convex optimisation problems with a quadratic objective function and a number of conic constraints in the standard form:

$$
\begin{array}{ll}
\text{minimise} & \frac{1}{2}x^\top P x + q^\top x \\
\text{subject to} & Ax + s = b,\ s \in \mathcal{K},
\end{array}
\tag{2}
$$

where $x \in \mathbb{R}^n$ is the primal *decision variable* and $s \in \mathbb{R}^m$ is the primal *slack variable*. The objective function is defined by positive semidefinite matrix $P \in \mathbb{S}^n_+$ and vector $q \in \mathbb{R}^n$. The constraints are defined by matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and a non-empty, closed, convex cone $\mathcal{K}$ which itself can be a Cartesian product of cones in the form

$$\mathcal{K} = \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \cdots \times \mathcal{K}_N^{m_N}, \tag{3}$$

with cone dimensions $\sum_{i=1}^{N} m_i = m$. Note that any LP, QP, SOCP, or SDP can be written in the form (2) using an appropriate choice of cones. The conditions for optimality (assuming linear independence constraint qualification) follow from the Karush–Kuhn–Tucker (KKT) conditions:

$$Ax + s = b, \quad Px + q - A^\top y = 0, \quad s \in \mathcal{K}, \quad y \in (\mathcal{K}^\infty)^\circ. \tag{4}$$

Assuming strong duality, if there exists a $x^* \in \mathbb{R}^n$, $s^* \in \mathbb{R}^m$, and $y^* \in \mathbb{R}^m$ that fulfil (4) then the pair (x^*, s^*) is called the primal solution and y^* is called the dual solution of problem (2).

ADMM Algorithm We employ the splitting used in [56] and extend it to accommodate constraints defined using proper convex cones and transform problem (2) into ADMM-compatible format. The problem is rewritten by introducing the dummy variables $\tilde{x} = x$ and $\tilde{s} = s$:

$$\begin{aligned} &\text{minimise } \tfrac{1}{2}\tilde{x}^\top P \tilde{x} + q^\top \tilde{x} + I_{Ax+s=b}(\tilde{x}, \tilde{s}) + I_{\mathcal{K}}(s) \\ &\text{subject to} (\tilde{x}, \tilde{s}) = (x, s), \end{aligned} \tag{5}$$

where the indicator functions of the sets $\{(x, s) \in \mathbb{R}^n \times \mathbb{R}^m \mid Ax + s = b\}$ and $\mathcal{K}$ were used to move the constraints of (2) into the objective. We then apply standard form ADMM [13] to (5) and refer the reader to [56] for a derivation of the algorithm steps shown in Algorithm 1.

Algorithm 1: ADMM iteration

Input : initial values x^0, s^0, y^0, problem data P, q, A, b, penalty parameters $\sigma > 0$, $\rho > 0$, and relaxation parameter $\alpha \in (0, 2)$

1 **Do**

2 $\quad (\tilde{x}^{k+1}, \tilde{s}^{k+1}) \leftarrow \underset{A\tilde{x}+\tilde{s}=b}{\text{argmin}} \tfrac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top \tilde{x} + \tfrac{\sigma}{2}\left\|\tilde{x} - x^k\right\|_2^2 + \tfrac{\rho}{2}\left\|\tilde{s} - s^k + \tfrac{1}{\rho}y^k\right\|_2^2$;

3 $\quad x^{k+1} \leftarrow \Pi_{\mathbb{R}^n}\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k\right)$;

4 $\quad s^{k+1} \leftarrow \Pi_{\mathcal{K}}\left(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k + \tfrac{1}{\rho}y^k\right)$;

5 $\quad y^{k+1} \leftarrow y^k + \rho(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s^{k+1})$;

6 **while** *termination criteria not satisfied*;

The evaluation of line 2 of the algorithm amounts to solving an equality constrained QP in the variables $\tilde{x}$ and $\tilde{s}$. The solution is obtained from the reduced linear system representing the corresponding KKT optimality conditions

$$\begin{bmatrix} P + \sigma I & A^\top \\ A & -\frac{1}{\rho}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} -q + \sigma x^k \\ b - s^k + \frac{1}{\rho}y^k \end{bmatrix}, \quad \tilde{s}^{k+1} = s^k - \frac{1}{\rho}\left(\nu^{k+1} + y^k\right), \tag{6}$$

with dual variable $\nu \in \mathbb{R}^m$.

The particular choice of splitting in (5) ensures that the coefficient matrix in (6) is always quasi-definite, and hence always has a well-defined LDL$^\top$ factorisation.

Assuming a problem dimension $N = m + n$, the factorisation must be computed only once at the start of the algorithm at a cost of $\frac{1}{3}N^3$ flops. Afterwards, the evaluation of line 2 involves forward- and back-substitution steps involving at most $2N^2$ flops. However, in practice the coefficient matrix is very sparse, so a permutation allows us to achieve much sparser factors and therefore computationally cheaper substitution steps. Lines 3 and 5 involve only vector scaling and additions at a complexity of $\mathcal{O}(n)$ and $\mathcal{O}(m)$, respectively. The projection onto $\mathcal{K}$ in line 4 is crucial to the performance of the algorithm depending on the particular cones employed in the model: projections onto the zero-cone or the nonnegative orthant are inexpensive at $\mathcal{O}(m)$, while a projection onto the positive-semidefinite cone of dimension $n_{\mathcal{K}}$ involves an eigenvalue decomposition. Since direct methods for eigendecompositions have a complexity of $\mathcal{O}(n_{\mathcal{K}}^3)$, this turns line 4 into the most computationally expensive operation of the algorithm for large SDPs; Sects. 3 and 4 describe how to improve the efficiency of this step.

Termination Criteria To measure progress of the algorithm, we define the primal and dual residuals of the problem as:

$$r_p = Ax + s - b, \quad r_d = Px + q - A^\top y. \tag{7a}$$

According to Section 3.3 of [13], a valid termination criterion is that the size of the norms of the residual iterates in (7) are small. Our algorithm terminates if the residual norms are below the sum of an absolute and a relative tolerance term:

$$\left\| r_p^k \right\|_\infty \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max \left\{ \left\| Ax^k \right\|_\infty, \left\| s^k \right\|_\infty, \|b\|_\infty \right\}, \tag{8a}$$

$$\left\| r_d^k \right\|_\infty \leq \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max \left\{ \left\| Px^k \right\|_\infty, \|q\|_\infty, \left\| A^\top y^k \right\|_\infty \right\}, \tag{8b}$$

where ϵ_{abs} and ϵ_{rel} are user defined tolerances. Moreover, we use the infeasibility conditions based on one-step differences $\delta x^k = x^k - x^{k-1}$, $\delta y^k = y^k - y^{k-1}$ in [7] to detect infeasibility and provide certificates.

3 Chordal Decomposition

As noted in Sect. 2, for large SDPs the principal performance bottleneck for Algorithm 1 is the eigendecomposition required in the projection step of line 4. However, since large-scale SDPs often exhibit a certain structure or sparsity pattern, a sensible strategy is to exploit any such structure to alleviate this bottleneck. It is well known that, if the aggregated sparsity pattern is *chordal*, Agler's [1] and Grone's [28] theorems can be used to decompose a large PSD constraint into a collection of smaller PSD constraints and additional coupling constraints. The projection step applied to the set of smaller PSD constraints is usually significantly faster than when applied to the original constraint. Since the projections are independent, they are also easily parallelised.

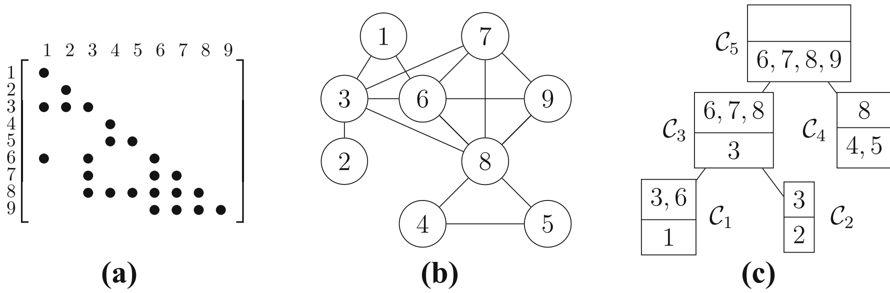

Fig. 1 **a** Aggregate sparsity pattern, **b** sparsity graph $G(V, E)$, and **c** clique tree $\mathcal{T}(\mathcal{B}, \mathcal{E})$

In this section, we describe the standard mathematical machinery behind chordal decomposition, with a view to improving it substantially in Sect. 4 when applied to large-scale SDPs.

3.1 Graph Preliminaries

In the following, we define graph-related concepts that are useful to describe the sparsity structure of a problem. A good overview of this topic is given in the survey paper [61]. Consider the *undirected graph* $G(V, E)$ with vertex set $V = \{1, \dots, n\}$ and edge set $E \subseteq V \times V$. If all vertices are pairwise adjacent, i.e. $E = \{\{v, u\} \mid v, u \in V,\ v \neq u\}$, the graph is called *complete*. We follow the convention of [61] by defining a *clique* as a subset of vertices $\mathcal{C} \subseteq V$ that induces a *maximal* complete subgraph of G. The number of vertices in a *clique* is given by the cardinality $|\mathcal{C}|$. 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. The following decomposition theory relies on a subset of graphs that exhibit the important property of *chordality*. A graph G 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 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. 1a, b.

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

$$\mathbb{S}^n(E, 0) = \left\{ S \in \mathbb{S}^n \mid S_{ij} = S_{ji} = 0 \text{ if } i \neq j \text{ and } (i, j) \notin E \right\}, \tag{9}$$

$$\mathbb{S}^n_+(E, 0) = \left\{ S \in \mathbb{S}^n(E, 0) \mid S \succeq 0 \right\}. \tag{10}$$

Given the definition in (9) and a matrix $S \in \mathbb{S}^n(E, 0)$, note that 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 as

$$\mathbb{S}^n_+(E, ?) = \left\{ Y \mid \exists \hat{Y} \in \mathbb{S}^n_+, Y_{ij} = \hat{Y}_{ij}, \text{if } i = j \text{ or } (i, j) \in E \right\}. \tag{11}$$

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 [61]. An important structure that conveys a lot of 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*, i.e. 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 the *clique-intersection property* in [46] and the *induced subtree property* in [61]. For a given chordal graph, a clique tree can be computed using the algorithm described in [15]. The clique tree for an example sparsity pattern is shown in Fig. 1c. In a *post-ordered* clique tree the descendants of a node are given consecutive numbers, and a suitable post-ordering can be found via depth-first search. 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*.

We define the function $\text{par} : 2^V \to 2^V$ and the multivalued function $\text{ch} : 2^V \rightrightarrows 2^V$ such that $\text{par}(\mathcal{C}_\ell)$ and $\text{ch}(\mathcal{C}_\ell)$ return the parent clique and set of child cliques of $\mathcal{C}_\ell$, respectively, where 2^V is the power set (set of all subsets) of V. Note that each clique in Fig. 1c 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. Following the authors in [29], we say that two cliques $\mathcal{C}_i, \mathcal{C}_j$ form a separating pair $\mathcal{P}_{ij} = \{\mathcal{C}_i, \mathcal{C}_j\}$ if their intersection is non-empty and if every path in the underlying graph G from a vertex $\mathcal{C}_i \setminus \mathcal{C}_j$ to a vertex $\mathcal{C}_j \setminus \mathcal{C}_i$ contains a vertex of the intersection $\mathcal{C}_i \cap \mathcal{C}_j$.

3.2 Agler's and Grone's Theorems

To explain how the concepts in the previous section can be used to decompose a positive semidefinite constraint, we first consider an SDP in standard primal form and standard dual form:

$$\begin{array}{llll} \text{minimise} & \langle C, X\rangle & \text{maximise} & b^\top y \\ \text{subject to} & \langle A_k, X\rangle = b_k,\ k = 1, \ldots, m & \text{subject to} & \sum_{k=1}^m A_k y_k + S = C \\ & X \in \mathbb{S}^n_+, & & S \in \mathbb{S}^n_+, \end{array} \tag{12}$$

with variable X, coefficient matrices $A_k, C \in \mathbb{S}^n$, dual variable $y \in \mathbb{R}^m$ and slack variable S. Let us assume that the problem matrices in (12) each have their own sparsity pattern $A_k \in \mathbb{S}^n(E_{A_k}, 0)$ 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 $G(V, \bar{E})$, where $E \subseteq \bar{E}$. Finding the minimum number of additional edges required to make the graph chordal is an NP-complete problem [64]. Consider a 0–1 matrix M with edge set E. A commonly used heuristic method to compute the chordal extension is to perform a symbolic Cholesky factorisation of M [24], with the edge set of the Cholesky factor then providing the chordal extension $\bar{E}$. To simplify notation, we assume that E represents a chordal graph or has been appropriately extended. Using the aggregate sparsity of the problem, we can modify the constraints on the matrix variables in (12) to be in the respective sparse positive semidefinite matrix spaces:

$$X \in \mathbb{S}^n_+(E, ?) \text{ and } S \in \mathbb{S}^n_+(E, 0). \tag{13}$$

We further define the entry-selector matrices $T_\ell \in \mathbb{R}^{|\mathcal{C}_\ell| \times n}$ for a clique $\mathcal{C}_\ell$ with $(T_\ell)_{ij} = 1$ if $\mathcal{C}_\ell(i) = j$ and 0 otherwise, where $\mathcal{C}_\ell(i)$ is the ith vertex of $\mathcal{C}_\ell$.

Assume that the sparsity pattern of the constraint is defined by a chordal graph $G(V, E)$ with a set of maximal cliques $\{\mathcal{C}_1, \dots, \mathcal{C}_p\}$. Then, we can express the constraints in (13) in terms of multiple smaller coupled constraints using Grone's and Agler's theorems:

$$\begin{aligned} &\text{Grone's theorem [39]}: X \in \mathbb{S}^n_+(E, ?) \Leftrightarrow X_\ell = T_\ell X T_\ell^\top \in \mathbb{S}^{|\mathcal{C}_\ell|}_+, \text{ for } \ell = 1, \dots, p. \\ &\text{Agler's theorem [38]}: S \in \mathbb{S}^n_+(E, 0) \Leftrightarrow \exists S_\ell \in \mathbb{S}^{|\mathcal{C}_\ell|}_+ \text{ for } \ell = 1, \dots, p \\ &\qquad \text{s.t. } S = \textstyle\sum_{\ell=1}^p T_\ell^\top S_\ell T_\ell. \end{aligned}$$

Applying Grone's theorem to the primal form SDP in (12) while restricting X to the positive semidefinite completable matrix cone as in (13) yields the decomposed primal SDP:

$$\begin{array}{llll} \text{minimise} & \langle C, X\rangle & \text{maximise} & b^\top y \\ \text{subject to} & \langle A_k, X\rangle = b_k, & \text{subject to} & \sum_{k=1}^m A_k y_k + \\ & X_\ell = T_\ell X T_\ell^\top, & & \sum_{\ell=1}^p T_\ell^\top S_\ell T_\ell = C \\ & X_\ell \in \mathbb{S}^{|\mathcal{C}_\ell|}_+, & & S_\ell \in \mathbb{S}^{|\mathcal{C}_\ell|}_+, \end{array} \tag{14}$$

for $k = 1, \dots, m$ and $\ell = 1, \dots, p$. Similarly, by utilising Agler's theorem we can transform the dual form SDP in (12) with the restriction on S in (13) to obtain the decomposed dual SDP. Note that the matrices T_ℓ serve to extract the submatrix S_ℓ such that $S_\ell = T_\ell S T_\ell^\top$ has rows and columns corresponding to the vertices of the $\mathcal{C}_\ell$.

4 Clique Merging

Given an initial decomposition with (chordally completed) edge set E and a set of cliques $\{\mathcal{C}_1, \dots, \mathcal{C}_p\}$, we are free to re-merge cliques back into larger blocks. This is equivalent to treating *structural* zeros in the problem as *numerical* zeros. Considering the decomposed dual problem in (14), the effects of merging two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ are twofold. First, 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. Second, 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 must be balanced, and the optimal merging strategy depends on the particular SDP solution algorithm employed.

In [46,58], a clique tree is used to search for favourable merge candidates. We will refer to their two approaches as SparseCoLO and the *parent–child* strategy, respectively, in the sequel. We will then propose a new merging method in Sect. 4.2 whose performance is superior to both methods when used in ADMM. Given a merging strategy, Algorithm 2 describes how to merge a set of cliques within the set $\mathcal{B}$ and update the edge set $\mathcal{E}$ accordingly.

Algorithm 2: Function $\mathrm{mergeCliques}(\mathcal{B}, \mathcal{E}, \mathcal{B}_m)$.

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 $\mathcal{C}_m$.

1 $\hat{\mathcal{E}} \leftarrow \mathcal{E}$;

2 $\mathcal{C}_m \leftarrow \mathcal{C}_{m,1} \cup \mathcal{C}_{m,2} \cup \dots \cup \mathcal{C}_{m,r}$;

3 $\hat{\mathcal{B}} \leftarrow (\mathcal{B} \setminus \mathcal{B}_m) \cup \{\mathcal{C}_m\}$;

4 Remove edges $\{(\mathcal{C}_i, \mathcal{C}_j) \mid i \neq j,\ \mathcal{C}_i, \mathcal{C}_j \in \mathcal{B}_m\}$ in $\hat{\mathcal{E}}$;

5 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 $\hat{\mathcal{E}}$;

4.1 Existing Clique Tree-Based Strategies

The parent–child strategy described in [58] traverses the clique tree depth-first and merges a clique $\mathcal{C}_\ell$ with its parent clique $\mathcal{C}_{\mathrm{par}(\ell)} = \mathrm{par}(\mathcal{C}_\ell)$ if at least one of the two following conditions are met:

$$\left(\left|\mathcal{C}_{\mathrm{par}(\ell)}\right| - |\eta_\ell|\right)\left(|\mathcal{C}_\ell| - |\eta_\ell|\right) \leq t_{\mathrm{fill}}, \quad \max\left\{|\nu_\ell|, \left|\nu_{\mathrm{par}(\ell)}\right|\right\} \leq t_{\mathrm{size}}, \tag{15}$$

with heuristic parameters t_{fill} and t_{size}. These conditions keep the amount of extra fill-in and the supernode cardinalities below the specified thresholds. The SparseCoLO strategy described in [23,46] 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

$$\min\left\{\frac{|\mathcal{C}_i \cap \mathcal{C}_j|}{|\mathcal{C}_i|}, \frac{|\mathcal{C}_i \cap \mathcal{C}_j|}{|\mathcal{C}_j|}\right\} \geq \sigma. \tag{16}$$

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

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 (16) holds, then:
   - $\mathcal{C}_i$ and $\mathcal{C}_j$ are merged, or

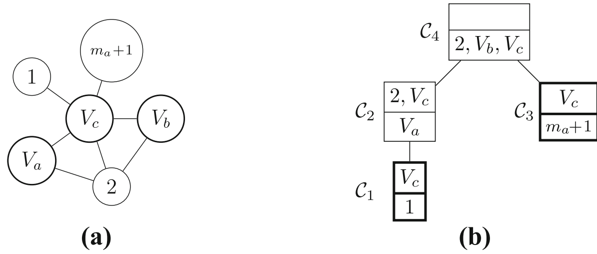

Fig. 2 Sparsity graph (**a**) that can lead to clique tree (**b**) with an advantageous "nephew–uncle" merge between $\mathcal{C}_1$ and $\mathcal{C}_3$

- if $(\mathcal{C}_i \cup \mathcal{C}_j) \supseteq \mathcal{C}_\ell$, then $\mathcal{C}_i$, $\mathcal{C}_j$, and $\mathcal{C}_\ell$ are merged.

2. For each clique pair $\{(\mathcal{C}_i, \mathcal{C}_\ell) \mid \mathcal{C}_i \in \text{ch}(\mathcal{C}_\ell)\}$, merge $\mathcal{C}_i$ and $\mathcal{C}_\ell$ if (16) is satisfied.

We note that the open-source implementation of the SparseCoLO algorithm described in [46] follows the algorithm outlined here, but also employs a few additional heuristics. An advantage of these 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, clique trees are not unique and in some cases it is beneficial to merge cliques that are not directly related on the particular clique tree that was computed. To see this, consider a chordal graph $G(V, E)$ consisting of three connected subgraphs $G_a(V_a, E_a)$, $G_b(V_b, E_b)$ and $G_c(V_c, E_c)$ with:

$$V_a = \{3, 4, \ldots, m_a\},\ V_b = \{m_a + 2, m_a + 3, \ldots, m_b\},\ V_c = \{m_b + 1, m_b + 2, \ldots, m_c\}$$

and some additional vertices $\{1, 2, m_a+1\}$. The graph is connected as shown in Fig. 2a, where the complete subgraphs are represented as nodes V_a, V_b, V_c. A corresponding clique tree is shown in Fig. 2b. By choosing the 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. In fact, for the particular sparsity graph in Fig. 2a, eight different clique trees can be computed. Only in half of them do the cliques $\mathcal{C}_1$ and $\mathcal{C}_3$ appear in a parent–child relationship. Therefore, a merge strategy that only considers parent–child relationships would miss this favourable merge in half the cases.

4.2 A New Clique Graph-Based Strategy

To overcome the limitations of existing strategies, we propose a merging strategy based on the *reduced clique graph* $\mathcal{G}(\mathcal{B}, \xi)$, which is defined as the union of all possible clique trees of G; see [29] for a detailed discussion. The set of vertices of this graph is given by the maximal cliques of the sparsity pattern. We then create the edge set ξ by introducing edges between pairs of cliques $(\mathcal{C}_i, \mathcal{C}_j)$ if they form a separating pair

$\mathcal{P}_{ij}$. We remark that ξ is a subset of the edges present in the *clique intersection graph* which is obtained by introducing edges for every two cliques that intersect. However, the reduced clique graph has the property that it remains a valid reduced clique graph of the altered sparsity pattern after performing a permissible merge between two cliques. This is not always the case for the clique intersection graph. For convenience, we will refer to the reduced clique graph simply as the clique graph in the following sections. Based on the permissibility condition for edge reduction in [29], we define a permissibility condition for merges:

Definition 4.1 (*Permissible merge*) Given a reduced clique graph $\mathcal{G}(\mathcal{B}, \xi)$, a merge between two cliques $(\mathcal{C}_i, \mathcal{C}_j) \in \xi$ is permissible if for every common neighbour $\mathcal{C}_k$ it holds that $\mathcal{C}_i \cap \mathcal{C}_k = \mathcal{C}_j \cap \mathcal{C}_k$.

We further define an *edge weighting function* $e\left(\mathcal{C}_i, \mathcal{C}_j\right) = w_{ij}$ with $e\colon 2^V \times 2^V \rightarrow \mathbb{R}$ that assigns a weight w_{ij} to each edge $(\mathcal{C}_i, \mathcal{C}_j) \in \xi$: 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 evaluates to a positive number if a merge reduces 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 $\mathcal{O}\big(|\mathcal{C}|^3\big)$, a simple choice would be:

$$e(\mathcal{C}_i, \mathcal{C}_j) = |\mathcal{C}_i|^3 + \left|\mathcal{C}_j\right|^3 - \left|\mathcal{C}_i \cup \mathcal{C}_j\right|^3. \tag{17}$$

More sophisticated weighting functions can be determined empirically; see Sect. 6.1. 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 Algorithm 3. This strategy considers the edges in terms of their weights, starting with the permissible 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 3: Clique graph-based merging strategy.
Input  : A weighted clique graph G(B, ξ).
Output: A merged clique graph G(B̂, ξ̂).
1  B̂ ← B and ξ̂ ← ξ;
2  STOP ← false;
3  while !STOP do
4      choose permissible edge (C_i, C_j) with maximum w_ij;
5      if w_ij > 0 then
6          B_m ← {C_i, C_j};
7          B̂, ξ̂, C_m ← mergeCliques(B̂, ξ̂, B_m);
8          for each edge (C_m, C_ℓ) ∈ ξ̂ do
9              update w_mℓ ← e(C_m, C_ℓ);
10     else
11         STOP ← true;
```

The clique graph for the clique tree in Fig. 1c is shown in Fig. 3a with the edge weighting function in (17). Following Algorithm 3, 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 merge is permissible because both cliques intersect with the only common neighbour {4, 5, 8} in the same way. The revised clique graph $\mathcal{G}(\hat{\mathcal{B}}, \hat{\xi})$ is shown in Fig. 3b. Since no positively weighted edges remain, the algorithm stops. After Algorithm 3 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 the cardinality of their intersection $\tilde{w}_{ij} = \left|\mathcal{C}_i \cap \mathcal{C}_j\right|$ 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, *e.g.* using Kruskal's algorithm described in [36].

Our merging strategy has several clear 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 expensive than the other methods. However, our numerical experiments show that the time spent on finding the clique graph, merging the cliques, and recomputing the clique tree is only a very small fraction of the total computational savings relative to other merging methods when solving SDPs.

5 Open-Source Implementation

We have implemented our algorithm in the conic operator splitting method (COSMO), an open-source package written in Julia [9]. The source code and documentation are available at https://github.com/oxfordcontrol/COSMO.jl.

Problems can be passed to COSMO via a direct interface, the Julia modelling packages JuMP [19] and Convex.jl [60], and a Python interface. By default, COSMO supports the zero cone, the nonnegative orthant, the hyperbox, the second-order cone, the PSD cone, the exponential cone and its dual, and the power cone and its dual.

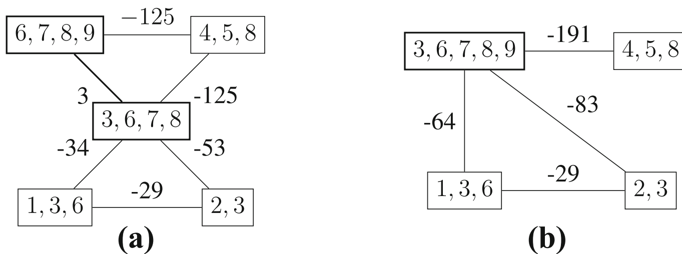

Fig. 3 **a** Clique graph $\mathcal{G}(\mathcal{B}, \xi)$ of the clique tree in Fig. 1c with edge weighting function $e(\mathcal{C}_i, \mathcal{C}_j) = |\mathcal{C}_i|^3 + \left|\mathcal{C}_j\right|^3 - \left|\mathcal{C}_i \cup \mathcal{C}_j\right|^3$ and **b** clique graph $\mathcal{G}(\hat{\mathcal{B}}, \hat{\xi})$ after merging the cliques {3, 6, 7, 8} and {6, 7, 8, 9} and updating edge weights

We further allow the user to define their own convex cones[1] and custom projection functions. To implement a custom cone $\mathcal{K}_c$ the user must provide a projection function that projects an input vector onto the cone. An example that shows the advantages of defining a custom cone is provided in Sect. 6.3. Custom projection functions in COSMO have been used by the authors in [53] to reduce solve time by a factor of 20. To solve the linear system in (6) the user can choose a direct solver such as QDLDL [56], SuiteSparse [17], Pardiso [34,55], or an indirect solver such as conjugate gradient, minimum residual method.

Multithreaded Projection Given a problem with multiple conic constraints, i.e. $\mathcal{K} = \mathcal{K}_1 \times \cdots \mathcal{K}_p$, the projection step in Algorithm 1

$$s^{k+1} = \Pi_{\mathcal{K}} \left(\alpha \tilde{s}^{k+1} + (1-\alpha) s^k + \frac{1}{\rho} y^k \right) \tag{18}$$

decomposes into p independent projections onto the individual cones and can be parallelised. This is particularly advantageous when used in combination with chordal decomposition, which typically yields a large number of smaller PSD constraints.

For the eigendecomposition involved in the projection step of a PSD constraint, the LAPACK [6] function `sveyr` is used, which can also utilise multiple threads. Consequently, this leads to two-level parallelism in the computation, i.e. on the higher level the projection functions are carried out in parallel and each projection function independently calls `sveyr`. Determining the optimal allocation of the CPU cores to each of these tasks depends on the number of PSD constraints and their dimensions and is a difficult problem. The impact of using different numbers of Julia threads and different BLAS settings on the accumulated projection time is shown for two example problems in Table 1.

We compare the cases that each call to `sveyr` is restricted to one single thread or that it gets automatically chosen by BLAS. We generate a best case SDP problem with a sparsity pattern, that can be decomposed into an equivalent problem with 120 10×10 PSD constraints. When each projection uses one thread, we see that doubling the number of threads halves the projection time. When BLAS chooses the number of threads for each projection dynamically we do not see an improvement for more than four threads. Next, we generate an SDP that decomposes into a problem with one dominant 400×400 block and 80 10×10 blocks. For the BLAS single-threaded configuration, this means that the projection of the large matrix block dominates and the number of threads do not make a significant difference.

Consequently, for the problem sets considered in Sect. 6 we achieved the best performance by running `sveyr` single-threaded and using all physical cores to carry out the projection functions in parallel.

[1] To allow infeasibility detection, the user has to either define a convex cone, a convex compact set, or a composition of the two.

Table 1 Impact of multithreading and BLAS settings on projection times

Threads[a]	BLAS single-threaded[b]		BLAS multithreaded[c]	
	Solve time (s)	Proj. time (s)	Solve time (s)	Proj. time (s)
Same size bl.				
1	56.29	30.72	58.33	32.06
2	42.04	16.16	43.05	16.41
4	34.10	8.51	37.08	8.87
8	30.39	4.77	36.41	8.54
Dominant block				
1	27.58	8.48	26.63	6.53
2	28.08	7.46	28.48	8.30
4	26.03	6.92	26.70	6.60
8	27.81	7.04	28.96	7.89

[a]Number of Julia threads with MKL BLAS on 8 physical Intel Xeon E5-2560 cores
[b]BLAS calls are restricted to one CPU thread
[c]BLAS calls can request number of threads automatically (independent of Julia threads)

6 Numerical Results

This section presents QP and SDP benchmark results of COSMO against the interior-point solver MOSEK v9.0 and the accelerated first-order ADMM solver SCS v2.1.1. When applied to a QP, COSMO's main algorithm becomes very similar to the first-order QP solver OSQP. To test the performance penalty of using a pure Julia implementation against a similar C implementation, we therefore also compare our solver against OSQP v0.6.0 on QP problems. We selected a number of problem sets to test different aspects of COSMO.

In Sect. 6.1, we first compare the impact of our clique graph-based merging algorithm and other clique tree-based strategies on the projection time of our solver. The clique graph-based merging strategy with COSMO is then compared to MOSEK and SCS on large structured SDPs from the SDPLib benchmark set [10], non-chordal SDPs generated with sparsity patterns from the SuiteSparse Matrix Collections [18], and SDPs with block-arrow shaped sparsity pattern.

In Sect. 6.2, we demonstrate the advantage of directly supporting a quadratic objective function in a conic solver. This is shown first by solving the nearest correlation matrix problem, a problem with both a quadratic objective and a positive semidefinite constraint. Secondly, we show that our conic solver performs well on a classic QP benchmark set, such as the Maros and Mészáros QP repository [38].

Finally, to highlight the advantages of custom constraints, we consider a problem set with doubly stochastic matrices in Sect. 6.3.

All the experiments were carried out on a computing node of the University of Oxford ARC-HTC cluster with 16 logical Intel Xeon E5-2560 cores and 64 GB of DDR3 RAM. All the problems were run using Julia v1.3 and the problems were passed to the solvers via MathOptInterface [37]. To evaluate the accuracy of the

returned solution, we compute three errors adapted from the DIMACS error measures for SDPs [33]:

$$\epsilon_1 = \frac{\|A_a x - b_a\|_2}{1 + \|b_a\|_2}, \quad \epsilon_2 = \frac{\left\|Px + q - A_a^\top y_a\right\|_2}{1 + \|q\|_2}, \quad \epsilon_3 = \frac{\left|x^\top Px + q^\top x - b_a^\top y_a\right|}{1 + \left|q^\top x\right| + \left|b_a^\top y_a\right|}, \tag{19}$$

where A_a, b_a and y_a correspond to the rows of A, b and y that represent active constraints. This is to ensure meaningful values even if the problem contains inactive constraints with very large, or possibly infinite, values b_i. The maximum of the three errors for each problem and solver is reported in the results below. We configured COSMO, MOSEK, SCS and OSQP to achieve an accuracy of $\epsilon = 10^{-3}$. We set the maximum allowable solve time for the Maros and Mészáros problems to 5 min and to 30 min for the other problem sets. Standard values were used for all other solver parameters. COSMO uses a Julia implementation of the QDLDL solver to factor the quasi-definite linear system. Similarly, we configured SCS to use a direct solver for the linear system. The datasets generated and analysed in this section are available from the corresponding author upon request.

6.1 Clique Merging for Decomposable SDPs

In this section, we first compare our proposed clique graph-based merge approach with the clique tree-based strategies of [46,58], all of which were discussed in Sect. 4. We then implement the clique graph-based approach in our solver and compare it against other solvers on three different SDP problem sets.

Comparison of Clique Merging Strategies All three clique merging strategies discussed in Sect. 4 were used to preprocess large sparse SDPs from SDPLib, a collection of SDP benchmark problems [10]. This problem set contains maximum cut problems, SDP relaxations of quadratic programs, and Lovász theta problems. Moreover, we consider a set of test SDPs generated from (non-chordal) sparsity patterns of matrices from the SuiteSparse Matrix Collections [18]. Both problem sets have been used in previous work to benchmark structured SDPs [5,66].

In this section, we consider the effect of different clique decomposition and merging techniques on the per-iteration computation times of COSMO when applied to these benchmark sets.

For the strategy described in [46], we used the SparseCoLO package to decompose the problem. The parent–child method discussed in [58] and the clique graph-based method described in Sect. 4.2 are available as options in COSMO. We further investigate the effect of using different edge weighting functions. The major operation affecting the per-iteration time is the projection step. This step 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 (17). 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 empirically *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. The weighting function is then defined as

$$e(\mathcal{C}_i, \mathcal{C}_j) = t_{\text{proj}}(|\mathcal{C}_i|) + t_{\text{proj}}\left(|\mathcal{C}_j|\right) - t_{\text{proj}}\left(|\mathcal{C}_i \cup \mathcal{C}_j|\right). \tag{20}$$

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`). The SparseCoLO method was used with default parameters. All cases were run single-threaded. Since the per-iteration projection times for some problems lie in the millisecond range every problem was benchmarked ten times and the median values are reported. Table 7 shows the 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 for each problem and strategy. The solve time includes the time spent on decomposition and clique merging. We do not report the total solver time when SparseCoLO was used for the decomposition because this has to be done in a separate preprocessing step in MATLAB which was orders of magnitude slower than the other methods.

Our clique graph-based methods lead to a consistent and substantial reduction in required projection time, and consequently to a reduction in overall solver time. The method with estimated edge weighting function `CG2` achieves the lowest average projection times for the majority of problems. In four cases, `ParCh` has a narrow advantage. The geometric mean of the ratios of projection time of `CG2` compared to the best non-graph method is 0.701, with a minimum ratio of 0.407 for problem `mcp500-2`. There does not seem to be a clear pattern that relates the projection time to the number of cliques or the maximum clique size of the decomposition. This is expected as the optimal merging strategy depends on the properties of the initial decomposition such as the overlap between the cliques. The merging strategies `ParCh`, `CG1` and `CG2` generally result in similar maximum clique sizes compared to `SparseCoLO`, with `CG1` being the most conservative in the number of merges.

Non-chordal Decomposable SDPs After verifying the effectiveness of the clique graph-based merging strategy on the projection step, we implemented the merging strategy `CG2` in COSMO and compared it against MOSEK and SCS. Table 2 shows the benchmark results for the three solvers. The decomposition helps COSMO to solve most problems faster than MOSEK and SCS. This is even more significant for the larger problems that were generated from the SuiteSparse Matrix Collection. The decomposition does not seem to provide a major benefit for the slightly denser problems `mcp500-3` and `mcp500-4`. Furthermore, COSMO seems to converge slowly for `qpG51` and `thetaG51`. Similar observations for `mcp500-3`, `mcp500-4`, and `thetaG51` have been made by the authors in [5]. Finally, many of the larger prob-

Table 2 Benchmark results for non-chordal sparse SDPs from SDPLib and SDPs generated with sparsity patterns from the SuiteSparse Matrix Collection

Problem	Solve time (s)			Iterations			Max error[b]		
	COSMO[a]	MOSEK	SCS	COSMO[a]	Mosek	SCS	COSMO[a]	MOSEK	SCS
maxG11	**1.47**	4.45	131.8	225	5	1220	7.84×10^{-4}	1.98×10^{-3}	4.87×10^{-4}
maxG32	**6.25**	50.84	840.79	375	5	1220	9.74×10^{-4}	2.76×10^{-3}	6.87×10^{-4}
maxG51	**8.09**	9.92	36.56	125	8	180	2.31×10^{-3}	2.83×10^{-4}	8.85×10^{-4}
mcp500-1	**0.24**	1.7	29.28	100	7	580	1.02×10^{-3}	1.52×10^{-3}	6.31×10^{-4}
mcp500-2	**1.68**	1.75	17.36	175	7	380	6.82×10^{-4}	7.37×10^{-4}	3.30×10^{-4}
mcp500-3	4.41	**1.68**	8.36	200	6	180	2.23×10^{-3}	4.18×10^{-4}	6.86×10^{-4}
mcp500-4	8.2	**1.76**	7.4	175	7	160	1.65×10^{-3}	2.79×10^{-4}	3.43×10^{-4}
qpG11	**2.36**	26.23	734.7	300	7	1820	4.57×10^{-4}	1.28×10^{-3}	9.77×10^{-4}
qpG51	121.6	**96.42**	527.55	1825	14	800	7.56×10^{-3}	4.80×10^{-4}	9.83×10^{-4}
thetaG11	**2.32**	8.53	142.53	400	9	1380	1.43×10^{-3}	1.11×10^{-3}	8.31×10^{-5}
thetaG51	71.21	**50.08**	967.43	1250	11	3240	1.38×10^{-1}	4.03×10^{-4}	9.94×10^{-4}
rs1184	**224.86**	***[d]	***[d]	200	***	***	5.65×10^{-4}	***	***
rs1555	**66.6**	***[c]	***[d]	175	***	***	5.32×10^{-4}	***	***
rs1907	**104.61**	***[c]	***[d]	200	***	***	2.66×10^{-4}	***	***
rs200	**12.47**	752.27	***[c]	125	11	***	1.87×10^{-4}	6.11×10^{-4}	***
rs228	**12.86**	395.24	982.5	125	11	1620	2.15×10^{-4}	6.34×10^{-4}	8.27×10^{-4}
rs35	**54.88**	919.19	***[c]	150	12	***	2.48×10^{-4}	3.26×10^{-4}	***
rs365	**62.65**	***[c]	***[c]	175	***	***	3.08×10^{-4}	***	***
rs828	**10.84**	825.03	***[c]	125	11	***	1.98×10^{-4}	8.48×10^{-4}	***

The bold values signifies the lowest solve time (lowest number) among the solvers [a]With chordal decomposition and clique merging strategy CG2
[b] $\max\{\epsilon_1, \epsilon_2, \epsilon_3\}$
[c]Time limit reached
[d]Out of memory error

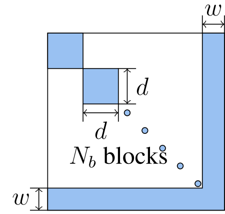

Fig. 4 Parameters of block-arrow sparsity pattern. The shaded area represents the nonzeros of the sparsity pattern

lems were not solvable within the time limit or caused out-of-memory problems if no decomposition was used in MOSEK and SCS.

Block-Arrow Sparse SDPs To demonstrate the scaling behaviour of our solver on decomposable SDPs with increasing dimension, we consider randomly generated SDPs in standard dual form (12) with a block-arrow aggregate sparsity pattern similar to test problems in [5,66]. Figure 4 shows the sparsity pattern of the PSD constraint. The sparsity pattern is generated based on the following parameters: block size d, number of blocks N_b and width of the arrow head w. Note that the graph corresponding to the sparsity pattern is always chordal and that, for this sparsity pattern, clique merging yields no benefit. In the following, we study the effects of independently increasing the block size d and the number of blocks N_b. The solve times for all solvers are shown in Table 3. The first part shows problems with increasing block size d and fixed number of blocks $N_b = 50$, arrowhead width $w = 10$ and number of constraints $m = 100$. In the second part, the number of blocks N_b is varied for fixed $d = 10$, $w = 10$, $m = 100$. The results show that both COSMO with and without chordal decomposition solve this problem type consistently faster than MOSEK and SCS. COSMO performs better than the other first-order solver (SCS) because it requires a the significantly smaller number of iterations. Furthermore, one can see that in both cases the solve time for each solver rises when the number of blocks and the block sizes are increased. The increase is smaller for COSMO(CD) which is more affected by the number of iterations than the problem dimension.

6.2 Advantages of Supporting a Quadratic Objective

COSMO's problem format (2) allows us to handle QPs and SDPs with quadratic objectives, without the need to transform the quadratic term into an additional second-order cone constraint. This avoids an additional matrix factorisation, preserves sparsity in the objective, and in many cases achieves faster convergence due to the strict convexity of the objective. To demonstrate these advantages, we solve the nearest correlation matrix problem and problems from the Maros and Mészáros QP test set.

Nearest Correlation Matrix Consider the problem of projecting a matrix C onto the set of correlation matrices, i.e. real symmetric positive semidefinite matrices with diagonal elements equal to 1. This problem is relevant in portfolio optimisation [32]. The correlation matrix of a stock portfolio might lose its positive semidefiniteness due to noise and rounding errors of previous data manipulations. Consequently, we are

Table 3 Benchmark results for block arrow sparse SDPs with varying number of blocks N_b and block size d

d	Solve time (s)				Iterations				Max error[b]			
	COSMO	COSMO[a]	MOSEK	SCS	COSMO	COSMO[a]	MOSEK	SCS	COSMO	COSMO[a]	MOSEK	SCS
10	12.71	**1.22**	16.5	56.99	125	175	9	1340	1.50×10^{-4}	5.60×10^{-4}	1.16×10^{-3}	9.01×10^{-4}
12	23.95	**2.3**	28.72	68.72	125	225	10	1120	2.29×10^{-4}	7.52×10^{-4}	3.65×10^{-5}	5.38×10^{-4}
14	27.4	**2.39**	41.66	147.72	150	175	10	1780	7.92×10^{-5}	8.60×10^{-4}	6.84×10^{-4}	5.46×10^{-4}
16	36.4	**3.6**	58.04	254.93	150	275	10	2360	4.76×10^{-5}	5.34×10^{-4}	2.64×10^{-4}	6.91×10^{-4}
18	42.96	**4.42**	85.81	472.77	150	275	11	3440	2.66×10^{-4}	5.77×10^{-4}	2.55×10^{-4}	3.25×10^{-4}
20	41.13	**5.08**	112.56	251.39	125	275	11	1460	2.61×10^{-4}	5.35×10^{-4}	6.33×10^{-5}	7.26×10^{-4}
22	50.7	**11.47**	142.14	520.81	125	700	11	2460	2.87×10^{-4}	5.85×10^{-4}	3.31×10^{-4}	4.86×10^{-4}
24	125.12	**10.52**	178.75	445.37	275	500	11	1780	7.46×10^{-5}	7.23×10^{-4}	3.57×10^{-4}	5.10×10^{-4}
26	88.74	**6.84**	220.26	623.46	150	200	11	2060	8.17×10^{-5}	3.27×10^{-4}	2.89×10^{-6}	1.39×10^{-4}
28	98.83	**7.47**	263.52	1626.82	150	200	11	4660	3.73×10^{-5}	8.93×10^{-4}	2.62×10^{-5}	4.59×10^{-4}
N_b	COSMO	COSMO[a]	MOSEK	SCS	COSMO	COSMO[a]	MOSEK	SCS	COSMO	COSMO[a]	MOSEK	SCS
50	17.31	**3.73**	28.69	115.36	150	300	10	2540	1.33×10^{-4}	6.28×10^{-4}	2.21×10^{-4}	9.12×10^{-4}
60	21.66	**6.4**	46.38	217.75	150	475	11	3380	1.42×10^{-4}	8.74×10^{-4}	7.27×10^{-5}	8.81×10^{-4}
70	34.51	**9.89**	64.8	177.24	175	700	11	2060	4.12×10^{-5}	6.14×10^{-4}	1.72×10^{-5}	8.99×10^{-4}
80	37.22	**5.78**	85.29	175.38	175	275	11	1540	5.15×10^{-5}	7.13×10^{-4}	2.08×10^{-4}	4.57×10^{-4}
90	63.32	**12.85**	113.24	234.56	225	550	11	1580	4.80×10^{-5}	1.12×10^{-3}	4.79×10^{-5}	6.40×10^{-4}
100	67.3	**25.16**	140.81	328.2	200	1100	11	1800	6.38×10^{-5}	9.54×10^{-4}	3.28×10^{-5}	8.59×10^{-4}
110	117.7	**39.86**	172.99	761.22	275	1650	11	3140	3.78×10^{-5}	6.80×10^{-4}	1.35×10^{-4}	3.53×10^{-4}
120	81.31	**29.06**	209.96	1191.64	150	1050	11	4620	3.59×10^{-4}	1.07×10^{-3}	3.66×10^{-4}	2.71×10^{-4}
130	115.21	**26.61**	269.06	1094.53	175	925	12	3580	8.48×10^{-5}	1.31×10^{-3}	8.59×10^{-5}	9.67×10^{-4}
140	114.95	**26.87**	290.23	***[c]	150	800	11	***	1.20×10^{-4}	1.92×10^{-3}	4.99×10^{-4}	***

The bold values signifies the lowest solve time (lowest number) among the solvers [a]With chordal decomposition
[b] $\max\{\epsilon_1, \epsilon_2, \epsilon_3\}$
[c]Time limit reached

interested to find the nearest correlation matrix X to a given data matrix $C \in \mathbb{R}^{n \times n}$:

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}\|X - C\|_F^2 \\ \text{subject to} & \operatorname{diag}(X) = \mathbf{1}_n,\ X \in \mathbb{S}_+^n. \end{array} \tag{21}$$

In order to transform the problem into the standard form (2) used by COSMO, C and X are vectorised and the objective function is expanded:

$$\begin{array}{ll} \text{minimise} & (1/2)(x^\top x - 2c^\top x + c^\top c) \\ \text{subject to} & \begin{bmatrix} E \\ -I \end{bmatrix} x + s = \begin{bmatrix} \mathbf{1}_n \\ \mathbf{0}_{n^2} \end{bmatrix}, \qquad s \in \{0\}^n \times \mathcal{S}_+^n, \end{array} \tag{22}$$

with $c = \operatorname{vec}(C) \in \mathbb{R}^{n^2}$ and $x = \operatorname{vec}(X) \in \mathbb{R}^{n^2}$. Here $E \in \mathbb{R}^{n \times n^2}$ is a matrix that extracts the n diagonal entries X_{ii} from its vectorised form x. For the benchmark problems, we randomly sample the data matrix C with entries $C_{i,j} \sim \mathcal{U}(-1, 1)$ from a uniform distribution. Table 4 shows the benchmark results for increasing matrix dimension n.

Unsurprisingly, the first-order methods SCS and COSMO outperform the interior-point solver MOSEK for these large SDPs. Furthermore, for larger problems the solve times of COSMO and SCS scale in a similar way. However, SCS has to transform the objective into an additional second-order cone constraint, which increases the factorisation- and projection time compared to COSMO.

Maros and Mészáros QP Test Set The Maros and Mészáros test problem set [38] is a repository of challenging convex QP problems that is widely used to compare the performance of QP solvers. For comparison metrics we compute the failure rate, the number of fastest solve times and the normalised shifted geometric mean for each solver. The shifted geometric mean is more robust against large outliers (compared to the arithmetic mean) and against small outliers (compared to the geometric mean) and is commonly used in optimisation benchmarks; see [43,56]. The shifted geometric mean $\mu_{g,s}$ is defined as $\mu_{g,s} = \sqrt[n]{\prod_p (t_{p,s} + \text{sh})} - \text{sh}$ with total solver time $t_{p,s}$ of solver s and problem p, shifting factor sh and size of the problem set n. In the reported results a shifting factor of $sh = 10$ was chosen and the maximum allowable time $t_{p,s} = 300$ s was used if solver s failed on problem p. Lastly, we normalise the shifted geometric mean for solver s by dividing by the geometric mean of the fastest solver. The failure rate $f_{r,s}$ is given by the number of unsolved problems compared to the total number of problems in the problem set. As unsolved problems, we count instances where the algorithm does not converge within the allowable time or fails during the setup or solve phase. Table 5 shows the normalised shifted geometric mean and the failure rate for each solver. Additionally, the number of cases where solver s was the fastest solver is shown.

OSQP shows the best performance in terms of lowest failure rates, number of fastest solves and in the shifted geometric mean of solve times. COSMO follows very closely. The shifted geometric mean of MOSEK seems to suffer from a higher failure rate compared to OSQP/COSMO, and SCS fails on a large number of problems. The higher failure rate is likely to be due to the necessary transformation into a second-order

Table 4 Benchmark results for nearest correlation matrix problems

n	Solve time (s)			Iterations			Max error[a]		
	COSMO	MOSEK	SCS	COSMO	MOSEK	SCS	COSMO	MOSEK	SCS
50	0.5	1.89	**0.07**	25	4	20	2.05×10^{-5}	1.35×10^{-3}	6.18×10^{-7}
100	0.45	35.68	**0.18**	25	4	20	2.02×10^{-5}	6.26×10^{-3}	4.24×10^{-6}
150	**0.39**	244.35	0.46	25	4	20	5.15×10^{-5}	8.61×10^{-3}	9.00×10^{-6}
200	0.94	1032.25	**0.74**	25	4	20	7.59×10^{-5}	3.64×10^{-3}	8.41×10^{-5}
400	**1.14**	* * *[b]	3.07	25	* * *	20	2.62×10^{-4}	* * *	8.52×10^{-5}
600	**3.05**	* * *[b]	7.38	25	* * *	20	1.53×10^{-4}	* * *	1.01×10^{-4}
800	**5.08**	* * *[b]	13.85	25	* * *	20	4.03×10^{-4}	* * *	1.07×10^{-4}
1000	**8.43**	* * *[b]	21.94	25	* * *	20	8.11×10^{-4}	* * *	1.40×10^{-4}
1200	**12.1**	* * *[b]	34.58	25	* * *	20	1.02×10^{-3}	* * *	1.73×10^{-4}
1400	**17.19**	* * *[b]	49.23	25	* * *	20	1.02×10^{-3}	* * *	1.93×10^{-4}
1600	**22.69**	* * *[b]	61.07	25	* * *	20	8.00×10^{-4}	* * *	2.21×10^{-4}
1800	**29.41**	* * *[b]	74.0	25	* * *	20	5.14×10^{-4}	* * *	2.43×10^{-4}
2000	**37.74**	* * *[b]	101.18	25	* * *	20	3.31×10^{-4}	* * *	2.57×10^{-4}

The bold values signifies the lowest solve time (lowest number) among the solvers [a] $\max\{\epsilon_1, \epsilon_2, \epsilon_3\}$
[b]Time limit reached

Table 5 Shifted geometric mean and solver failure rates for the Maros and Mészáros QP test set

	OSQP	COSMO	MOSEK	SCS
Normalised shifted geometric mean	1.000	1.169	1.897	75.015
Number of fastest solve times	75	27	33	0
Failure rates $f_{r,s}$ (%)	4.444	5.185	10.370	83.704

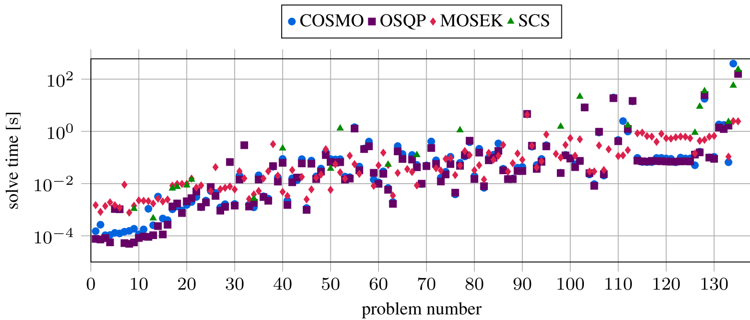

Fig. 5 Solve time of benchmarked solvers for problems of the Maros and Mészáros QP problem set. Only problem results classified as solved are shown. The problems are ordered by increasing number of nonzeros in the constraint matrix

cone problem. For this problem set of QPs, COSMO's algorithm reduces, with some minor differences, to the algorithm of OSQP. Consequently, this benchmark is useful to evaluate the performance penalty incurred by COSMO due to its implementation in the higher-order language Julia. The results in Table 5 show that the performance difference is very small. This is confirmed by the solve times for increasing problem dimension, shown in Fig. 5.

COSMO achieves lower solve times than both MOSEK and SCS especially for very small and very large problems. COSMO and OSQP have nearly identical performance, aside from very small problems that are solved in the range of 1×10^{-5} s to 1×10^{-4} s. The marginally better performance of OSQP for the smallest problems in the test set is the reason that OSQP is the faster solver in a larger number of cases in Table 5. This difference is primarily due to overheads incurred from features in our Julia implementation that support more than one constraint type during problem setup. We demonstrate the resulting extensibility and customisability of our solver by solving QPs with custom constraints in Sect. 6.3.

6.3 Custom Convex Cones

In many cases, writing a custom solver algorithm for a particular problem can be faster than using available solver packages if a particular aspect of the problem structure can be exploited to speed up parts of the computations. As mentioned earlier, COSMO supports user customisation by allowing the definition of new convex cones. This is useful if constraints of the problem can be expressed using this new convex cone and

a fast projection method onto the cone exists. A fast specialised projection method in an ADMM framework has for example been used by the authors in [8] to solve the error-correcting code decoding problem.

To demonstrate the advantage of custom convex cones, consider the problem of finding the doubly stochastic matrix that is nearest, in the Frobenius norm, to a given symmetric matrix $C \in \mathbb{S}^n$. Doubly stochastic matrices are used for instance in spectral clustering [65] and matrix balancing [50]. A specialised algorithm for this problem type is discussed in [54]. Doubly stochastic matrices have the property that all rows and columns each sum to one and all entries are nonnegative. The nearest doubly stochastic matrix X can be found by solving the following optimisation problem:

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}\left\|X - C\right\|_F^2 \\ \text{subject to} & X_{ij} \geq 0,\ X\mathbf{1} = \mathbf{1},\ X^\top \mathbf{1} = \mathbf{1}, \end{array} \tag{23}$$

with symmetric real matrix $C \in \mathbb{S}^n$ and decision variable $X \in \mathbb{R}^{n\times n}$. This problem can be solved as a QP in the following form using equality and inequality constraints:

$$\begin{array}{ll} \text{minimise} & \frac{1}{2}(x^\top x - 2c^\top x + c^\top c) \\ \text{subject to} & \begin{bmatrix} \mathbf{1}_n^\top \otimes I_n \\ I_n \otimes \mathbf{1}_n^\top \\ -I_{n^2} \end{bmatrix} x + s = \begin{bmatrix} \mathbf{1}_{2n} \\ \mathbf{1}_{2n} \\ \mathbf{0}_{n^2} \end{bmatrix},\ s \in \{0\}^{4n} \times \mathbb{R}_+^{n^2}, \end{array} \tag{24}$$

with $x = \operatorname{vec}(X)$ and $c = \operatorname{vec}(C)$ However, the problem can be written in a more compact form by using a custom projection function to project the matrix iterate onto the affine set of matrices $\mathcal{C}_\sum$, whose rows and columns each sum to one. In general the projection of vector $s \in \mathbb{R}^n$ onto the affine set $\mathcal{C}_a = \{s \in \mathbb{R}^n \mid As = b\}$ is given by $\Pi_{\mathcal{C}_a}(s) = s - A^\top \left(AA^\top\right)^{-1}(As - b)$, where A is assumed to have full rank. In the case of $\mathcal{C}_a = \mathcal{C}_\sum$ we can exploit the fact that the inverse of $AA^\top$ can be efficiently computed. The projection can be carried out as described in Algorithm 4; see Appendix A.1 for a derivation.

Algorithm 4: Projection of $s \in \mathbb{R}^n$ onto $\mathcal{C}_\sum$

1. $A = \begin{bmatrix} A_r & A_c \end{bmatrix}^\top$ with $A_r = \mathbf{1}_n^\top \otimes I_n$, $\quad A_c = \begin{bmatrix} I_{n-1} \otimes \mathbf{1}_n^\top & \mathbf{0}_{n-1\times n} \end{bmatrix}$;
2. $r = [r_1, r_2]^\top = As - \mathbf{1}_{2n-1}$;
3. $\eta = [\eta_1, \eta_2]^\top$ with $\eta_2 = \frac{1}{n}\left(I_{n-1} + \mathbf{1}_{n-1}\mathbf{1}_{n-1}^\top\right) \cdot \left(r_2 - \frac{1}{n}\mathbf{1}_{n-1}\mathbf{1}_n^\top r_1\right)$, $\eta_1 = \frac{1}{n}\left(r_1 - \mathbf{1}_n \mathbf{1}_{n-1}^\top \eta_2\right)$;
4. $\Pi_{\mathcal{C}_\sum}(s) = s - A^\top \eta$;

Notice that Algorithm 4 can be implemented efficiently without ever assembling and storing A and $\mathbf{1}\mathbf{1}^\top$. By using the custom convex set $\mathcal{C}_\sum$ and the corresponding projection function, (23) can now be rewritten as:

$$\begin{array}{ll} \text{minimise} & (1/2)(x^\top x - 2c^\top x + c^\top c) \\ \text{subject to} & \begin{bmatrix} -I_{n^2} \\ -I_{n^2} \end{bmatrix} x + s = \mathbf{0}_{2n^2}, \qquad s \in \mathcal{C}_\sum \times \mathbb{R}_+^{n^2}. \end{array} \tag{25}$$

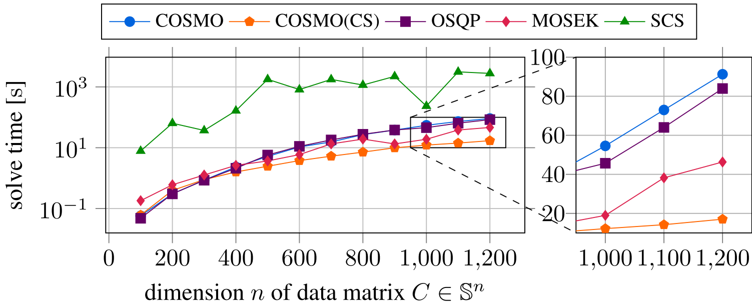

Fig. 6 Solve time of benchmarked solvers for increasing problem size of doubly stochastic matrix problems. The orange line shows the solve time of COSMO(CS) with a custom convex set and projection function

Table 6 Solve times and factorisation times of COSMO and COSMO(CS) for small, medium and large doubly stochastic matrix problems

n	Factorisation time (s)		Solve time (s)	
	COSMO	COSMO(CS)[a]	COSMO	COSMO(CS)[a]
100	0.018	0.006	0.058	0.061
400	0.995	0.138	2.329	1.596
800	13.551	0.552	26.932	7.163
1200	52.717	1.322	1.322	17.032

[a] Solving (25) with a custom convex set $\mathcal{C}_{\sum}$ and projection function

The sparsity pattern of the new constraint matrix A only consists of two diagonals and the number of nonzeros reduces from $3n^2$ to $2n^2$. We expect this to reduce the initial factorisation time of the linear system in (6) as well as the forward- and back-substitution steps.

Figure 6 shows the total solve time of all the solvers for problem (23) with randomly generated dense matrix C with $C_{ij} \sim \mathcal{U}(0, 1)$ and increasing matrix dimension. Additionally, we show the solve time for COSMO in the problem form (23) and with a specialised custom set and projection function as in (25).

It is not surprising that COSMO and OSQP scale in the same way for this problem type. MOSEK is slightly slower for smaller problem dimensions and overtakes COSMO/OSQP for problems of dimensions $n \geq 500$. This is probably due to the use in MOSEK of a faster multithreaded linear system solver while OSQP/COSMO relies on the single-threaded solver QDLDL. The longer solve time of SCS is due to slow convergence of the algorithm for this problem type. Furthermore, when the problem is solved with a custom convex set as in (25) COSMO(CS) is able to outperform all other solvers. Table 6 shows the total solve time and the factorisation time of the two versions of COSMO for small, medium and large problems. As predicted, the lower solve time can be mainly attributed to the faster factorisation time.

7 Conclusions

This paper describes the first-order solver COSMO which combines direct support of quadratic objectives, infeasibility detection, custom constraints, chordal decomposition of PSD constraints and automatic clique merging. The performance of the solver is illustrated on a number of benchmark problems that challenge different aspects of modern solvers. WE show how our clique graph-based merging strategy can achieve up to 50% time reduction in the projection step when solving sparse SDPs. We further show how multithreading and chordal decomposition can be utilised to solve very large SDPs that arise in real applications within minutes where competing solvers run out of memory or need hours to find a solution. Our implementation in the high-level Julia language facilitates rapid development and testing of ideas and allows users to customise the solver for their applications. We illustrate the utility of these features by demonstrating a 5x reduction in solve time when using a custom constraint projection for large QPs. It further allows the abstraction of precision and array types which in principle make it possible for COSMO to run on GPUs. Further performance gains are likely to be achieved by exploring acceleration methods to speed up convergence to higher accuracies and reduce the dependency on problem scaling.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

A Appendix

A.1 Projection onto $\mathcal{C}_{\sum}$

The projection of a symmetric matrix $S \in \mathbb{S}^n$ onto the set of matrices where the sum of rows and columns each equal to one can be written as the following optimisation problem:

$$\text{minimise } \tfrac{1}{2} \left\| s_p - s \right\|_F^2 \text{ subject to } A s_p = \mathbf{1}_{2n-1} \tag{26}$$

where $s = \text{vec}(S)$ is the vectorised matrix, s_p is the projected vector and A is partitioned $A = [A_r, A_c]^\top$. $A_r = [\mathbf{1}_n^\top \otimes I_n]$ is used to constrain the rows of S and $A_c = [I_{n-1} \otimes \mathbf{1}_n^\top, \mathbf{0}_{n-1 \times n}]$ is used to constrain the columns of S. Notice that A_c is a $(n-1) \times n$ matrix because the redundant constraint on the last column was removed. The KKT conditions for (26) are given by:

$$A s_p = \mathbf{1}_{2n-1} \text{ and } s_p - s + A^\top \eta = 0, \tag{27}$$

with dual variable η. Eliminating s_p and solving for η yields $\eta = (AA^\top)^{-1}(As - \mathbf{1}_{2n-1})$. The projected vector s_p can be recovered from (27):

$$s_p = s - A^\top \eta. \tag{28}$$

It turns out that the inverse of $AA^\top$ can be efficiently computed without a factorisation. To see this, write the equation for η as a linear system:

$$\begin{bmatrix} nI_n & \mathbf{1}_n \mathbf{1}_{n-1}^\top \\ \mathbf{1}_{n-1}\mathbf{1}_n^\top & nI_{n-1} \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \text{ with } \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = As - \mathbf{1}_{2n-1}, \tag{29}$$

where η and the right-hand side were partitioned in such a way that $\eta_1, r_1 \in \mathbb{R}^n$ and $\eta_2, r_2 \in \mathbb{R}^{n-1}$. By eliminating η_1 from (29) and applying the Sherman–Morrison formula to compute the matrix inverse we compute η_2

$$\left(I_{n-1} - \tfrac{1}{n}\mathbf{1}_{n-1}\mathbf{1}_{n-1}^\top\right)\eta_2 = \tfrac{1}{n}\left(r_2 - \tfrac{1}{n}\mathbf{1}_{n-1}\mathbf{1}_n^\top r_1\right) \tag{30}$$

$$\eta_2 = \tfrac{1}{n}\left(I_{n-1} + \mathbf{1}_{n-1}\mathbf{1}_{n-1}^\top\right)\left(r_2 - \tfrac{1}{n}\mathbf{1}_{n-1}\mathbf{1}_n^\top r_1\right). \tag{31}$$

By substituting η_2 into the upper equation of (29), we compute $\eta_1 = \frac{1}{n}(r_1 - \mathbf{1}_n \mathbf{1}_{n-1}^\top \eta_2)$. Having computed the dual variable η, the projected vector s_p is obtained by solving (28).

A.2 Benchmark Results

See Table 7.

Table 7 Benchmark results for non-chordal sparse SDPs from SDPLib and SDPs generated with sparsity patterns from the SuiteSparse Matrix Collection for different merging strategies

Problem	Median solve time (s)						Median projection time (ms)					
	NoDe[c]	NoMer[d]	SpCo[e]	ParCh[f]	CG1[g]	CG2[h]	NoDe	NoMer	SpCo	ParCh	CG1	CG2
maxG11	35.85	4.69	* * *	3.33	3.09	**2.93**	143.3	18.4	12.7	**10.5**	13.4	11.2
maxG32	407.82	25.95	* * *	15.62	**14.38**	19.56	1257.7	73.2	73.1	49.9	51.6	**43.1**
maxG51	40.61	32.8	* * *	9.56	**6.93**	9.62	245.8	230.4	219.3	122.2	65.5	**58.5**
mcp500-1	21.46	1.33	* * *	0.7	0.89	**0.6**	56.0	7.7	7.4	**4.6**	6.1	4.9
mcp500-2	13.55	12.28	* * *	8.37	2.6	**2.42**	54.9	45.4	32.1	28.0	14.3	**11.4**
mcp500-3	11.28	32.45	* * *	30.13	7.01	**5.14**	51.1	104.9	105.0	83.4	28.2	**21.2**
mcp500-4	14.57	72.27	* * *	13.87	**5.84**	7.67	59.2	253.8	193.8	141.2	44.1	**35.8**
qpG11	142.31	7.3	* * *	4.79	**4.62**	4.7	305.1	20.0	12.3	**10.9**	15.6	13.2
qpG51	450.74	186.49	* * *	**89.81**	150.86	132.68	523.6	247.2	246.9	134.1	73.9	**65.2**
thetaG11	332.06	9.43	* * *	9.43	9.46	**6.81**	477.5	21.9	18.9	**12.5**	16.3	14.5
thetaG51	1062.91	110.64	* * *	107.02	**37.22**	85.92	252.8	230.8	* * *[a]	131.7	66.5	**53.7**
rs1184	* * *[b]	1227.48	* * *	882.27	632.96	**569.29**	* * *[b]	4192.8	3495.8	3424.1	2483.9	**2301.3**
rs1555	* * *[a]	79.83	* * *	**65.93**	80.72	83.84	* * *[a]	316.7	242.7	268.8	160.7	**132.1**
rs1907	* * *[a]	233.86	* * *	197.99	178.79	**166.23**	* * *[a]	483.8	490.3	455.3	382.7	**352.1**
rs200	640.0	31.33	* * *	21.09	24.78	**19.29**	3366.2	121.6	93.0	82.9	93.2	**71.7**
rs228	206.2	40.88	* * *	27.79	25.29	**18.6**	1220.2	116.0	59.4	76.2	67.1	**50.9**
rs35	296.52	196.93	* * *	146.56	88.86	**71.25**	1269.8	548.1	358.2	404.6	272.5	**223.2**
rs365	* * *[a]	159.75	* * *	127.77	110.48	**92.5**	* * *[a]	433.1	364.6	351.1	289.9	**262.0**
rs828	603.55	29.86	* * *	19.24	23.25	**17.81**	3716.7	113.2	80.0	71.1	87.5	**64.2**

Table 7 continued

Problem	Median solve time (s)						Median projection time (ms)					
	NoDe[c]	NoMer[d]	SpCo[e]	ParCh[f]	CG1[g]	CG2[h]	NoDe	NoMer	SpCo	ParCh	CG1	CG2
maxG11	225	225	500	275	200	225	1/800	598/24	13/80	198/32	473/28	407/36
maxG32	300	300	425	250	225	375	1/2000	1498/76	21/210	481/76	1164/92	664/102
maxG51	150	100	75	50	75	125	1/1000	674/326	181/322	163/326	448/362	313/395
mcp500-1	350	150	150	125	125	100	1/500	457/39	451/44	105/43	437/54	361/65
mcp500-2	225	225	200	250	150	175	1/500	363/138	144/138	96/140	316/156	226/171
mcp500-3	200	250	250	300	200	200	1/500	259/242	101/242	71/242	211/263	135/285
mcp500-4	225	225	375	75	100	175	1/500	161/340	63/346	46/341	105/368	87/393
qpG11	400	325	525	375	250	300	1/1600	1398/24	813/80	287/32	1273/28	1207/36
qpG51	750	600	800	550	1800	1825	1/2000	1674/326	1182/304	275/326	1448/362	1313/395
thetaG11	675	375	2275	650	500	400	1/801	598/25	13/81	198/33	494/29	423/41
thetaG51	3825	325	* * *[a]	575	375	1250	1/1001	676/324	150/323	157/324	424/358	267/396
rs1184	* * *[b]	225	200	200	200	200	1/14822	2236/500	78/1330	1043/500	664/608	258/632
rs1555	* * *[a]	150	150	150	150	175	1/7479	6891/184	3350/187	2556/184	5529/236	4858/276
rs1907	* * *[a]	200	200	175	175	200	1/5357	577/261	47/585	419/261	441/324	219/405
rs200	175	125	125	125	125	125	1/3025	1632/95	94/216	444/95	1123/112	583/119
rs228	150	125	125	125	125	125	1/1919	790/88	48/180	255/88	369/95	129/127
rs35	200	175	200	175	150	150	1/2003	589/343	53/735	189/343	214/457	106/520
rs365	* * *[a]	175	175	175	175	175	1/4704	1230/296	110/350	539/296	688/349	346/474
rs828	150	125	150	125	125	125	1/3169	1875/86	112/174	501/86	1378/102	708/126

The bold values signifies the lowest solve time (lowest number) among the solvers [a]Time limit reached
[b]Out of memory error
[c]No decomposition
[d]No merging
[e]SparseCoLO merging
[f]Parent–child merging
[g]Clique graph with nominal edge weighting (17)
[h]Clique graph with estimated edge weighting (20)

References

1. Agler, J., Helton, W., McCullough, S., Rodman, L.: Positive semidefinite matrices with a given sparsity pattern. Linear Algebra Appl. **107**, 101–149 (1988)
2. Alizadeh, F.: Interior point methods in semidefinite programming with applications to combinatorial optimization. SIAM J. Optim. **5**(1), 13–51 (1995)
3. Alizadeh, F., Haeberly, J., Overton, M.L.: Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results. SIAM J. Optim. **8**(3), 746–768 (1998)
4. Andersen, M.S., Vandenberghe, L.: CHOMPACK: a python package for chordal matrix computations (2015)
5. Andersen, M.S., Dahl, J., Vandenberghe, L.: Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones. Math. Program. Comput. **2**(3–4), 167–201 (2010)
6. Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A.: LAPACK Users' Guide. SIAM, Philadelphia (1999)
7. Banjac, G., Goulart, P., Stellato, B., Boyd, S.: Infeasibility detection in the alternating direction method of multipliers for convex optimization. J. Optim. Theory Appl. **183**(2), 490–519 (2019)
8. Barman, S., Liu, X., Draper, S.C., Recht, B.: Decomposition methods for large scale LP decoding. IEEE Trans. Inf. Theory **59**(12), 7870–7886 (2013)
9. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: a fresh approach to numerical computing. SIAM Rev. **59**(1), 65–98 (2017)
10. Borchers, B.: SDPLIB 1.2, a library of semidefinite programming test problems. Optim. Methods Softw. **11**(1–4), 683–690 (1999)
11. Borm, P., Hamers, H., Hendrickx, R.: Operations research games: a survey. Top **9**(2), 139 (2001)
12. Boyd, S., El Ghaoui, L., Feron, E., Balakrishnan, V.: Linear Matrix Inequalities in System and Control Theory, vol. 15. SIAM, Philadelphia (1994)
13. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
14. Boyd, S., Busseti, E., Diamond, S., Kahn, R.N., Koh, K., Nystrup, P., Speth, J.: Multi-period trading via convex optimization. Found. Trends Optim. **3**(1), 1–76 (2017)
15. Coleman, T.F., Li, Y.: Compact clique tree data structures in sparse matrix factorizations. In: Large-scale Numerical Optimization. SIAM (1990)
16. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
17. Davis, T.A.: Suitesparse: a suite of sparse matrix software. http://faculty.cse.tamu.edu/davis/suitesparse.html (2015)
18. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. (TOMS) **38**(1), 1–25 (2011)
19. Dunning, I., Huchette, J., Lubin, M.: JuMP: a modeling language for mathematical optimization. SIAM Rev. **59**(2), 295–320 (2017)
20. Eckstein, J.: Splitting methods for monotone operators with applications to parallel optimization. Ph.D. thesis, Massachusetts Institute of Technology (1989)
21. Everett, H.: Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. Oper. Res. **11**(3), 399–417 (1963)
22. Fujisawa, K., Kim, S., Kojima, M., Okamoto, Y., Yamashita, M.: User's manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems. Research Report B-453, Dept. of Math. and Comp. Sci. Japan, Tech. Rep., pp. 152–8552 (2009)
23. Fujisawa, K., Fukuda, M., Nakata, K.: Preprocessing sparse semidefinite programs via matrix completion. Optim. Methods Softw. **21**(1), 17–39 (2006)
24. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting sparsity in semidefinite programming via matrix completion I: general framework. SIAM J. Optim. **11**(3), 647–674 (2001)
25. Gabay, D., Mercier, B.: A dual algorithm for the solution of non linear variational problems via finite element approximation. Institut de recherche d'informatique et d'automatique (1975)
26. Garstka, M., Cannon, M., Goulart, P.: COSMO: a conic operator splitting method for large convex problems. In: 18th European Control Conference (ECC) (Naples, Italy), pp. 1951–1956 (2019)
27. Garstka, M., Cannon, M., Goulart, P.: A clique graph based merging strategy for decomposable SDPs. IFAC-PapersOnLine **53**(2), 7355–7361 (2020). 21th IFAC World Congress
28. Grone, R., Johnson, C.R., Sá, E.M., Wolkowicz, H.: Positive definite completions of partial Hermitian matrices. Linear Algebra Appl. **58**, 109–124 (1984)

29. Habib, M., Stacho, J.: Reduced clique graphs of chordal graphs. Eur. J. Comb. **33**(5), 712–735 (2012)
30. Helmberg, C., Rendl, F., Vanderbei, R.J., Wolkowicz, H.: An interior-point method for semidefinite programming. SIAM J. Optim. **6**(2), 342–361 (1996)
31. Hestenes, M.R.: Multiplier and gradient methods. J. Optim. Theory Appl. **4**(5), 303–320 (1969)
32. Higham, N.J.: Computing the nearest correlation matrix-a problem from finance. IMA J. Numer. Anal. **22**(3), 329–343 (2002)
33. Johnson, D., Pataki, G., Alizadeh, F.: Seventh dimacs implementation challenge: semidefinite and related problems (2000). https://dimacs.rutgers.edu/Challenges/Seventh
34. Kalinkin, A., Anders, A., Anders, R.: Intel Math Kernel Library PARDISO* for Intel Xeon Phi many-core coprocessor. Appl. Math. **6**(08), 1276 (2015)
35. Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: Proceedings of the 16th Annual ACM Symposium on Theory of Computing, pp. 302–311. ACM (1984)
36. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**(1), 48–50 (1956)
37. Legat, B., Dowson, O., Garcia, J.D., Lubin, M.: MathOptInterface: a data structure for mathematical optimization problems (2020). arXiv: 2002.03447 [math.OC]
38. Maros, I., Mészáros, C.: A repository of convex quadratic programming problems. Optim. Methods Softw. **11**(1–4), 671–681 (1999)
39. Mehrotra, S.: On the implementation of a primal-dual interior point method. SIAM J. Optim. **2**(4), 575–601 (1992)
40. Miele, A., Cragg, E.E., Iyer, R.R., Levy, A.V.: Use of the augmented penalty function in mathematical programming problems, part 1. J. Optim. Theory Appl. **8**(2), 115–130 (1971)
41. Miele, A., Cragg, E.E., Levy, A.V.: Use of the augmented penalty function in mathematical programming problems, part 2. J. Optim. Theory Appl. **8**(2), 131–153 (1971)
42. Miele, A., Moseley, P.E., Levy, A.V., Coggins, G.M.: On the method of multipliers for mathematical programming problems. J. Optim. Theory Appl. **10**(1), 1–33 (1972)
43. Mittelmann, H.: Benchmarks for optimization software (2020). http://plato.asu.edu/bench.html. Accessed 22 June 2020
44. Molzahn, D.K., Holzer, J.T., Lesieutre, B.C., DeMarco, C.L.: Implementation of a large-scale optimal power flow solver based on semidefinite programming. IEEE Trans. Power Syst. **28**(4), 3987–3998 (2013)
45. MOSEK, Aps. The MOSEK optimization toolbox for MATLAB manual. Version 8.0 (2017)
46. Nakata, K., Fujisawa, K., Fukuda, M., Kojima, M., Murota, K.: Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical results. Math. Program. **95**(2), 303–327 (2003)
47. O'Donoghue, B., Chu, E., Parikh, N., Boyd, S.: Conic optimization via operator splitting and homogeneous self-dual embedding. J. Optim. Theory Appl. **169**, 1042–1068 (2016)
48. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends Optim. **1**(3), 127–239 (2014)
49. Powell, M.J.D.: A method for nonlinear constraints in minimization problems. In: Fletcher, R. (ed.) Optimization, pp. 283–298. Academic Press, London, New York (1969)
50. Rao, S., Huntley, M.H., Durand, N.C., Stamenova, E.K., Bochkov, I.D., Robinson, J.T., Sanborn, A.L., Machol, I., Omer, A.D., Lander, E.S.: A 3d map of the human genome at kilobase resolution reveals principles of chromatin looping. Cell **159**(7), 1665–1680 (2014)
51. Rockafellar, R.T.: Convex Analysis. Princeton University Press, Princeton (1970)
52. Rockafellar, R.T.: Monotone operators and the proximal point algorithm. SIAM J. Control Optim. **14**(5), 877–898 (1976)
53. Rontsis, N., Goulart, P.J., Nakatsukasa, Y.: Efficient semidefinite programming with approximate ADMM (2019). arXiv:1912.02767 [math.OC]
54. Rontsis, N., Goulart, P.: Optimal approximation of doubly stochastic matrices. In: International Conference on Artificial Intelligence and Statistics, pp. 3589–3598. PMLR (2020)
55. Schenk, O., Gärtner, K., Karypis, G., Röllin, S., Hagemann, M.: Pardiso solver project. http://www.pardiso-project.org (2010)
56. Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S.: OSQP: an operator splitting solver for quadratic programs. Math. Program. Comput. **12**, 637–672 (2020)
57. Sturm, J.F.: Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optim. Methods Softw. **11**(1–4), 625–653 (1999)

58. Sun, Y., Andersen, M.S., Vandenberghe, L.: Decomposition in conic optimization with partially separable structure. SIAM J. Optim. **24**(2), 873–897 (2014)
59. Toh, K., Todd, M.J., Tütüncü, R.H.: SDPT3–a MATLAB software package for semidefinite programming, version 1.3. Optim. Methods Softw. **11**(1–4), 545–581 (1999)
60. Udell, M., Mohan, K., Zeng, D., Hong, J., Diamond, S., Boyd, S.: Convex optimization in Julia. In: SC14 Workshop on High Performance Technical Computing in Dynamic Languages (2014)
61. Vandenberghe, L., Andersen, M.S.: Chordal graphs and semidefinite optimization. Found. Trends Optim. **1**(4), 241–433 (2015)
62. Wright, S.J.: Primal-Dual Interior-Point Methods, vol. 54. SIAM, Philadelphia (1997)
63. Wright, S., Nocedal, J.: Numerical Optimization, vol. 35. Springer, Berlin (1999)
64. Yannakakis, M.: Computing the minimum fill-in is NP-complete. SIAM J. Algebraic Discrete Methods **2**(1), 77–79 (1981)
65. Zass, R., Shashua, A.: Doubly stochastic normalization for spectral clustering. In: Advances in Neural Information Processing Systems, pp. 1569–1576 (2007)
66. Zheng, Y., Fantuzzi, G., Papachristodoulou, A., Goulart, P., Wynn, A.: Chordal decomposition in operator-splitting methods for sparse semidefinite programs. Math. Program. **180**(1), 489–532 (2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.