COSMO: A conic operator splitting method for large convex problems

Michael Garstka, Mark Cannon & Paul Goulart[1]

***Abstract*— This paper describes the Conic Operator Splitting Method (COSMO), an operator splitting algorithm 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 solver is able to exploit chordal sparsity in the problem data and to detect infeasible problems. The low per-iteration computational cost makes the method particularly efficient for large problems, e.g. semidefinite programs in portfolio optimisation, graph theory, and robust control. Our Julia implementation is open-source, extensible, integrated into the Julia optimisation ecosystem and performs well on a variety of large convex problem classes.**

I. INTRODUCTION

We consider convex optimisation problems in the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \dots, l \\ & h_i(x) = 0, \quad i = 1, \dots, k, \end{array} \tag{1}$$

where we assume that both the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the inequality constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}$ are convex, and that the equality constraints $h_i(x) := a_i^\top x - b_i$ are affine. Convex optimisation problems feature in a wide range of applications, including problems in machine learning, finance, optimal control, and operations research [1]. Concrete examples of problems fitting the general form (1) include linear programming (LP), quadratic programming (QP), second-order cone programming (SOCP), and semidefinite programming (SDP) problems. Methods to solve each of these standard problem classes are well understood 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 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*, which stems from Karmarkar's original projective algorithm [2], and is able to solve LPs and QPs in polynomial time. Interior point methods were extended to problems with positive semidefinite (PSD) constraints in [3]. Primal-dual interior point methods apply variants of Newton's method to iteratively find a solution to a set of optimality 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. However, interior-point methods typically do not scale well for very large problems, since the Jacobian matrix has to be calculated and factored at each step.

Two approaches to overcome this limitation are active research areas. Firstly, a renewed focus on first-order methods with computationally cheaper per-iteration-cost and secondly the exploitation of sparsity in the problem data. Algorithms based on the alternating direction method of multipliers (ADMM), first described in [4], are simple to implement and computationally cheap, even for very large problems. They do not require strict convexity of the objective function and can therefore be applied to a variety of problems. Furthermore, [5] showed that ADMM can be analysed from the perspective of monotone operators which allowed further insight into the method. Two drawbacks of ADMM-based algorithms are that they tend to converge slowly to a high accuracy solution and the detection of infeasibility is more involved compared to interior-point methods. They are therefore most often used in applications where a modestly accurate solution is sufficient [6].

Methods of exploiting the sparsity pattern of PSD constraints in interior-point algorithms were considered in [7]. Sparsity has also been exploited in this context for problems with underlying graph structure, *e.g.* optimal power-flow problems in [8] and graph optimization problems in [9].

With the COSMO solver described in this paper we make the following contributions: (i) A first-order method is developed for large conic problems that is able to detect infeasibility without the need of a homogeneous self-dual embedding. (ii) COSMO directly supports quadratic objective functions, thus reducing overheads for applications with both PSD constraints and quadratic costs. (iii) The solver is written in a modular way in the fast and flexible programming language Julia, which allows rapid testing of extensions such as acceleration methods, approximate projections, and parallel computing on GPUs.

Related Work: Widely used solvers for conic problems, especially SDPs, are: SeDuMi [10], SDPT3 [11] (both open source, MATLAB), and MOSEK [12] (commercial, C). All of these implement primal-dual interior-point methods. Fukuda [7] developed an interior-point solver that exploits chordal sparsity patterns in PSD constraints.

Some solvers based on ADMM have been released recently. OSQP [13] is implemented in C, solves quadratic programs, and detects infeasibility based on the differences of the iterates [14]. The C-based SCS [15] implements an operator splitting method that solves a primal-dual pair of conic programs in order to provide infeasibility certificates. The underlying homogeneous self-dual embedding method has been extended [16] to exploit sparsity and implemented

[1]The authors are with the Department of Engineering Science, University of Oxford, Oxford, OX1 3PJ, UK. Email: {michael.garstka, mark.cannon, paul.goulart}@eng.ox.ac.uk

in the MATLAB solver CDCS. None of these conic solvers are able to handle quadratic cost functions directly. Instead they reformulate the problem by adding a second-order cone constraint, which increases the problem size. Moreover, they rely on primal-dual formulations to detect infeasibility.

Outline: In Section II we define the general conic problem format, its dual problem, as well as optimality and infeasibility conditions. The following section describes the ADMM algorithm underlying the COSMO solver. Section IV explains how to decompose SDPs in a preprocessing step provided the problem data has an aggregated sparsity pattern. Section V shows benchmark results of the COSMO solver compared with other state-of-the art solvers on nearest correlation matrix problems and block-diagonally structured SDPs. Section VI concludes the paper.

Notation: Denote the transformation of a matrix S into a vector s by stacking the columns as $s \coloneqq \operatorname{vec}(S)$. The inverse operation is denoted by $\operatorname{vec}^{-1}(s) =: \operatorname{mat}(s)$. Denote the Kronecker product of two matrices A and B as $A \otimes B$. Denote the space of real numbers $\mathbb{R}$, the n-dimensional real space $\mathbb{R}^n$, the space of symmetric $n \times n$ matrices $\mathbb{S}^n$, and the set of positive semidefinite $n \times n$ matrices $\mathbb{S}^n_+$. Sometimes we consider positive semidefinite constraints in vector form, and so define the space of vectorized positive semidefinite matrices as $\mathcal{S}^n_+ \coloneqq \{s \in \mathbb{R}^{n^2} : \operatorname{mat}(s) \in \mathbb{S}^n_+\}$. For a convex cone $\mathcal{K}$ denote the dual cone by $\mathcal{K}^*$. The *proximal operator* of a convex, closed and proper function $f \colon \mathbb{R}^n \to \mathbb{R}$ is given by $\operatorname{prox}_f(x) \coloneqq \operatorname{argmin}_y\{f(y)+\frac{1}{2}\|y-x\|_2^2\}$. We denote the *indicator function* of a nonempty, closed convex set $\mathcal{C} \subseteq \mathbb{R}^n$ as $\mathcal{I}_\mathcal{C}$ and the *projection* of $x \in \mathbb{R}^n$ onto $\mathcal{C}$ by $\Pi_\mathcal{C}(x) \coloneqq \operatorname{argmin}_{y \in \mathcal{C}} \|x-y\|_2^2$. We further denote the *support function* of $\mathcal{C}$ by $\sigma_\mathcal{C}(x) \coloneqq \sup_{y \in \mathcal{C}}\langle x, y\rangle$, and the *recession cone* of $\mathcal{C}$ by $\mathcal{C}^\infty \coloneqq \{y \in \mathbb{R}^n \mid x + ay \in \mathcal{C},\, x \in \mathcal{C}, a \geq 0\}$.

II. CONIC PROBLEMS

In this paper we address convex optimisation problems with quadratic objective functions and conic constraints in the form:

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

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 the 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

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

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 dual problem associated with (2) is given by:

$$\begin{array}{ll} \text{maximize} & -\frac{1}{2}x^\top P x + b^\top y - \sigma_\mathcal{K}(y) \\ \text{subject to} & Px - A^\top y = -q \\ & y \in (\mathcal{K}^\infty)^\circ, \end{array}$$

with dual variable $y \in \mathbb{R}^m$. The conditions for optimality follow from the Karush-Kuhn-Tucker (KKT) conditions for convex optimization problems:

$$Ax + s = b, \tag{3a}$$

$$Px + q - A^\top y = 0, \tag{3b}$$

$$s \in \mathcal{K}, \quad y \in (\mathcal{K}^\infty)^\circ. \tag{3c}$$

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 (3a)–(3c) then the pair (x^*, s^*) is called the primal solution and y^* is called the dual solution of problem (2).

A. Infeasibility certificates

Conditions for primal and dual infeasibility of ADMM that are directly applicable to problems of the form (2) are derived in [14]. To simplify notation, define the cone $\mathcal{C} = -\mathcal{K} + \{b\}$. Then, the following sets provide certificates for primal and dual infeasibility:

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Px = 0,\, Ax \in \mathcal{C}^\infty,\, \langle q, x\rangle < 0\}, \tag{4}$$

$$\mathcal{D} = \left\{y \in \mathbb{R}^m \mid A^\top y = 0,\, \sigma_\mathcal{C}(y) < 0\right\}. \tag{5}$$

The existence of some $y \in \mathcal{D}$ is a certificate that problem (2) is primal infeasible, while the existence of some $x \in \mathcal{P}$ is a certificate for dual infeasibility.

III. ADMM ALGORITHM

We use the same splitting as in [13] to transform problem (2) into standard ADMM format. The problem is rewritten by introducing the dummy variables $\tilde{x} = x$ and $\tilde{s} = s$:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\tilde{x}^\top P \tilde{x} + q^\top \tilde{x} + \mathcal{I}_{Ax+s=b}(\tilde{x}, \tilde{s}) + \mathcal{I}_\mathcal{K}(s) \\ \text{subject to} & (\tilde{x}, \tilde{s}) = (x, s), \end{array} \tag{6}$$

where the indicator functions of the sets $\mathcal{K}$ and$\{(x, s) \in \mathbb{R}^n \times \mathbb{R}^m \mid Ax + s = b\}$ have been introduced to move the constraints of (2) into the objective function. The augmented Lagrangian of (6) is given by

$$\begin{aligned} L(x, s, \tilde{x}, \tilde{s}, \lambda, y) = {} & \tfrac{1}{2}\tilde{x}^\top P \tilde{x} + q^\top \tilde{x} + \mathcal{I}_{Ax+s=b}(\tilde{x}, \tilde{s}) + \mathcal{I}_\mathcal{K}(s) \\ & + \tfrac{\sigma}{2}\left\|\tilde{x} - x + \tfrac{1}{\sigma}\lambda\right\|_2^2 + \tfrac{\rho}{2}\left\|\tilde{s} - s + \tfrac{1}{\rho}y\right\|_2^2, \end{aligned}$$

with step size parameters $\rho > 0$ and $\sigma > 0$ and dual variables $\lambda \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. The corresponding ADMM iteration steps are given by:

$$(\tilde{x}^{k+1}, \tilde{s}^{k+1}) = \operatorname*{argmin}_{\tilde{x}, \tilde{s}} L\left(\tilde{x}, \tilde{s}, x^k, s^k, \lambda^k, y^k\right), \tag{7a}$$

$$x^{k+1} = \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \frac{1}{\sigma}\lambda^k, \tag{7b}$$

$$\begin{aligned} s^{k+1} = \operatorname*{argmin}_s \Big(& \tfrac{\rho}{2}\left\|\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s + \tfrac{1}{\rho}y^k\right\|_2^2 \\ & + \mathcal{I}_\mathcal{K}(s)\Big), \end{aligned} \tag{7c}$$

$$\lambda^{k+1} = \lambda^k + \sigma\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k - x^{k+1}\right), \tag{7d}$$

$$y^{k+1} = y^k + \rho\left(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s^{k+1}\right), \tag{7e}$$

where we relaxed the (x, s)-updates and the dual variable update with relaxation parameter $\alpha \in (0, 2)$ according

to [17]. Notice from (7b) and (7d) that, for the dual variable λ corresponding to the constraint $x = \tilde{x}$, it holds that $\lambda^k = 0$ for all k.

A. Solution of the linear system

The minimization problem in (7a) can be solved as an equality-constrained quadratic program:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top\tilde{x} + \tfrac{\sigma}{2}\left\|\tilde{x} - x^k\right\|_2^2 \\ & + \tfrac{\rho}{2}\left\|\tilde{s} - s^k + \tfrac{1}{\rho}y^k\right\|_2^2 \\ \text{subject to} \quad & A\tilde{x} + \tilde{s} = b. \end{aligned}$$

Following the approach of [13], which uses the same splitting, this amounts to solving the linear system:

$$\begin{bmatrix} P + \sigma I & A^\top \\ A & -\frac{1}{\rho}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} -q + \sigma x^k \\ b - s^k + \frac{1}{\rho}y^k \end{bmatrix} \tag{8}$$

with

$$\tilde{s}^{k+1} = s^k - \frac{1}{\rho}\left(\nu^{k+1} + y^k\right),$$

and the Lagrangian multiplier $\nu \in \mathbb{R}^m$ associated with the equality constraint $Ax + s = b$. Note that the coefficient matrix in (8) is always quasi-definite and therefore has a well-defined $LDL^\top$ factorization with a diagonal D.

B. Projection step

As shown in [6], the minimization problem in (7c) can be interpreted as the $(1/\rho)$-weighted proximal operator of the indicator function $\mathcal{I}_\mathcal{K}$. This step is therefore equivalent to an Euclidean projection $\Pi_\mathcal{K}$ onto the cone $\mathcal{K}$, i.e.

$$s^{k+1} = \Pi_\mathcal{K}\Big(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k + \frac{1}{\rho}y^k\Big). \tag{9}$$

C. Algorithm steps

The calculations performed at each iteration are summarized in Algorithm 1. Observe that the coefficient matrix of the linear system in (8) is constant, so that one can precompute and cache the $LDL^\top$ factorization and efficiently evaluate line 2 with changing right hand sides. Lines 3, 4, and 6 are computationally inexpensive since they involve only vector addition and scalar-vector multiplication. The projection in line 5 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, while a projection onto the positive-semidefinite cone of dimension N involves an eigen-decomposition. Since direct methods for eigen-decompositions have a complexity of $\mathcal{O}(N^3)$, this makes line 5 the most computationally expensive operation of the algorithm for large SDPs.

D. Algorithm convergence

For feasible problems, Algorithm 1 produces a sequence of iterates (x^k, s^k, y^k) that satisfy the optimality conditions in (3) as $k \to \infty$. The convergence of Algorithm 1 is demonstrated in [13] by applying the Douglas-Rachford splitting to a problem reformulation.

In the Douglas-Rachford formulation, lines 5 and 6 become projections onto $\mathcal{K}$ and $\mathcal{K}^\circ$ respectively, so that the conic constraints in (3c) always hold. Furthermore, convergence of the residual iterates in (3a)–(3b) can be concluded from the convergence of the splitting variables

$$x^k - \tilde{x}^k \to 0, \quad s^k - \tilde{s}^k \to 0,$$

which generally holds for Douglas-Rachford splitting [18].

Let δx^k, δy^k, δs^k denote the differences between successive iterates of Algorithm 1:

$$\delta x^k = x^k - x^{k-1},\ \delta s^k = s^k - s^{k-1},\ \delta y^k = y^k - y^{k-1}.$$

For infeasible problems, [14] shows that these differences converge, with $\delta y = \lim_{k\to\infty}\delta y^k \in \mathcal{D}$ for primal infeasible problems and $\delta x = \lim_{k\to\infty}\delta x^k \in \mathcal{P}$ for dual infeasible problems, where the sets $\mathcal{P}$ and $\mathcal{D}$ are defined in (4) and (5).

We use the same termination conditions as [13].

Algorithm 1: ADMM steps

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

1 **Do**
2 $\quad(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow$ solve linear system (8);
3 $\quad\tilde{s}^{k+1} \leftarrow s^k - \frac{1}{\rho}(\nu^{k+1} + y^k)$;
4 $\quad x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k$;
5 $\quad s^{k+1} \leftarrow \Pi_\mathcal{K}\Big(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k + \frac{1}{\rho}y^k\Big)$;
6 $\quad y^{k+1} \leftarrow y^k + \rho(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s^{k+1})$;
7 **while** *termination criteria not satisfied*;

E. Scaling the problem data

The convergence rate of ADMM and other first-order methods depends on the scaling of the problem data; see [19]. For badly conditioned problems in particular, this suggests a preprocessing step in which the problem data is scaled in order to improve convergence. We use the same modified Ruiz equilibration method as in [13] to compute the diagonal positive definite scaling matrices D and E. The scaled problem data is given by

$$\hat{P} = DPD, \quad \hat{q} = Dq, \quad \hat{A} = EAD, \quad \hat{b} = Eb,$$

and the scaled convex cone is given by $E\mathcal{K} := \{Ev \in \mathbb{R}^m \mid v \in \mathcal{K}\}$. After solving the scaled problem, the original solution is obtained by reversing the scaling:

$$x = D\hat{x}, \quad s = E^{-1}\hat{s}, \quad y = E\hat{y}.$$

IV. CHORDAL DECOMPOSITION

As noted in Section III-C, for large SDPs the eigen-decomposition in the projection step (9) is the principal performance bottleneck for the algorithm. 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. If the aggregated sparsity pattern is represented by a *chordal* graph or a graph that has a chordal extension, then a large PSD constraint can be decomposed into a collection of smaller PSD constraints and additional coupling constraints [20]. 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 of each other, further performance improvement can be achieved by carrying them out in parallel. The approach is similar to [16]. We show how to transform the decomposed problem into the original problem format. This allows the decomposition to be performed in a preprocessing step before the problem is handed to the solver.

A. Graph preliminaries

Consider the undirected graph $G(V, E)$ with vertices $V = \{1, \ldots, n\}$ 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*. Any complete subset of vertices C of V is called a *clique* with cardinality $|C|$. The clique is called a *maximal* clique if it is not contained in any other clique. A *cycle* is a path of edges where every vertex is reachable from itself. A graph G is called *chordal* if every cycle of length greater than three has a *chord*, which is an edge between nonconsecutive vertices of the cycle. One can always find a *chordal embedding* $\bar{G}\left(V, \bar{E}\right)$ for a non-chordal graph $G(V, E)$ by adding extra edges [21].

B. Agler's theorem

The sparsity pattern of a symmetric matrix $S \in \mathbb{S}^n$ can be represented by an undirected graph $G(V, E)$. Every non-zero entry $S_{i,j} = S_{j,i} \neq 0$ introduces one edge $(i, j) \in E$. For a certain sparsity pattern $G(V, E)$ we define the spaces of sparse symmetric matrices as

$$\mathbb{S}^n(E, 0) \coloneqq \{S \in \mathbb{S}^n \mid S_{i,j} = S_{j,i} \neq 0 \text{ if } (i, j) \in E\},$$

and positive semidefinite sparse symmetric matrices as

$$\mathbb{S}^n_+(E, 0) \coloneqq \{S \in \mathbb{S}^n(E, 0) \mid S \succeq 0\}.$$

Let $S \in \mathbb{S}^n$ be a sparse symmetric matrix with a corresponding chordal graph G. Denote the set of maximal cliques of G as $\{C_1, \ldots, C_p\}$, where each C_ℓ contains an ordered list of vertices. Define the matrix $T_\ell \in \mathbb{R}^{|C_\ell| \times n}$ for clique C_ℓ as:

$$(T_\ell)_{i,j} \coloneqq \begin{cases} 1, & \text{if } C_\ell(i) = j \\ 0, & \text{otherwise.} \end{cases}$$

where $C_\ell(i)$ is the ith vertex of C_ℓ. A positive semidefinite constraint on S can then be decomposed according to the following theorem:

Theorem 1 (Agler's theorem [20]): Let $G(V, E)$ be a chordal graph with a set of maximal cliques $\{C_1, \ldots, C_p\}$. Then $S \in \mathbb{S}^n_+(E, 0)$ if and only if there exist matrices $S_\ell \in \mathbb{S}^{|C_\ell|}_+$ for $\ell = 1, \ldots, p$ such that

$$S = \sum_{\ell=1}^{p} T_\ell^\top S_\ell T_\ell. \tag{10}$$

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 clique C_ℓ. The vectorized form of (10) is:

$$s = \sum_{\ell=1}^{p} H_\ell^\top s_\ell, \quad \text{with } H_\ell = T_\ell \otimes T_\ell.$$

where $s = \operatorname{vec}(S)$.

C. Decomposition of PSD constraints

Consider the following problem with a PSD constraint in matrix form:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^\top P x + q^\top x \\ \text{subject to} \quad & \sum_{i=1}^{m} \mathcal{A}_i x_i + S = B \\ & S \in \mathbb{S}^r_+, \end{aligned} \tag{11}$$

with symmetric matrices $\mathcal{A}_i, B, S \in \mathbb{S}^r$. Note that (11) can be transformed into (2) by setting $b = \operatorname{vec}(B)$, $s = \operatorname{vec}(S)$, $A = [\operatorname{vec}(\mathcal{A}_1), \ldots, \operatorname{vec}(\mathcal{A}_m)]$, and $\mathcal{K} = \mathcal{S}^r_+$. Assume that each matrix $\mathcal{A}_i$ and B has a sparsity pattern

$$\mathcal{A}_i \in \mathbb{S}^r(E_{\mathcal{A}_i}, 0) \text{ and } B \in \mathbb{S}^r(E_B, 0),$$

with edge sets $E_{\mathcal{A}_i}$ and E_B. The *aggregated sparsity pattern* of (11) is described by the graph $G(V, E)$ where E is given by

$$E = E_{B_i} \cup E_{\mathcal{A}_1} \cup \cdots \cup E_{\mathcal{A}_m}.$$

In the following we assume that $G(V, E)$ is chordal or that a chordal embedding has been found. Moreover, the graph has a set of maximal cliques $\{C_1, \ldots, C_p\}$. The equality constraint in (11) constrains the decision variable S to have the aggregated sparsity pattern, *i.e.* $S \in \mathbb{S}^r(E, 0)$.

Under the preceding assumptions Agler's theorem can be applied to decompose the PSD constraint in (11), yielding

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^\top P x + q^\top x \\ \text{subject to} \quad & \sum_{i=1}^{m} \mathcal{A}_i x_i + \sum_{\ell=1}^{p} T_\ell^\top S_\ell T_\ell = B \\ & S_\ell \in \mathbb{S}^{|C_\ell|}_+, \quad \ell = 1, \ldots, p. \end{aligned} \tag{12}$$

The vectorized form of (12) is given by:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^\top P x + q^\top x \\ \text{subject to} \quad & A x + \sum_{\ell=1}^{p} H_\ell^\top s_\ell = b \\ & s_\ell \in \mathcal{S}^{|C_\ell|^2}_+, \quad \ell = 1, \ldots, p. \end{aligned} \tag{13}$$

D. Decomposition as a preprocessing step

In order to perform the chordal decomposition before handing it to the solver, we transform (13) into the original solver format. The required transformation is achieved by rewriting the constraints of (13) as:

$$A x + \mathcal{H} \xi = b, \quad \xi \in \bar{\mathcal{K}} \tag{14}$$

where $\xi = [s_1^\top, \ldots, s_p^\top]^\top$, $\mathcal{H} = \left[H_1^\top, \ldots, H_p^\top\right]$, and $\bar{\mathcal{K}} = \mathcal{S}^{|C_1|}_+ \times \cdots \times \mathcal{S}^{|C_p|}_+$. Using the augmented primal variables $\bar{x} = [x^\top, \xi^\top]^\top$ and $\bar{s} = [\bar{s}_1^\top, \bar{s}_2^\top]^\top$ the optimization problem is transformed into

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} \bar{x}^\top \begin{bmatrix} P & 0 \\ 0 & 0 \end{bmatrix} \bar{x} + \begin{bmatrix} q \\ 0 \end{bmatrix}^\top \bar{x} \\ \text{subject to} \quad & \begin{bmatrix} A & \mathcal{H} \\ 0 & -I \end{bmatrix} \bar{x} + \bar{s} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \\ & \bar{s}_1 \in \{0\}^{r^2},\ \bar{s}_2 \in \bar{\mathcal{K}}, \end{aligned}$$

where $\{0\}^n$ denotes the zero cone.

The steps described in this section allow us to exploit the chordal sparsity of the problem in order to decompose the PSD constraints. The decomposed problem is then transformed back into the solver format in a preprocessing step and subsequently handed to the solver. The projection step of the decomposed problem (13)

$$s_\ell^{k+1} = \Pi_{\mathcal{S}_+^{|C_\ell|}} \left(\alpha \tilde{s}_\ell^{k+1} + (1-\alpha) s_\ell^k + \frac{1}{\rho} y_\ell^k \right), \quad \text{for } \ell = 1, \ldots, p, \tag{15}$$

is implemented serially. However, since the p projections can be carried out independently, a parallel implementation promises substantial performance gains.

V. IMPLEMENTATION & NUMERICAL RESULTS

The algorithm described in this paper is implemented by the Conic Operator Splitting Method (COSMO) solver, an open-source package written in Julia [22]. The source code and documentation are available at:

https://github.com/oxfordcontrol/COSMO.jl.

The solver provides its own interface but can also be called via the modelling language JuMP [23]. This section compares the performance of COSMO with the interior-point solvers MOSEK v8.1 and SeDuMi v1.3, and with the first-order ADMM solver SCS v2.0.2 in benchmark tests. All the experiments were carried out on a MacBook with a 2.6 GHz Intel Core i5 CPU and 8 GB of RAM. We configured COSMO with absolute accuracy $\epsilon = 10^{-4}$, relaxation parameter $\alpha = 1.6$, and step sizes $\rho_0 = 0.1$ and $\sigma = 10^{-6}$. It should be noted that all solvers use different termination criteria and are configured with their default error tolerance. COSMO and SCS are configured with an accuracy of 10^{-4} and 10^{-5} respectively, whereas the interior-point methods are configured with a tolerance of 10^{-6}. Experiments with the same error tolerance for all solvers resulted in qualitatively similar results.

A. 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 optimization. The correlation matrix of a stock portfolio might lose its positive semidefiniteness due to noise and rounding errors of previous data manipulations. Consequently, it is of interest to find the nearest correlation matrix X to a given data matrix $C \in \mathbb{R}^{n \times n}$. The problem is given by:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} \|X - C\|_F^2 \\ \text{subject to} \quad & X_{ii} = 1, \quad i = 1, \ldots, n \\ & X \in \mathbb{S}_+^n, \end{aligned} \tag{16}$$

with Frobenius norm $||X||_F = \left(\sum_{ij} |x_{ij}|^2\right)^{1/2}$. In order to transform the problem into the solver format, C and X are vectorized and the squared norm is expanded:

$$\begin{aligned} \text{minimize} \quad & (1/2)(x^\top x - 2c^\top x + c^\top c) \\ \text{subject to} \quad & \begin{bmatrix} E \\ -I \end{bmatrix} x + s = \begin{bmatrix} 1_{n\times 1} \\ 0_{n^2 \times 1} \end{bmatrix} \\ & s \in \{0\}^n \times \mathcal{S}_+^n, \end{aligned} \tag{17}$$

with $c = \text{vec}(C)$ and $x = \text{vec}(X)$. The matrix E extracts the n diagonal entries X_{ii} from the vectorized form $x = \text{vec}(X)$.

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. Figure 1 shows the benchmark results for increasing matrix dimension n.

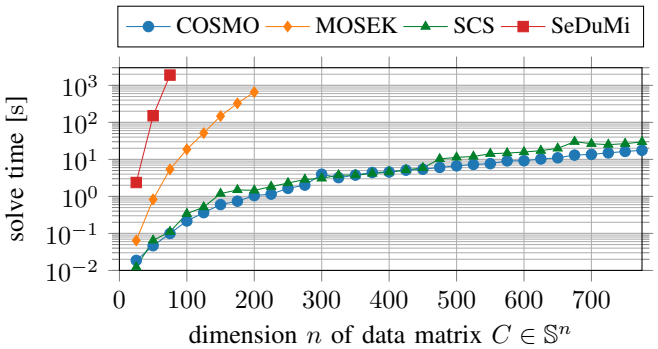

Fig. 1. Solve time of benchmarked solvers for increasing problem size of nearest correlation matrix problems. The results for MOSEK and SeDuMi are shown until they exceeded the time limit of 60 min.

The first-order methods SCS and COSMO outperform the interior-point solvers. Furthermore, COSMO solves the problems faster than SCS as the problem dimension gets larger, *e.g.* 18.6 s vs. 31.7 s for $N = 800$. This is likely because SCS has to transform the quadratic cost term into an additional second-order cone constraint which increases the problem size.

B. Chordal block-arrow SDP

To show the benefits of the chordal decomposition technique we consider randomly generated SDPs of the form (11) with a block-arrow aggregate sparsity pattern similar to test problems in [16]. Figure 2 shows the sparsity pattern of the PSD constraint.

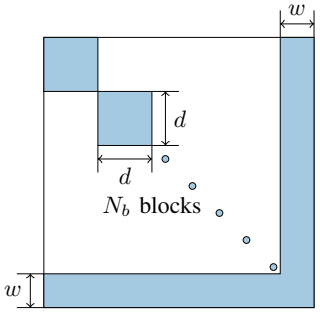

Fig. 2. Parameters of block-arrow sparsity pattern.

The sparsity pattern is generated based on the following parameters: block size d, number of blocks N_b, and width of

the arrowhead w. The dimension of the PSD cone is given by $r = N_b d + w$. Note that the graph corresponding to the sparsity pattern is always chordal.

In this section we study the effects of independently increasing the block size d and the number of blocks N_b. The parameters for the two test cases are:

- $N_b = 50, 60, \ldots, 140$, $d = 10$, $w = 20$, and $m = 100$
- $d = 10, 12, \ldots, 28$, $N_b = 50$, $w = 10$, and $m = 100$.

Solve times are shown in Fig. 3 and 4. The line COSMO(CD) refers to the solver with chordal decomposition turned on.

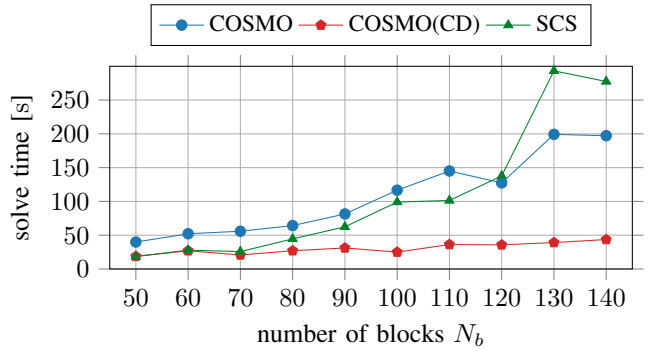

Fig. 3. Solve time for increasing number of blocks of block-arrow sparsity pattern.

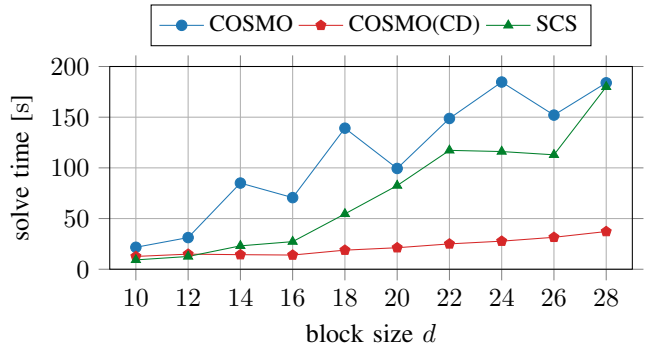

Fig. 4. Solve time for increasing block size of block-arrow sparsity pattern.

Only the results for the first-order methods are shown since both MOSEK and SeDuMi exited with errors when trying to solve problems of this dimension. It can be seen that COSMO(CD) outperforms both SCS and COSMO without decomposition in terms of solve time. Moreover, COSMO(CD) scales better when the number of blocks or the block size are increased.

VI. CONCLUSIONS

This paper describes the first-order solver COSMO and the ADMM algorithm on which it is based. The solver combines direct support of quadratic objectives, infeasibility detection and chordal decomposition of PSD constraints. The performance of the solver is illustrated on two benchmark problems that challenge different aspects of modern solvers. The implementation in the Julia language facilitates rapid development and testing of ideas. Further performance gains seem achievable by exploring acceleration methods, approximate projections onto the PSD cone, and parallel computing on GPUs.

REFERENCES

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

[2] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the 16th annual ACM symposium on Theory of Computing*. ACM, 1984, pp. 302–311.

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

[4] D. Gabay and B. Mercier, *A dual algorithm for the solution of non linear variational problems via finite element approximation*. Institut de recherche d'informatique et d'automatique, 1975.

[5] J. Eckstein, "Splitting methods for monotone operators with applications to parallel optimization," Ph.D. dissertation, Massachusetts Institute of Technology, 1989.

[6] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

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

[8] D. K. Molzahn, J. T. Holzer, B. C. Lesieutre, and C. L. DeMarco, "Implementation of a large-scale optimal power flow solver based on semidefinite programming," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 3987–3998, 2013.

[9] F. Alizadeh, "Interior point methods in semidefinite programming with applications to combinatorial optimization," *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 13–51, 1995.

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

[11] K.-C. Toh, M. J. Todd, and R. H. Tütüncü, "SDPT3 - A MATLAB software package for semidefinite programming, version 1.3," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 545–581, 1999.

[12] MOSEK, Aps, "The MOSEK optimization toolbox for MATLAB manual. Version 8.0 (revision 57)," 2017.

[13] B. Stellato, G. Banjac, P. Goulart, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *arXiv preprint arXiv:1711.08013*, 2017.

[14] G. Banjac, P. Goulart, B. Stellato, and S. Boyd, "Infeasibility detection in the alternating direction method of multipliers for convex optimization," 2017.

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

[16] Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn, "Chordal decomposition in operator-splitting methods for sparse semidefinite programs," *arXiv preprint arXiv:1707.05058*, 2017.

[17] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1, pp. 293–318, 1992.

[18] H. Bauschke and P. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces Springer*, 1st ed. Springer, 2011.

[19] P. Giselsson and S. Boyd, "Diagonal scaling in Douglas-Rachford splitting and ADMM," in *53rd IEEE Annual Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 5033–5039.

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

[21] L. Vandenberghe and M. S. Andersen, "Chordal graphs and semidefinite optimization," *Foundations and Trends in Optimization*, vol. 1, no. 4, pp. 241–433, 2015.

[22] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[23] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.