COSMO: A conic operator splitting method for convex conic problems

Michael Garstka Mark Cannon Paul Goulart [*]

January 31, 2019

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.

1 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 assume that the the equality constraints $h_i(x) := a_i^\top x - b_i$ are affine. We will denote an optimal solution to this problem (if it exists) as x^*. Convex optimisation problems feature heavily in a wide reange of research areas and industries, including problems in machine learning [CV95], finance [Boy+17], optimal control [Boy+94], and operations research [BHH01]. Concrete examples of problems fitting the general form (1) include linear programming (LP), quadratic programming (QP), second-order cone programming (SOCP), and

[*]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

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 making problems and machine learning poses a challenge for state-of-the-art software.

Algorithms for LPs were first used to solve military planning and allocation problems in the 1940s [Dan63]. In 1947 Danzig developed the simplex method that solves LPs by searching for the optimal solution along the vertices of the inequality polytope. Extensions to the method led to the general field of *active-set methods* [Wol59] that are able to solve both LPs and QPs, and which search for an optimal point by iteratively constructing a set of active constraints. Although often efficient in practice, a major theoretical drawback is that the worst-case complexity increases exponentially with the problem size [WN99].

The most common approach taken by modern convex solvers is the *interior-point method* [WN99], which stems from Karmarkar's original projective algorithm [Kar84], and is able to solve LPs and QPs in polynomial time. Interior point methods have since been extended to problems with positive semidefinite (PSD) constraints in [Hel$^+$96] and [AHO98]. The 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. Most notably, the Mehrotra predictor-corrector method in [Meh92] forms the basis of several implementations because of its strong practical performance [Wri97]. However, interior-point methods typically do not scale well for large problems, since the Jacobian matrix has to be calculated and factored at each step.

Two main 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. First-order methods are known to handle larger problems at the expense of reduced accuracy compared to interior-point methods. In the 1960s Everett [Eve63] proposed a dual decomposition method that allows one to decompose a separable objective function which makes each iteration cheaper. Augmented Lagrangian methods by Miele ([Mie$^+$71; MCL71; Mie$^+$72]), Hestenes [Hes69], and Powell [Pow69] 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 [GM75], allowed to combine the advantages of dual decomposition and the superior convergence and robustness of augmented Lagrangian methods. Subsequently, it was shown that ADMM can be analysed from the perspective of monotone operators and that it is a special case of the Douglas-Rachford splitting [Eck89] as well as of the proximal point algorithm in [Roc76], 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 compared to interior-point methods. They are therefore most often used in applications where a modestly accurate solution is sufficient [P$^+$14]. Most of the discoveries around first-order methods such as ADMM happened in the 1970s/80s long before the demand for large scale optimisation which explains why they stayed less known and have now resurfaced.

A method of exploiting the sparsity pattern of PSD constraints in an interior-point algorithm was developed in [Fuk+01]. Sparsity has also been studied in this context for problems with underlying graph structure, *e.g.* optimal power-flow problems in [Mol+13] and graph optimization problems in [Ali95].

With the COSMO solver described in this paper we make the following contributions:

1. We implemented a first-order method for large conic problems that is able to detect infeasibility without the need of a homogeneous self-dual embedding.
2. COSMO directly supports quadratic objective functions, *i.e.* no reformulation of the problem is necessary. For QPs this avoids a handicap compared to native QP solvers and introduces less overhead for applications with both PSD constraints and quadratic costs.
3. The solver is written in a modular way in the flexible, yet fast, programming language Julia. This allows quick testing of extensions in the future, such as acceleration methods, approximate projections, and parallel computing on GPUs.
4. The object-oriented design of the solver allows users to extend the solver by defining their own convex sets if they provide the corresponding projection function.

Related Work Widely used solver for conic problems, especially SDPs, are SeDuMi [Stu99], SDPT3 [TTT99] (both open source, MATLAB), and MOSEK [MOS17] (commercial, C). All of them implement primal-dual interior-point methods. Fukuda [Fuk+01] developed an interior-point solver that exploits chordal sparsity patterns in PSD constraints.

Some solvers based on the ADMM method have been released recently. OSQP [Ste+18] is implemented in C, solves quadratic programs, and detects infeasibility based on the differences of the iterates [Ban+17]. The C-based SCS [ODo+16] 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 [Zhe+17] to exploit sparsity and implemented in the MATLAB solver CDCS. The mentioned conic solvers are unable 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 2 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 4 explains how to decompose SDPs in a preprocessing step provided the problem data has an aggregated sparsity pattern. Implementation details and code related design choices are discussed in Section 5. Section 6 shows benchmark results of the COSMO solver vs. other state-of-the art solvers on nearest correlation matrix and block-diagonally structured test problems. Section 7 concludes the paper.

Notation We introduce some notation and definitions that will be used throughout this paper. Denote the transformation of a matrix S into a vector s by stacking the columns as $s := (S)$. The inverse operation is denoted by $^{-1}(s) =: \mathrm{mat}(s)$. Denote the Kronecker product of two matrices A and B as $C = A \otimes B$. The ℓ-largest norm of $v \in \mathbb{R}^n$, *i.e.* the sum of the ℓ largest absolute values of v, is denotes as $\|v\|_{[\ell]}$. Denote the space of real numbers $\mathbb{R}$, the n-dimensional real space $\mathbb{R}^n$, the space of symmetric matrices $\mathbb{S}^n$, and the set of positive semidefinite matrices $\mathbb{S}^n_+$. Sometimes we consider positive semidefinite constraints in vector form, and so define the space of vectorized positive semidefinite matrices as

$$\mathcal{S}^n_+ := \left\{ s \in \mathbb{R}^{n^2} : \mathrm{mat}(s) \in \mathbb{S}^n_+ \right\}.$$

For a convex cone $\mathcal{K}$ denote the *polar cone* by

$$\mathcal{K}^\circ := \left\{ y \in \mathbb{R}^n \mid \sup_{x \in \mathcal{K}} \langle x, y \rangle \leq 0 \right\},$$

the *normal cone* of $\mathcal{K}$ by

$$N_{\mathcal{K}}(x) := \left\{ y \in \mathbb{R}^n \mid \sup_{\bar{x} \in \mathcal{K}} \langle \bar{x} - x, y \rangle \leq 0 \right\},$$

and following [Roc70] the *recession cone* of $\mathcal{K}$ by

$$\mathcal{K}^\infty := \{ y \in \mathbb{R}^n \mid x + ay \in \mathcal{K},\ x \in \mathcal{K}, a \geq 0 \}.$$

The *proximal operator* of a convex, closed and proper function $f \colon \mathbb{R}^n \to \mathbb{R}$ is given by

$$\mathrm{prox}_f(x) := \operatorname*{argmin}_y \left\{ f(y) + \tfrac{1}{2} \|y - x\|_2^2 \right\}.$$

We denote the *indicator function* of a nonempty, closed convex set $\mathcal{C} \subseteq \mathbb{R}^n$ by

$$I_{\mathcal{C}}(x) := \begin{cases} 0 & x \in \mathcal{C} \\ +\infty & \text{otherwise,} \end{cases}$$

the *projection* of $x \in \mathbb{R}^n$ onto $\mathcal{C}$ by:

$$\Pi_{\mathcal{C}}(x) := \operatorname*{argmin}_{y \in \mathcal{C}} \|x - y\|_2^2,$$

and the *support function* of $\mathcal{C}$ by:

$$\sigma_{\mathcal{C}}(x) := \sup_{y \in \mathcal{C}} \langle x, y \rangle.$$

2 Conic Problems

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

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

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 dual problem associated with (2) is given by:

$$\begin{array}{ll} \text{maximize} & -\frac{1}{2}x^\top Px + b^\top y - \sigma_{\mathcal{K}}(y) \\ \text{subject to} & Px - A^\top y = -q \\ & y \in (\mathcal{K}^\infty)^\circ, \end{array} \tag{4}$$

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

The conditions for optimality follow from the Karush-Kuhn-Tucker (KKT) conditions for convex optimization problems:

$$Ax + s = b, \tag{5a}$$

$$Px + q - A^\top y = 0, \tag{5b}$$

$$s \in \mathcal{K}, \quad y \in N_{\mathcal{K}}(s). \tag{5c}$$

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 (5a)–(5c) then the pair (x^*, s^*) is called the primal solution and y^* is called the dual solution of problem (2).

2.1 Infeasibility certificates

The authors of [Ban+17] developed primal and dual infeasibility conditions for ADMM. These conditions are directly applicable to problems of the form (2). To simplify the notation of the conditions define the cone $\mathcal{C} = -\mathcal{K} + \{b\}$. Then, the following sets provide certificates for primal and dual infeasibility:

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Px = 0,\ Ax \in \mathcal{C}^\infty,\ \langle q, x\rangle < 0\}\,. \tag{6}$$

$$\mathcal{D} = \left\{y \in \mathbb{R}^m \mid A^\top y = 0,\ \sigma_{\mathcal{C}}(y) < 0\right\}, \tag{7}$$

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.

3 ADMM Algorithm

We use the same splitting as in [Ste⁺18] to transform problem (2) into standard ADMM format. The problem is rewritten by introducing the dummy variables $\tilde{x} = x$ and $\tilde{s} = s$:

$$\begin{aligned} &\text{minimize } \tfrac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top \tilde{x} + I_{Ax+s=b}(\tilde{x}, \tilde{s}) + I_{\mathcal{K}}(s) \\ &\text{subject to } (\tilde{x}, \tilde{s}) = (x, s), \end{aligned} \tag{8}$$

where the indicator functions of the sets $\left\{(x, s) \in \mathbb{R}^n \times \mathbb{R}^m \mid Ax + s = b\right\}$ and $\mathcal{K}$ were used to move the constraints of (2) into the objective function. The augmented Lagrangian of (8) is given by

$$\begin{aligned} L(x, s, \tilde{x}, \tilde{s}, \lambda, y) = \tfrac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top \tilde{x} + \mathcal{I}_{Ax+s=b}(\tilde{x}, \tilde{s}) + \mathcal{I}_{\mathcal{K}}(s) \\ + \frac{\sigma}{2}\left\|\tilde{x} - x + \tfrac{1}{\sigma}\lambda\right\|_2^2 + \frac{\rho}{2}\left\|\tilde{s} - s + \tfrac{1}{\rho}y\right\|_2^2, \end{aligned} \tag{9}$$

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:

$$(\tilde{x}^{k+1}, \tilde{s}^{k+1}) = \operatorname*{argmin}_{\tilde{x}, \tilde{s}} L\left(\tilde{x}, \tilde{s}, x^k, s^k, \lambda^k, y^k\right), \tag{10a}$$

$$x^{k+1} = \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \frac{1}{\sigma}\lambda^k, \tag{10b}$$

$$s^{k+1} = \operatorname*{argmin}_{s} \frac{\rho}{2}\left\|\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s + \tfrac{1}{\rho}y^k\right\|_2^2 + I_{\mathcal{K}}(s), \tag{10c}$$

$$\lambda^{k+1} = \lambda^k + \sigma\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k - x^{k+1}\right), \tag{10d}$$

$$y^{k+1} = y^k + \rho\left(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s^{k+1}\right), \tag{10e}$$

where we relaxed the z-update and the dual variable update with relaxation parameter $\alpha \in (0, 2)$ according to [EB92]. A study in [Eck94] found faster convergence of their ADMM algorithm by choosing an over-relaxation with $\alpha \approx 1.6$ on the considered test problems. Notice from (10b) and (10d) that for the dual variable corresponding to the constraint $x = \tilde{x}$ it holds $\lambda^k = 0$ for all k.

3.1 Solution of the equality constrained QP

The minimization problem in (10a) has the form of an equality constrained quadratic program:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top\tilde{x} + \frac{\sigma}{2}\left\|\tilde{x} - x^k\right\|_2^2 + \frac{\rho}{2}\left\|\tilde{s} - s^k + \frac{1}{\rho}y^k\right\|_2^2 \\ \text{subject to} & A\tilde{x} + \tilde{s} = b. \end{array} \tag{11}$$

The solution of (11) is obtained by solving a linear system. The corresponding Lagrangian is given by:

$$\mathcal{L}(\tilde{x}, \tilde{s}, \nu) = \frac{1}{2}\tilde{x}^\top P\tilde{x} + q^\top\tilde{x} + \frac{\sigma}{2}\left\|\tilde{x} - x^k\right\|_2^2 + \frac{\rho}{2}\left\|\tilde{s} - s^k + \frac{1}{\rho}y^k\right\|_2^2 + \nu^\top\left(A\tilde{x} + \tilde{s} - b\right), \tag{12}$$

where the Lagrangian multiplier $\nu \in \mathbb{R}^m$ accounts for the equality constraint $Ax + s = b$. Thus, the KKT optimality conditions for this equality constrained QP are given by:

$$\frac{\partial\mathcal{L}}{\partial\tilde{x}} = P\tilde{x}^{k+1} + q + \sigma\left(\tilde{x}^{k+1} - x^k\right) + A^\top\nu^{k+1} = 0, \tag{13}$$

$$\frac{\partial\mathcal{L}}{\partial\tilde{s}} = \rho\left(\tilde{s}^{k+1} - s^k + \frac{1}{\rho}y^k\right) + \nu^{k+1} = 0, \tag{14}$$

$$A\tilde{x}^{k+1} + \tilde{s}^{k+1} - b = 0. \tag{15}$$

Elimination of $\tilde{s}^{k+1}$ from the equations leads to the linear system:

$$\begin{bmatrix} P + \sigma I & A^\top \\ A & -\frac{1}{\rho}I \end{bmatrix}\begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} -q + \sigma x^k \\ b - s^k + \frac{1}{\rho}y^k \end{bmatrix} \tag{16}$$

with

$$\tilde{s}^{k+1} = s^k - \frac{1}{\rho}\left(\nu^{k+1} + y^k\right). \tag{17}$$

Note that the introduction of the dummy variable $\tilde{x}$ led to the term σI in the upper-left corner of the coefficient matrix in (16). Consequently, the coefficient matrix in (16) is always quasi-definite [Van95], *i.e.* it always has a positive definite upper-left block and a negative definite lower-right block, and is therefore full rank even when $P = 0$ or A is rank deficient. Following [Van95] the left hand side of (16) always has a well-defined $LDL^\top$ factorization with a diagonal D.

3.2 Projection step

As shown in [P+14] the minimization problem in (10c) can be interpreted as the ρ-weighted proximal operator of the indicator function $I_{\mathcal{K}}$. It is therefore equivalent to the Euclidean projection Π onto the cone $\mathcal{K}$, i.e.

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

```
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     $(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow$ solve linear system (16);
3     $\tilde{s}^{k+1} \leftarrow s^k - \frac{1}{\rho}(\nu^{k+1} + y^k)$;
4     $x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k$;
5     $s^{k+1} \leftarrow \Pi_{\mathcal{K}}\left(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k + \frac{1}{\rho}y^k\right)$;
6     $y^{k+1} \leftarrow y^k + \rho(\alpha\tilde{s}^{k+1} + (1-\alpha)s^k - s^{k+1})$;
7 while termination criteria not satisfied;
```

3.3 Algorithm steps

The calculations performed at each iteration are summarized in Algorithm 1. Observe that the coefficient matrix of the linear system in (16) 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 eigendecompositions have a complexity of $\mathcal{O}(N^3)$, this turns line 5 into the most computationally expensive operation of the algorithm for large SDPs.

3.4 Algorithm convergence

For feasible problems, Algorithm 1 produces a sequence of iterates (x^k, s^k, y^k) that satisfy the optimality conditions in (5) as $k \to \infty$. The authors in [Ste+18] show convergence of Algorithm 1 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 (5c) always hold. Furthermore, convergence of the residual iterates in (5a)–(5b) can be concluded from the convergence of the splitting variables

$$x^k - \tilde{x}^k \to 0, \quad s^k - \tilde{s}^k \to 0, \tag{18}$$

which generally holds for Douglas-Rachford splitting [BC11].

For infeasible problems, [Ban+17] showed that Algorithm 1 leads to convergence of the successive differences between iterates

$$\delta x^k = x^k - x^{k-1},\ \delta s^k = s^k - s^{k-1},\ \delta y^k = y^k - y^{k-1}. \tag{19}$$

For primal infeasible problems $\delta y = \lim_{k\to\infty} \delta y^k$ will satisfy condition (7), whereas for dual infeasible problems $\delta x = \lim_{k\to\infty} \delta x^k$ is a certificate of (6).w

3.5 Scaling the problem data

The convergence rate of ADMM and other first-order methods depends on the scaling of the problem data; see [GB14]. Especially for badly conditioned problems this suggests a preprocessing step where the problem data is scaled in order to improve convergence. For certain problem classes an optimal scaling has been found, see [GB14; P G15; Gre97]. However, the computation of the optimal scaling is often more complicated than solving the original problem. Consequently, most algorithms rely on heuristic methods such as matrix equilibration.

We scale the equality constraints by diagonal positive definite matrices D and E. The scaled form of (2) is given by:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\hat{x}^\top \hat{P}\hat{x} + \hat{q}^\top \hat{x} \\ \text{subject to} & \hat{A}\hat{x} + \hat{s} = \hat{b}, \\ & \hat{s} \in E\mathcal{K}, \end{array} \tag{20}$$

with scaled problem data

$$\hat{P} = DPD, \quad \hat{q} = Dq, \quad \hat{A} = EAD, \quad \hat{b} = Eb, \tag{21}$$

and the scaled convex cone $E\mathcal{K} := \{Ev \in \mathbb{R}^m \mid v \in \mathcal{K}\}$. After solving (20) the original solution is obtained by reversing the scaling:

$$x = D\hat{x}, \quad s = E^{-1}\hat{s}, \quad y = E\hat{y}. \tag{22}$$

One heuristic strategy that has been shown to work well in practice is to choose the scaling matrices D and E to equilibrate, *i.e.* reduce the condition number of, the problem data. The Ruiz equilibration technique described in [Rui01] iteratively scales the rows and columns of a matrix to have a norm of 1 and converges linearly. We apply the modified Ruiz algorithm shown in Algorithm 2 to reduce the condition number of the symmetric matrix R

$$R = \begin{bmatrix} P & A^\top \\ A & 0 \end{bmatrix} \tag{23}$$

which represents the problem data. Since R is symmetric it suffices to consider the columns $R_{c,i}$ of R. At each iteration the scaling routine calculates the norms of each column. For the columns with norms higher than the tolerance $\tau = 10^{-6}$ the scaling vector c is updated with the inverse square root of the norm. If the norm is below the tolerance, the corresponding column will be scaled by 1.

Since the matrix E scales the (possibly composite) cone constraint, the scaling must ensure that if $s \in \mathcal{K}$ then $E^{-1}s \in \mathcal{K}$. Let $\mathcal{K}$ be a Cartesian product of N cones as in (3) and partition E into blocks

$$E = \text{diag}\left(E_1, \dots, E_N\right) \text{ with block } E_i \in \mathbb{R}^{m_i \times m_i}, \tag{24}$$

Algorithm 2: Modified Ruiz equilibration

```
1  set $D = I_n$, $E = I_m$, $c = \mathbf{1}_{m+n}$;
2  Do
3      for $i = 1, \dots, n+m$ do
4          if $\|R_{c,i}\|_\infty > \tau$ then
5              $c_i \leftarrow \|R_{c,i}\|_\infty^{-\frac{1}{2}}$;
6      $\hat{D} = \text{diag}(c_{1:n}), \ \hat{E} = \text{diag}(c_{n+1:n+m})$;
7      $D = \hat{D} \cdot D, \ E = \hat{E} \cdot E$;
8      $P = \hat{D}P\hat{D}, \ A = \hat{E}A\hat{D}$;
9      assemble $R$;
10 while $\|\mathbf{1} - c\|_\infty > \text{tol}$;
11 return $D$, $E$;
```

which scales the constraint corresponding to $\mathcal{K}_i$. For each cone $\mathcal{K}_i \in \mathbb{R}^{m_i}$ that requires a scalar or symmetric scaling, *e.g.* a second-order cone or positive semidefinite cone, the corresponding block E_i is replaced with

$$\bar{E}_i := e_i I_{m_i}, \quad \text{for } i = 1, \dots, N \tag{25}$$

where $e_i = \text{tr}(E_i)/m_i$ is the mean value of the diagonal entries of the original block in E.

3.6 Termination criteria

The termination criteria discussed in the following section are based on the unscaled problem data and iterates. Thus, before checking for termination the solver first reverses the scaling according to equations (21)–(22). To measure the progress of the algorithm, we define the primal and dual residuals of the problem as:

$$r_p := Ax + s - b, \tag{26a}$$

$$r_d := Px + q - A^\top y. \tag{26b}$$

According to [Boy+11] a valid termination criterion is that the size of the norms of the residual iterates in (26) are small. Our algorithm terminates if the residual norms are below the sum of an absolute and a relative tolerance term:

$$\left\|r_p^k\right\|_\infty \le \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\left\{\left\|Ax^k\right\|_\infty, \left\|s^k\right\|_\infty, \|b\|_\infty\right\}, \tag{27a}$$

$$\left\|r_d^k\right\|_\infty \le \epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\left\{\left\|Px^k\right\|_\infty, \|q\|_\infty, \left\|A^\top y^k\right\|_\infty\right\}, \tag{27b}$$

with user defined absolute ϵ_{abs} and relative ϵ_{rel} tolerances.

Following [Ban+17] the algorithm determines if the one-step differences δx^k and δy^k of the primal and dual variable fulfil the normalized infeasibility conditions (7)–(6) up to certain tolerances $\epsilon_{\text{p,inf}}$

and $\epsilon_{\mathrm{d,inf}}$. The solver returns a primal infeasibility certificate if

$$\left\|A^\top \delta y^k\right\|_\infty / \left\|\delta y^k\right\|_\infty \leq \epsilon_{\mathrm{p,inf}}, \tag{28a}$$

$$\sigma_{\mathcal{C}}\left(\delta y^k\right) \leq \epsilon_{\mathrm{p,inf}}, \tag{28b}$$

holds and a dual infeasibility certificate if

$$\left\|P\delta x^k\right\|_\infty / \left\|\delta x^k\right\|_\infty \leq \epsilon_{\mathrm{d,inf}}, \tag{29a}$$

$$q^\top \delta x^k / \left\|\delta x^k\right\|_\infty \leq \epsilon_{\mathrm{d,inf}}, \tag{29b}$$

$$A\delta x^k + v \in \mathcal{C}^\infty, \tag{29c}$$

$$\text{with } \|v\|_\infty \leq \epsilon_{\mathrm{d,inf}} \left\|\delta x^k\right\|_\infty,$$

holds.

4 Chordal Decomposition

As noted in Section 3.3, for large SDPs the eigendecomposition in the projection step (18) 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 *chordal* or can be chordal extended, Agler's [Agl+88] and Grone's [Gro+84] 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 of each other, further performance improvement can be achieved by carrying them out in parallel. The approach is similar to [Zhe+17]. 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.

4.1 Graph preliminaries

Consider the undirected graph $G(V, E)$ with vertices $V = \{1, \dots, n\}$ 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*. 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 [V+15].

4.2 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

$$\mathbb{S}^n(E, 0) \coloneqq \{S \in \mathbb{S}^n \mid S_{i,j} = S_{j,i} \neq 0 \text{ if } (i, j) \in E\}, \tag{30}$$

and positive semidefinite sparse symmetric matrices as

$$\mathbb{S}^n_+(E, 0) \coloneqq \{S \in \mathbb{S}^n(E, 0) \mid S \succeq 0\}. \tag{31}$$

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 the vertices in each C_ℓ are sorted in the natural way. Define the matrix $T_\ell \in \mathbb{R}^{|C_\ell| \times n}$ for clique C_ℓ as:

$$(T_\ell)_{i,j} \coloneqq \begin{cases} 1, & \text{if } C_\ell(i) = j \\ 0, & \text{otherwise.} \end{cases} \tag{32}$$

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

Theorem 1 (Agler's theorem [Agl+88]). *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*

$$S = \sum_{\ell=1}^{p} T_\ell^\top S_\ell T_\ell. \tag{33}$$

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 clique C_ℓ. The vectorized form of (33) is given by:

$$s = \sum_{\ell=1}^{p} H_\ell^\top s_\ell, \quad \text{with } H_\ell = T_\ell \otimes T_\ell. \tag{34}$$

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

4.3 Decomposition of PSD constraints

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

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^\top P x + q^\top x \\ \text{subject to} & \sum_{i=1}^{m} \mathcal{A}_i x_i + S = B \\ & S \in \mathbb{S}^r_+, \end{array} \tag{35}$$

with symmetric matrices $\mathcal{A}_i, B, S \in \mathbb{S}^r$. Note that (35) can be transformed into (2) by setting $b = (B)$, $s = (S)$, $A = [(\mathcal{A}_1), \ldots, (\mathcal{A}_m)]$, and $\mathcal{K} = \mathcal{S}_+^r$. Assume that each matrix $\mathcal{A}_i$ and B has a sparsity pattern

$$\mathcal{A}_i \in \mathbb{S}^r(E_{\mathcal{A}_i}, 0) \text{ and } B \in \mathbb{S}^r(E_B, 0), \tag{36}$$

with edge sets $E_{\mathcal{A}_i}$ and E_B. The *aggregated sparsity pattern* of (35) is described by the graph $G(V, E)$ where E is:

$$E = E_{B_i} \cup E_{\mathcal{A}_1} \cup \cdots \cup E_{\mathcal{A}_m}. \tag{37}$$

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 (35) 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 (35), yielding

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^\top P x + q^\top x \\ \text{subject to} & \displaystyle\sum_{i=1}^m \mathcal{A}_i x_i + \sum_{\ell=1}^p T_\ell^\top S_\ell T_\ell = B \\ & S_\ell \in \mathbb{S}_+^{|C_\ell|}, \quad \ell = 1, \ldots, p. \end{array} \tag{38}$$

The vectorized form of (38) is given by:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^\top P x + q^\top x \\ \text{subject to} & Ax + \displaystyle\sum_{\ell=1}^p H_\ell^\top s_\ell = b \\ & s_\ell \in \mathcal{S}_+^{|C_\ell|}, \quad \ell = 1, \ldots, p. \end{array} \tag{39}$$

4.4 Decomposition as a preprocessing step

In order to perform the chordal decomposition before handing it to the solver, we transform (39) into the original solver format. The necessary transformation is achieved by rewriting the constraints of (39) as:

$$Ax + \mathcal{H}\xi = b, \quad \xi \in \bar{\mathcal{K}} \tag{40}$$

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

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\bar{x}^\top \begin{bmatrix} P & 0 \\ 0 & 0 \end{bmatrix} \bar{x} + \begin{bmatrix} q \\ 0 \end{bmatrix}^\top \bar{x} \\ \text{subject to} & \begin{bmatrix} A & \mathcal{H} \\ 0 & -I \end{bmatrix} \bar{x} + \bar{s} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \\ & \bar{s}_1 \in \{0\}^{r^2}, \ \bar{s}_2 \in \bar{\mathcal{K}}, \end{array} \tag{41}$$

where $\{0\}^{r^2}$ 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.

5 Implementation Details

We have implemented our algorithm in the Conic Operator Splitting Method (COSMO), an open-source package written in Julia [Bez⁺17]. The source code and documentation are available at

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

The graph algorithms to analyse the aggregate sparsity pattern, to compute a chordal embedding, and to find the maximal cliques have been implemented in a separate graph module without relying on external packages. This avoids dependencies and allows to tailor the algorithms exactly to the needs of the solver, which avoids overhead and unnecessary computations. The algorithms are based on the work by Vandenberghe and Andersen on chordal graphs [V⁺15], and similar to the implementations in the SparseCOLO [Fuj⁺09] and CHOMPACK [AV15] packages.

Since the matrices P and A in (2) are typically sparse, COSMO stores them in Compressed-Sparse-Column format by default. The SuiteSparse LDL solver is used to perform the LDL factorization in (16) on the sparse left hand side.

The projection step of the decomposed problem (39)

$$s_\ell^{k+1} = \Pi_{\mathcal{S}_+^{|\mathcal{C}_\ell|}}\left(\alpha\tilde{s}_\ell^{k+1} + (1-\alpha)s_\ell^k + \frac{1}{\rho}y_\ell^k\right) \quad \text{for } \ell = 1,\ldots,p, \tag{42}$$

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

COSMO implements a *Model* type that stores the problem data, the factorization of the linear system, and initial values for the primal and dual variables. This design choice allows factorization caching for parametric programs and warm starting of the optimisation problem. COSMO offers the user two interfaces to describe the constraints of the optimisation problem: a direct interface, and an interface to the modelling language JuMP [DHL17]. The JuMP interface connects the solver to the JuliaOpt ecosystem which provides flexible problem description and automatic problem reformulation.

The COSMO direct interface allows the user to define the constraints of the optimisation problem in a simple way. Each constraint is defined as a pair of an affine vector function $g(x) = Ax + b$ and a convex set. It further allows the user to specify the constraint on some elements of the primal variable x. A convex set is defined by its projection function. By default COSMO supports the zero cone, the nonnegative orthant, the second-order cone, and the PSD cone. Moreover, the object oriented nature of the interface allows the user to define their own convex sets assuming that they can provide a projection function that maps an input vector onto the convex set. For example consider a constraint on the ∞-norm of a variable $x \in \mathbb{R}^n$, which amounts to a projection onto the ∞-norm cone $\mathcal{K}_\infty$:

$$\|x\|_\infty \leq t \Leftrightarrow \begin{bmatrix} x \\ t \end{bmatrix} \in \mathcal{K}_\infty, \tag{43}$$

where $t \geq 0$. A user could implement this constraint in COSMO by providing the corresponding projection function. In this case the projection of $(\bar{x}, \bar{t}) \in \mathbb{R}^n \times \mathbb{R}_+$ onto $\mathcal{K}_\infty$ is given by the solution (x^*, t^*) to the following optimisation problem:

$$\begin{array}{ll} \text{minimize} & \|x - \bar{x}\|^2 + (t - \bar{t})^2 \\ \text{subject to} & \|x\|_\infty \leq t \end{array} \tag{44}$$

which can be solved by evaluating:

$$t^* = \max_{\ell \in (0,\ldots,n)} \frac{\bar{t} + \|\bar{x}\|_{[\ell]}}{1 + l}, \tag{45}$$

$$x^* = \max(\min(\bar{x}, t^*), -t^*). \tag{46}$$

A proof for (45)–(46) is given in the appendix.

6 Numerical Results

This section presents benchmark results of COSMO against the interior-point solvers MOSEK v8.1, SeDuMi v1.3, and the first-order ADMM solver SCS v2.0.2. To test the versatility of each solver we present numerical results for two test problem sets. Firstly, a nearest correlation matrix problem that can be formulated as a SDP with quadratic objective function. This problem class shows the advantages of a solver that handles quadratic functions directly. Secondly, we demonstrate the advantages of chordal decomposition on randomly generated SDPs where the constraints exhibit a block-arrow aggregate sparsity pattern. All the experiments were carried out on a MacBook with a $2.6\,\mathrm{GHz}$ Intel Core i5 CPU and $8\,\mathrm{GB}$ of RAM. We configured COSMO with absolute accuracy $\epsilon = 10^{-3}$, relaxation parameter $\alpha = 1.6$, and step sizes $\rho_0 = 0.1$ and $\sigma = 10^{-6}$. It should be noted that COSMO and SCS are configured with an accuracy of 10^{-3}, whereas the interior-point methods produce more accurate solutions and are configured with a tolerance of 10^{-6}. This is in line with other first-order solvers, see [Ste+18], [ODo+16], and [Zhe+17].

By default Julia uses OpenBLAS without multithreading for basic linear algebra operations. This must be kept in mind when comparing solver performance against solvers written in C or MATLAB, which often use BLAS libraries from the Intel MKL library that offers performance gains on Intel CPUs.

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 [Hig02]. 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:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\left\|X - C\right\|_F^2 \\ \text{subject to} & X_{ii} = 1, \quad i = 1,\dots,n \\ & X \in \mathbb{S}_+^n, \end{array} \tag{47}$$

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:

$$\begin{array}{ll} \text{minimize} & (1/2)(x^\top x - 2c^\top x + c^\top c) \\ \text{subject to} & \begin{bmatrix} E \\ -I \end{bmatrix} x + s = \begin{bmatrix} 1_{n\times 1} \\ 0_{n^2\times 1} \end{bmatrix} \\ & s \in \{0\}^n \times \mathcal{S}_+^n, \end{array} \tag{48}$$

with $c = \text{vec}(C) \in \mathbb{R}^{n^2}$ and $x = \text{vec}(X) \in \mathbb{R}^{n^2}$. $E \in \mathbb{R}^{n\times n^2}$ is a matrix that extracts the n diagonal entries X_{ii} from its vectorized 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. Figure 1 shows the benchmark results for increasing matrix dimension n. The first-order methods SCS and COSMO outperform the interior-point solvers. Furthermore,

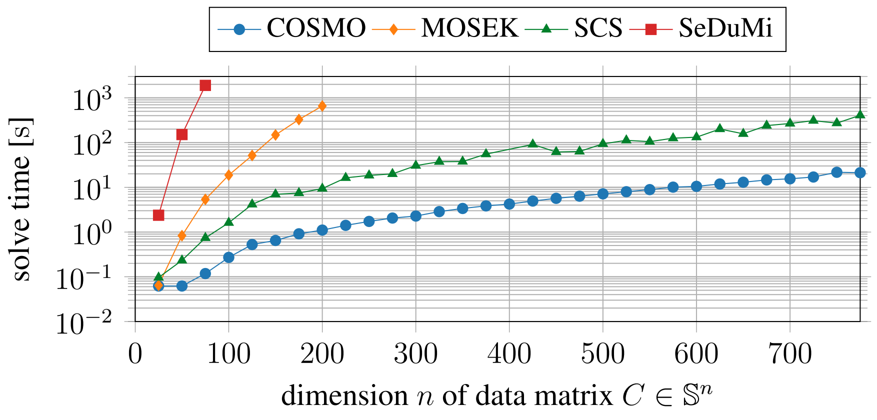

Figure 1: Solve time of benchmarked solvers for increasing problem size of nearest correlation matrix problems. The results for MOSEK and SeDuMi are shown until they exceeded the time limit of 60 min.

COSMO solves the problems faster than SCS as the problem dimension gets larger which is likely because SCS has to transform the quadratic cost term into an additional second-order cone constraint. This seems to impact the convergence of the underlying algorithm as COSMO needs on average 72.65 iterations to converge, whereas SCS needs 318 iterations.

Chordal block-arrow SDP To show the benefits of the chordal decomposition technique we consider randomly generated SDPs of the form (35) with a block-arrow aggregate sparsity pattern similar to test problems in [Zhe+17; ADV10]. Figure 2 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. In the following we study the effects of independently increasing

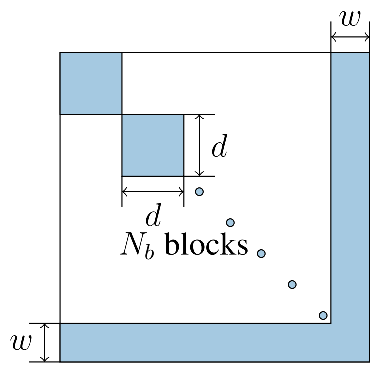

Figure 2: Parameters of block-arrow sparsity pattern.

the block size d and the number of blocks N_b. The parameters for the two test cases are:

- $N_b = 3, 6, 9, \dots, 30$, $d = 4$, $w = 4$, and $m = 20$.
- $d = 2, 3, 4, \dots, 11$, $N_b = 10$, $w = 4$, and $m = 20$.

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

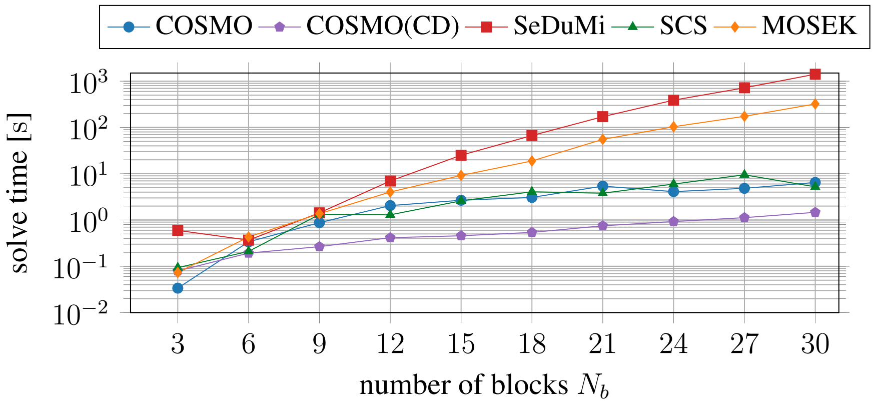

Figure 3: Solve time for increasing number of blocks N_b of block-arrow sparsity pattern.

As before the first-order methods show a substantial advantage over the interior-point solvers. Moreover COSMO with chordal decomposition outperforms the rest of the solvers. Comparing the results in Fig. 3 and 4, it can be seen that that chordal decomposition provides more benefit when the number of blocks is increased. This makes sense since a greater number of blocks leads to more smaller constraints that can be projected faster.

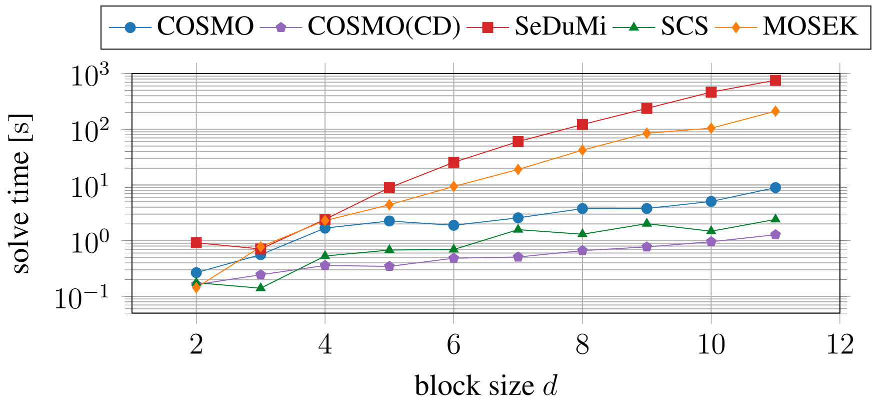

Figure 4: Solve time for increasing block size d of block-arrow sparsity pattern.

7 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 differenct 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.

Appendix

Proof. Projection onto ∞-norm cone $\mathcal{K}_\infty$
Rewrite (44) as a two-step minimisation problem:

$$\min_t \left[(t-\bar{t})^2 + \min_{\|x\|_\infty \le t} \|x - \bar{x}\|^2 \right]. \tag{49}$$

For fixed t, the inner optimisation problem amounts to a projection of $\bar{x}$ onto the ∞-norm box with radius t. (49) is rewritten solely in terms of t:

$$\min_t f(t), \quad \text{with: } f(t) := \left[(t-\bar{t})^2 + \sum_{i=1}^{n} \max(0, |\bar{x}_i| - t)^2 \right]. \tag{50}$$

The strictly convex piecewise quadratic function $f\colon \mathbb{R}_+ \to \mathbb{R}_+$ is differentiable with piecewise affine derivative:

$$\frac{df}{dt} = 2\left[(t-\bar{t}) + \sum_{i=1}^{n} \min(0, t - |\bar{x}_i|) \right]. \tag{51}$$

Note that $\frac{df}{dt}$ is concave, strictly increasing, and can be represented as the pointwise infimum of affine functions. To replace the $\min$-function consider an ordering operator τ such that:

$$|\bar{x}_{\tau(1)}| \ge \ldots \ge |\bar{x}_{\tau(n)}|. \tag{52}$$

Next, rewrite the derivative as the infimum of a collection of affine functions $\frac{df}{dt} = \inf_\ell q_\ell$ where $q_\ell\colon \mathbb{R} \to \mathbb{R}$ is defined as:

$$q_0(t) := 2\,(t-\bar{t}) \tag{53}$$

$$q_\ell(t) := 2\left[(t-\bar{t}) + \sum_{i=1}^{\ell} (t - |\bar{x}_{\tau(\ell)}|) \right]. \tag{54}$$

The optimal value t^* corresponds to the maximum zero of the affine functions q_ℓ, which yields the solution in (45). The optimal value x^* is then calculated as the projection onto a box of radius t^*. $\square$

References

[ADV10] M. S. Andersen, J. Dahl, and L. Vandenberghe. “Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones”. In: *Mathematical Programming Computation* 2.3-4 (2010), pp. 167–201.

[Agl+88] J. Agler et al. “Positive semidefinite matrices with a given sparsity pattern”. In: *Linear Algebra and its Applications* 107 (1988), pp. 101–149.

[AHO98] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton. “Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results”. In: *SIAM Journal on Optimization* 8.3 (1998), pp. 746–768.

[Ali95] F. Alizadeh. “Interior point methods in semidefinite programming with applications to combinatorial optimization”. In: *SIAM Journal on Optimization* 5.1 (1995), pp. 13–51.

[AV15] M. Andersen and L. Vandenberghe. *CHOMPACK: A Python package for chordal matrix computations*. 2015.

[Ban$^+$17] G. Banjac et al. “Infeasibility detection in the alternating direction method of multipliers for convex optimization”. In: *Preprint* (2017). URL: http://people.ee.ethz.ch/~gbanjac/pdfs/admm_infeas.pdf.

[BC11] H. Bauschke and P. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. 1st ed. Springer, 2011.

[Bez$^+$17] J. Bezanson et al. “Julia: A fresh approach to numerical computing”. In: *SIAM Review* 59.1 (2017), pp. 65–98.

[BHH01] P. Borm, H. Hamers, and R. Hendrickx. “Operations research games: A survey”. In: *Top* 9.2 (2001), p. 139.

[Boy$^+$11] S. Boyd et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends® in Machine Learning* 3.1 (2011), pp. 1–122.

[Boy$^+$17] S. Boyd et al. “Multi-period trading via convex optimization”. In: *Foundations and Trends® in Optimization* 3.1 (2017), pp. 1–76.

[Boy$^+$94] S. Boyd et al. *Linear matrix inequalities in system and control theory*. Vol. 15. Society for Industrial and Applied Mathematics, 1994.

[CV95] C. Cortes and V. Vapnik. “Support-vector networks”. In: *Machine Learning* 20.3 (1995), pp. 273–297.

[Dan63] G. Dantzig. “Linear programming and extensions”. In: *Princeton University Press, Princeton, NT* 19 (1963), p. 63.

[DHL17] I. Dunning, J. Huchette, and M. Lubin. “JuMP: A Modeling Language for Mathematical Optimization”. In: *SIAM Review* 59.2 (2017), pp. 295–320.

[EB92] J. Eckstein and D. P. Bertsekas. “On the Douglas-Rachford Splitting Method and the Proximal Point Algorithm for Maximal Monotone Operators”. In: *Mathematical Programming* 55.1 (1992), pp. 293–318.

[Eck89] J. Eckstein. “Splitting methods for monotone operators with applications to parallel optimization”. PhD thesis. Massachusetts Institute of Technology, 1989.

[Eck94] J. Eckstein. “Parallel alternating direction multiplier decomposition of convex programs”. In: *Journal of Optimization Theory and Applications* 80.1 (1994), pp. 39–62.

[Eve63] H. Everett. “Generalized Lagrange multiplier method for solving problems of optimum allocation of resources”. In: *Operations Research* 11.3 (1963), pp. 399–417.

[Fuj$^+$09] K. Fujisawa et al. “User’s manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems”. In: *Research Report B-453, Dept. of Math. and Comp. Sci. Japan, Tech. Rep.* (2009), pp. 152–8552.

[Fuk+01] M. Fukuda et al. "Exploiting sparsity in semidefinite programming via matrix completion I: General framework". In: *SIAM Journal on Optimization* 11.3 (2001), pp. 647–674.

[GB14] P. Giselsson and S. Boyd. "Diagonal scaling in Douglas-Rachford splitting and ADMM". In: *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE. 2014, pp. 5033–5039.

[GM75] 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.

[Gre97] A. Greenbaum. *Iterative methods for solving linear systems*. Vol. 17. Society for Industrial and Applied Mathematics, 1997.

[Gro+84] R. Grone et al. "Positive definite completions of partial Hermitian matrices". In: *Linear Algebra and its Applications* 58 (1984), pp. 109–124.

[Hel+96] C. Helmberg et al. "An interior-point method for semidefinite programming". In: *SIAM Journal on Optimization* 6.2 (1996), pp. 342–361.

[Hes69] M. R. Hestenes. "Multiplier and gradient methods". In: *Journal of Optimization Theory and Applications* 4.5 (1969), pp. 303–320.

[Hig02] N. J. Higham. "Computing the nearest correlation matrix-a problem from finance". In: *IMA Journal of Numerical Analysis* 22.3 (2002), pp. 329–343.

[Kar84] 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.

[MCL71] A. Miele, E. Cragg, and A. Levy. "Use of the augmented penalty function in mathematical programming problems, part 2". In: *Journal of Optimization Theory and Applications* 8.2 (1971), pp. 131–153.

[Meh92] S. Mehrotra. "On the implementation of a primal-dual interior point method". In: *SIAM Journal on Optimization* 2.4 (1992), pp. 575–601.

[Mie+71] A. Miele et al. "Use of the augmented penalty function in mathematical programming problems, part 1". In: *Journal of Optimization Theory and Applications* 8.2 (1971), pp. 115–130.

[Mie+72] A. Miele et al. "On the method of multipliers for mathematical programming problems". In: *Journal of Optimization Theory and Applications* 10.1 (1972), pp. 1–33.

[Mol+13] D. K. Molzahn et al. "Implementation of a large-scale optimal power flow solver based on semidefinite programming". In: *IEEE Transactions on Power Systems* 28.4 (2013), pp. 3987–3998.

[MOS17] MOSEK, Aps. *The MOSEK optimization toolbox for MATLAB manual. Version 8.0 (Revision 57)*. 2017.

[ODo+16] B. O'Donoghue et al. "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding". In: *Journal of Optimization Theory and Applications* 169.3 (June 2016), pp. 1042–1068.

[P G15] P. Giselsson and S. Boyd. "Metric selection in fast dual forward-backward splitting". In: *Automatica* 62 (2015), pp. 1–10.

[P$^+$14] N. Parikh, S. Boyd, et al. "Proximal algorithms". In: *Foundations and Trends® in Optimization* 1.3 (2014), pp. 127–239.

[Pow69] M. J. Powell. "A method for nonlinear constraints in minimization problems". In: *Optimization* (1969), pp. 283–298.

[Roc70] R. T. Rockafellar. *Convex analysis*. Princeton University Press, 1970.

[Roc76] R. T. Rockafellar. "Monotone operators and the proximal point algorithm". In: *SIAM Journal on Control and Optimization* 14.5 (1976), pp. 877–898.

[Rui01] D. Ruiz. *A scaling algorithm to equilibrate both rows and columns norms in matrices*. Tech. rep. Rutherford Appleton Laboratory, 2001.

[Ste$^+$18] B. Stellato et al. "OSQP: An Operator Splitting Solver for Quadratic Programs". In: *ArXiv e-prints* (Jan. 2018). arXiv: 1711.08013 [math.OC]. URL: https://arxiv.org/abs/1711.08013.

[Stu99] J. F. Sturm. "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 625–653.

[TTT99] K. Toh, M. J. Todd, and R. H. Tütüncü. "SDPT3 - A MATLAB software package for semidefinite programming, version 1.3". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 545–581.

[V$^+$15] L. Vandenberghe, M. S. Andersen, et al. "Chordal graphs and semidefinite optimization". In: *Foundations and Trends® in Optimization* 1.4 (2015), pp. 241–433.

[Van95] R. J. Vanderbei. "Symmetric Quasidefinite Matrices". In: *SIAM Journal on Optimization* 5.1 (1995), pp. 100–113.

[WN99] S. Wright and J. Nocedal. "Numerical optimization". In: *Springer Science* 35 (1999).

[Wol59] P. Wolfe. "The simplex method for quadratic programming". In: *Econometrica: Journal of the Econometric Society* (1959), pp. 382–398.

[Wri97] S. J. Wright. *Primal-dual interior-point methods*. Vol. 54. Society for Industrial and Applied Mathematics, 1997.

[Zhe$^+$17] Y. Zheng et al. "Chordal decomposition in operator-splitting methods for sparse semidefinite programs". In: *ArXiv e-prints* (2017). arXiv: 1707.05058 [math.OC]. URL: https://arxiv.org/abs/1707.05058.