Decomposition methods for large-scale semidefinite programs with chordal aggregate sparsity and partial orthogonality

Yang Zheng, Giovanni Fantuzzi, and Antonis Papachristodoulou

Abstract Many semidefinite programs (SDPs) arising in practical applications have useful structural properties that can be exploited at the algorithmic level. In this chapter, we review two decomposition frameworks for large-scale SDPs characterized by either *chordal aggregate sparsity* or *partial orthogonality*. Chordal aggregate sparsity allows one to decompose the positive semidefinite matrix variable in the SDP, while partial orthogonality enables the decomposition of the affine constraints. The decomposition frameworks are particularly suitable for the application of first-order algorithms. We describe how the decomposition strategies enable one to speed up the iterations of a first-order algorithm, based on the *alternating direction method of multipliers*, for the solution of the homogeneous self-dual embedding of a primal-dual pair of SDPs. Precisely, we give an overview of two structure-exploiting algorithms for semidefinite programming, which have been implemented in the open-source MATLAB solver CDCS. Numerical experiments on a range of large-scale SDPs demonstrate that the decomposition methods described in this chapter promise significant computational gains.

Y. Zheng
Department of Engineering Science, University of Oxford, Parks Road, Oxford, OX1 3PJ, U.K.
e-mail: yang.zheng@eng.ox.ac.uk

G. Fantuzzi
Department of Aeronautics, Imperial College London, South Kensington Campus, SW7 2AZ, U.K.
e-mail: gf910@ic.ac.uk

A. Papachristodoulou
Department of Engineering Science, University of Oxford, Parks Road, Oxford, OX1 3PJ, U.K.
e-mail: antonis@eng.ox.ac.uk

1 Introduction

Semidefinite programs (SDPs) are a type of convex optimization problems that arise in many fields, for example control theory, combinatorics, machine learning, operations research, and fluid dynamics [9, 13, 17, 45]. SDPs generalize other common types of optimization problems such as linear and second-order cone programs [11], and have attracted considerable attention because many nonlinear constraints admit numerically-tractable SDP reformulations or relaxations [41]. The standard primal and dual forms of an SDP are, respectively,

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

and

$$\begin{aligned} \max_{y,\, Z} \quad & \langle b, y\rangle \\ \text{subject to} \quad & Z + \sum_{y=1}^{m} y_i A_i = C, \\ & Z \in \mathbb{S}^n_+, \end{aligned} \tag{2}$$

where X is the primal variable, y and Z are the dual variables, and the vector $b \in \mathbb{R}^m$ and the matrices $C,\ A_1,\ \dots,\ A_m \in \mathbb{S}^n$ are given problem data. In (1), (2), and throughout this chapter, $\mathbb{R}^m$ denotes the usual m-dimensional Euclidean space, $\mathbb{S}^n$ is the space of $n \times n$ symmetric matrices, and $\mathbb{S}^n_+$ represents the cone of positive semidefinite (PSD) matrices. The notation $\langle \cdot, \cdot \rangle$ denotes the appropriate inner product: $\langle x, y\rangle = x^{\mathsf{T}} y$ for $x, y \in \mathbb{R}^m$ and $\langle X, Y\rangle = \mathrm{trace}(XY)$ for $X, Y \in \mathbb{S}^n$.

It is well-known that in theory SDPs can be solved in polynomial time using interior-point methods (IPMs). At the time of writing, however, these are only practical for small- to medium-sized problem instances [3,23]: Memory or CPU time constraints prevent the solution of (1)–(2) when n is larger than a few hundred and m is larger than a few thousand using a regular PC. Improving the scalability of current SDP solvers therefore remains an active area of research [4], with particular emphasis being put on taking advantage of structural properties pertaining to specific problem classes. This chapter gives an overview of promising recent developments that exploit two kinds of structural properties, namely chordal aggregate sparsity and partial orthogonality, in semidefinite programming.

Since the data b, C, $A_1, \dots, A_m$ are sparse in many large-scale SDPs encountered in applications [9], perhaps the most obvious approach to improve computational efficiency is to try and exploit this sparsity [16]. The main challenge in this respect is that the optimal solution X^* to (1) or the inverse of the optimal solution Z^* to (2) (required to compute the gradient and Hessian of the dual barrier function [5]) can be dense even when the problem data are extremely sparse. Nonetheless, one can take advantage of sparsity if the aggregate sparsity pattern of an SDP—that is, the union

of the sparsity pattern of the matrices C, A_1, ..., A_m—is *chordal*, or has a sparse *chordal extension* (precise definitions of these properties will be given in Section 2). In these cases, Grone's theorem [22] and Agler's theorem [1] enable the decomposition of the large PSD matrix variables in (1)–(2) into smaller PSD matrices, coupled by an additional set of affine constraints. Such a reformulation is attractive because common implementations of IPMs can handle multiple small PSD matrices very efficiently. This observation leads to the development of so-called *domain-* and *range-space* decomposition techniques [19, 25], which are implemented in the MATLAB package SparseCoLO [18].

One drawback of these sparsity-based decomposition methods is that the added equality constraints introduced by the application of Grone's and Agler's theorems can offset the benefit of working with smaller PSD cones. To overcome this problem, it has been suggested that chordal sparsity can be exploited directly at the algorithmic level to develop specialized interior-point solvers: Fukuda *et al.* developed a primal-dual path-following method for sparse SDPs [19]; Burer proposed a nonsymmetric primal-dual method using Cholesky factors of the dual variable Z and maximum determinant completion of the primal variable X [12]; Andersen *et al.* developed fast recursive algorithms for SDPs with chordal sparsity [5]. Another promising solution is to abandon IPMs altogether, and solve the decomposed SDP utilizing first-order algorithms instead. These only aim to achieve a solution of moderate accuracy, but scale more favourably than IPMs and can be implemented on highly-parallel computer architectures with relative ease. For these reasons, Sun *et al.* proposed a first-order operator splitting method for conic optimization with partially separable structure [37], and Madani *et al.* developed a highly-parallelizable first-order algorithm for sparse SDPs with inequality constraints [29]. Both approaches offer fast first-order algorithms built on Grone's and Agler's theorems [1, 22].

The second limitation of the decomposition methods of Refs. [19, 25] is that, of course, they can only be applied to SDPs whose aggregate sparsity patterns are chordal or admit a sparse chordal extension. One notable class of problems for which the data matrices are individually but not aggregately sparse is that of SDPs arising from sum-of-squares (SOS) programming. In this case, the aggregate sparsity pattern is *fully dense* (see, *e.g.*, [46, Section III-B] and [47, Section 3]). SOS programming is a powerful relaxation technique to handle NP-hard polynomial optimization problems [26, 34], with far-reaching applications in systems analysis and control theory [33]. While it is possible to tackle large-scale SOS programs using further relaxation techniques based on diagonally dominant matrices (see the DSOS and SDSOS techniques [2]), algorithms that enable their solution without introducing additional conservativeness are highly desired.

In this chapter, we describe two recent methods [47, 49] that address the two aforementioned disadvantages of the domain- and range-space decomposition techniques of Refs. [19, 25]. The first method [49] applies to SDPs with chordal aggregate sparsity patterns and is based on a decomposition framework for the PSD cone, which resembles that of Refs. [19, 25] but is more suitable for the application of fast first-order methods. The second algorithm [47] specializes in SDPs that enjoy a structural property that we call *partial orthogonality*: Loosely speaking, the data

matrices $A_1, \dots, A_m$ in (1)-(2) are mutually orthogonal up to a small submatrix (a precise definition will be given in Section 5). This property is inherent (but of course not exclusive) to SOS programs [47] and enables a decomposition of the affine constraints of the SDP, rather than of the PSD variable. More precisely, it leads to a diagonal plus low-rank representation of a certain matrix associated with the affine constraints. The computational backbone of both methods we describe is a variant of a classical first-order operator splitting method known as the *alternating direction method of multipliers* (ADMM), tailored to solve the homogeneous self-dual embedding of a primal-dual pair of conic programs [30].

The rest of this chapter is organized as follows. Section 2 briefly introduces the PSD matrix decomposition techniques based on chordal sparsity. The homogeneous self-dual embedding of the primal-dual pair of SDPs (1)–(2) and the ADMM algorithm of Ref. [30] are reviewed in Section 3. We present a fast ADMM algorithm for SDPs with chordal sparsity in Section 4, and another fast ADMM algorithm for SDPs with partial orthogonality in Section 5. Section 6 describes the MATLAB solver CDCS (Cone Decomposition Conic Solver) [48], which includes an implementation of both algorithms, and reports the results of numerical experiments on a set of benchmark problems. Concluding remarks and suggestions for future research are given in Section 7.

2 Matrix decomposition using chordal sparsity

As mentioned in the introduction, one of the decomposition methods for SDPs (1)–(2) described in this chapter exploits the sparsity of the problem data. One key to this approach is the description of sparse matrices using graphs, so here we review some essential concepts from graph theory and their relation to sparse matrices.

2.1 Essential notions from graph theory

An undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is defined as a set of nodes $\mathcal{V} = \{1, 2, \dots, n\}$ plus a set of edges (connections between nodes) $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. For example, Fig. 1(a) illustrates a graph with nodes $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$ and edges $\mathcal{E} = \{(1,2), (1,3), (1,6), (2,4), (2,5), (3,4), (3,5)\}$. A subset of nodes $\mathcal{C} \subseteq \mathcal{V}$ is called a *clique* if any pair of distinct nodes in $\mathcal{C}$ is connected by an edge, *i.e.*, $(i,j) \in \mathcal{E}, \forall\, i, j \in \mathcal{C}$. If a clique $\mathcal{C}$ is not a subset of any other clique, we refer to it as a *maximal clique*. In Fig. 1(b), there are four maximal cliques: $\mathcal{C}_1 = \{1,2,3\}, \mathcal{C}_2 = \{2,3,5\}, \mathcal{C}_3 = \{2,3,4\}$ and $\mathcal{C}_4 = \{1,6\}$. We denote the number of nodes in a clique $\mathcal{C}$ by $|\mathcal{C}|$, while $\mathcal{C}(i)$ indicates its i-th node when sorted in the natural ordering.

A cycle of length k is a subset of nodes $\{v_1, v_2, \dots, v_k\} \subseteq \mathcal{V}$ such that $(v_k, v_1) \in \mathcal{E}$ and $(v_i, v_{i+1}) \in \mathcal{E}$ for $i = 1, \dots, k-1$. Any additional edge joining two nonconsecutive nodes in a cycle is called a *chord*, and an undirected graph

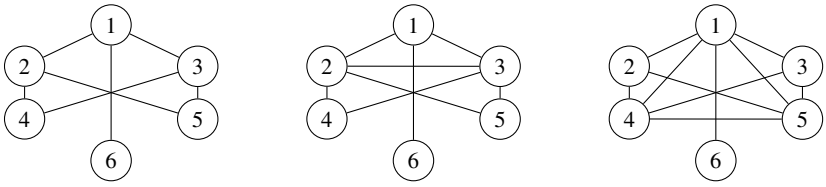


Fig. 1 (a) A simple nonchordal graph: The cycles $\{1,2,4,3\}$, $\{1,2,5,3\}$, and $\{2,4,3,5\}$ have length four but no chords. (b) A chordal extension of the nonchordal graph in (a), obtained by adding edge $(2,3)$. (c) A different chordal extension of the nonchordal graph in (a), obtained by adding edges $(1,4)$, $(1,5)$, and $(4,5)$.

$\mathcal{G}$ is called *chordal* if every cycle of length greater than three has one chord. Of course, given a nonchordal graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, one can always construct a chordal graph $\mathcal{G}'(\mathcal{V},\mathcal{E}')$ by adding suitable edges to the original edge set $\mathcal{E}$. This process, known as *chordal extension*, is not unique as illustrated by Fig. 1. The graph in Fig. 1(a) is not chordal: Adding the edge $(2,3)$ results in the chordal graph depicted in Fig. 1(b), while adding edges $(1,4)$, $(1,5)$, and $(4,5)$ yields the chordal graph in Fig. 1(c). Note that computing the chordal extension with the minimum number of additional edges is NP-hard in general [43], but sufficiently good extensions can often be found efficiently using heuristic approaches [40]. More examples can be found in [40,49].

2.2 Graphs, sparse matrices, and chordal decomposition

Given an undirected graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, let $\mathcal{E}^* = \mathcal{E} \cup \{(i,i), \forall i \in \mathcal{V}\}$ be the extended set of edges to which all self loops have been added. This can be used to define the space of sparse matrices

$$\mathbb{S}^n(\mathcal{E},0) := \{X \in \mathbb{S}^n : (i,j) \notin \mathcal{E}^* \Rightarrow X_{ij} = 0\}. \tag{3}$$

Note that this definition does not preclude $X_{ij} = 0$ if $(i,j) \in \mathcal{E}^*$, so a matrix $X \in \mathbb{S}^n(\mathcal{E},0)$ can be sparser than allowed by the underlying graph representation.

Two subsets of this sparse matrix space will be useful in the following. The first is the subspace of positive semidefinite sparse matrices,

$$\mathbb{S}^n_+(\mathcal{E},0) := \{X \in \mathbb{S}^n(\mathcal{E},0) : X \in \mathbb{S}^n_+\} \equiv \mathbb{S}^n(\mathcal{E},0) \cap \mathbb{S}^n_+. \tag{4}$$

The second is the subspace of sparse matrices that can be completed into a positive semidefinite matrix by filling in the zero entries (sometimes called *partial semidefinite matrices*), defined according to

$$\mathbb{S}^n_+(\mathcal{E},?) := \{X \in \mathbb{S}^n(\mathcal{E},0) : \exists M \in \mathbb{S}^n_+ \mid (i,j) \in \mathcal{E}^* \Rightarrow M_{ij} = X_{ij}\}. \tag{5}$$

Note that $\mathbb{S}^n_+(\mathcal{E},?)$ can be viewed as the projection of the space of positive semidefinite matrices $\mathbb{S}^n_+$ onto $\mathbb{S}^n(\mathcal{E},0)$ with respect to the Frobenius norm (the natural norm induced by the trace inner product). Moreover, it is not difficult to check that for any graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ the sets $\mathbb{S}^n_+(\mathcal{E},?)$ and $\mathbb{S}^n_+(\mathcal{E},0)$ are a pair of dual cones with respect to the ambient space $\mathbb{S}^n(\mathcal{E},0)$ [40].

When the graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ that defines the space $\mathbb{S}^n(\mathcal{E},0)$ is chordal, we say that $X \in \mathbb{S}^n(\mathcal{E},0)$ has a chordal sparsity pattern. In this case, the cones $\mathbb{S}^n_+(\mathcal{E},?)$ and $\mathbb{S}^n_+(\mathcal{E},0)$ enjoy a particularly useful property: Membership to the cone can be expressed in terms of PSD constraints on the submatrices corresponding to the maximal cliques of the underlying graph. Replacing the condition of cone membership with this equivalent set of PSD constraints is known as *chordal decomposition*. To make this concept precise, we will need some further notation: Given any maximal clique $\mathcal{C}$ of $\mathcal{G}(\mathcal{V},\mathcal{E})$, consider the matrix $E_{\mathcal{C}} \in \mathbb{R}^{|\mathcal{C}|\times n}$ with entries defined by

$$(E_{\mathcal{C}})_{ij} = \begin{cases} 1, & \text{if } \mathcal{C}(i) = j, \\ 0, & \text{otherwise.} \end{cases}$$

For any matrix $X \in \mathbb{S}^n(\mathcal{E},0)$, the submatrix corresponding to the maximal clique $\mathcal{C}$ can therefore be represented as $E_{\mathcal{C}} X E_{\mathcal{C}}^{\mathsf{T}} \in \mathbb{S}^{|\mathcal{C}|}$. Moreover, note that the operation $E_{\mathcal{C}}^{\mathsf{T}} Y E_{\mathcal{C}}$ "inflates" a $|\mathcal{C}| \times |\mathcal{C}|$ matrix Y into a sparse $n \times n$ matrix with nonzero entries in the rows/columns specified by the indices of the clique $\mathcal{C}$.

We can now state the conditions for membership to the cones $\mathbb{S}^n_+(\mathcal{E},?)$ and $\mathbb{S}^n_+(\mathcal{E},0)$ when the sparsity pattern is chordal more precisely using Grone's [22] and Agler's [1] theorems. These have historically been proven individually, but can be derived from each other using duality [40].

Theorem 1 (Grone's theorem [22]). *Let $\mathcal{G}(\mathcal{V},\mathcal{E})$ be a chordal graph with maximal cliques $\{\mathcal{C}_1,\mathcal{C}_2,\ldots,\mathcal{C}_p\}$. Then, $X \in \mathbb{S}^n_+(\mathcal{E},?)$ if and only if $X_k := E_{\mathcal{C}_k} X E_{\mathcal{C}_k}^{\mathsf{T}} \in \mathbb{S}^{|\mathcal{C}_k|}_+$ for all $k = 1, \ldots, p$.*

Theorem 2 (Agler's theorem [1]). *Let $\mathcal{G}(\mathcal{V},\mathcal{E})$ be a chordal graph with maximal cliques $\{\mathcal{C}_1,\mathcal{C}_2,\ldots,\mathcal{C}_p\}$. Then, $Z \in \mathbb{S}^n_+(\mathcal{E},0)$ if and only if there exist matrices $Z_k \in \mathbb{S}^{|\mathcal{C}_k|}_+$ for $k = 1, \ldots, p$ such that $Z = \sum_{k=1}^p E_{\mathcal{C}_k}^{\mathsf{T}} Z_k E_{\mathcal{C}_k}$.*

3 Homogeneous self-dual embedding and ADMM

As anticipated in the introduction, the computational engine of the scalable SDP solvers described in this chapter is a specialized ADMM algorithm for the solution of the homogeneous self-dual embedding of conic programs. To make this chapter self-contained, in this section we review the main ideas behind the homogeneous self-dual embedding of SDPs and summarize the ADMM algorithm for its solution developed in [30].

3.1 Homogeneous self-dual embedding

An elegant method to solve the primal-dual pair of SDPs (1)–(2) that is also able to detect primal or dual infeasibility is to embed it into a homogeneous self-dual feasibility problem [44]. Solving the latter amounts to finding a nonzero point in the intersection of a convex cone and an affine space. Once such a nonzero point is found, one can either recover an optimal solution for the primal-dual pair (1)–(2), or construct a certificate of primal or dual infeasibility (more on this is outlined below).

To formulate the homogeneous self-dual embedding of (1)–(2), it is convenient to consider their vectorized form. Specifically, let $\mathrm{vec} : \mathbb{S}^n \to \mathbb{R}^{n^2}$ be the operator that maps a matrix to the stack of its columns[1], and define the vectorized data

$$c := \mathrm{vec}(C), \quad A := \left[\mathrm{vec}(A_1) \ \ldots \ \mathrm{vec}(A_m)\right]^{\mathsf{T}}. \tag{6}$$

It is assumed that $A_1, \ldots, A_m$ are linearly independent matrices, and consequently that A has full row rank. Upon defining the set of vectorized PSD matrices as

$$\mathcal{S} := \{x \in \mathbb{R}^{n^2} : \mathrm{vec}^{-1}(x) \in \mathbb{S}^n_+\}, \tag{7}$$

the primal SDP (1) can be rewritten as

$$\begin{array}{ll} \min\limits_{x} & c^{\mathsf{T}} x \\ \text{subject to} & Ax = b, \\ & x \in \mathcal{S}, \end{array} \tag{8}$$

while the dual SDP (2) becomes

$$\begin{array}{ll} \max\limits_{y,z} & b^{\mathsf{T}} y \\ \text{subject to} & z + A^{\mathsf{T}} y = c, \\ & z \in \mathcal{S}. \end{array} \tag{9}$$

When strong duality holds, an optimal solution of (8)–(9) or a certificate of infeasibility can be recovered from any non-zero solution of the homogeneous linear system

$$\begin{bmatrix} z \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 & -A^{\mathsf{T}} & c \\ A & 0 & -b \\ -c^{\mathsf{T}} & b^{\mathsf{T}} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}, \tag{10}$$

provided that it also satisfies

[1] We use the vec operator for both conceptual and notational simplicity. Practical implementation of the methods described in this chapter should take advantage of the symmetry of all matrices. This can be done using symmetric vectorization through operator svec, which maps a matrix $X \in \mathbb{S}^n$ to the vector $\mathrm{svec}(X) := (X_{11}, \sqrt{2}X_{21}, \ldots, \sqrt{2}X_{n1}, X_{22}, \sqrt{2}X_{32}, \ldots, X_{nn}) \in \mathbb{R}^{n(n+1)/2}$. All statements made in this chapter hold when vec is replaced by svec.

$$(x, y, \tau) \in \mathcal{S} \times \mathbb{R}^m \times \mathbb{R}_+, \qquad (z, s, \kappa) \in \mathcal{S} \times \{0\}^m \times \mathbb{R}_+. \tag{11}$$

Note that the linear system (10) embeds the optimality conditions of (8)–(9): When $\tau = 1$ and $\kappa = 0$ the linear system (10) reduces to the KKT conditions

$$Ax = b, \;\; x \in \mathcal{S}, \;\; z + A^\mathsf{T} y = c, \;\; z \in \mathcal{S}, \;\; c^\mathsf{T} x = b^\mathsf{T} y. \tag{12}$$

(These are necessary and sufficient optimality conditions under the assumption of strong duality.)

In addition, τ and κ are requied to be nonnegative and complementary variables, *i.e.*, at most one of them is nonzero [44]. If $\tau > 0$ and $\kappa = 0$ the point

$$x^* = \frac{x}{\tau}, \quad y^* = \frac{y}{\tau}, \quad z^* = \frac{z}{\tau},$$

satisfies the KKT conditions (12) and is a primal-dual optimal solution of (8)–(9). If $\tau = 0$ and $\kappa > 0$, instead, a certificate of primal or dual infeasibility can be constructed depending on the values of $c^\mathsf{T} x$ and $b^\mathsf{T} y$. We refer the interested reader to [30] and references therein for more details.

For notational convenience, we finally rewrite the linear system (10) and the conditions in (11) in a compact form by defining the vectors and matrix

$$u := \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}, \quad v := \begin{bmatrix} z \\ s \\ \kappa \end{bmatrix}, \quad Q := \begin{bmatrix} 0 & -A^\mathsf{T} & c \\ A & 0 & -b \\ -c^\mathsf{T} & b^\mathsf{T} & 0 \end{bmatrix}, \tag{13}$$

and the cones

$$\mathcal{K} := \mathcal{S} \times \mathbb{R}^m \times \mathbb{R}_+, \qquad \mathcal{K}^* := \mathcal{S} \times \{0\}^m \times \mathbb{R}_+. \tag{14}$$

Then, an optimal point for (8)–(9) or a certificate of infeasibility can be recovered from a nonzero solution of the homogeneous self-dual feasibility problem

$$\begin{aligned} \text{find} \quad & (u, v) \\ \text{subject to} \quad & v = Qu, \\ & (u, v) \in \mathcal{K} \times \mathcal{K}^*. \end{aligned} \tag{15}$$

3.2 A tailored ADMM algorithm

It has been shown in Ref. [30] that problem (15) can be solved with a simplified version of the classical ADMM algorithm (see *e.g.*, [10]) by virtue of its self-dual character. The k-th iteration of this tailored ADMM algorithm consists of the following three steps:

$$w^{(k)} = (I + Q)^{-1} \left(u^{(k-1)} + v^{(k-1)} \right), \tag{16a}$$

$$u^{(k)} = \mathbb{P}_{\mathcal{K}} \left(w^{(k)} - v^{(k-1)} \right), \tag{16b}$$

$$v^{(k)} = v^{(k-1)} - w^{(k)} + u^{(k)}. \tag{16c}$$

Here and in the following $\mathbb{P}_{\mathcal{K}}$ denotes the projection onto the cone $\mathcal{K}$ and the superscript (k) indicates that a variable has been fixed to its value after the k-th iteration.

Since the last step is computationally trivial, practical implementations of the algorithm require an efficient computation of (16a) and (16b). An efficient C implementation that handles generic conic programs with linear, second-order, semidefinite and exponential cones is available in the solver SCS [31].

In this chapter, we show that when one is interested in solving SDPs with chordal aggregate sparsity or partial orthogonality (which are very common in certain practical applications), the computational efficiency of (16a) and (16b) can be improved further. In particular, chordal sparsity allows one to speed up the conic projection (16b) because one replaces a large PSD cone with smaller ones. Following Refs. [49–51], we refer to this procedure as *cone decomposition*. Partial orthogonality, instead, can be exploited to reduce the size of matrix to be inverted (or factorized) when solving the linear system (16a). Since (16a) can be interpreted as a projection onto the affine constraints of (8)–(9), in this chapter we will slightly abuse terminology and say that partial orthogonality allows an *affine decomposition*.

Remark 1. Recall from (13) that $u = \left(\begin{smallmatrix} x \\ y \\ \tau \end{smallmatrix}\right)$ and $v = \left(\begin{smallmatrix} z \\ s \\ \kappa \end{smallmatrix}\right)$ in (15), and that x and z are vectorized symmetric matrices. It is not difficult to check that if the corresponding entries in the initial guesses $u^{(0)}$ and $v^{(0)}$ for algorithm (16a)–(16c) are vectorized symmetric matrices, then the same is true for all subsequent iterates $w^{(k)}$, $u^{(k)}$, and $v^{(k)}$, $k = 1, 2, \ldots$. Indeed, steps (16b) and (16c) preserve the vectorized matrix structure provided that the vector $w^{(k)}$ obtained with (16a) possesses it. Since step (16a) is equivalent the solution of a linear system of the form

$$\begin{bmatrix} I & -A^{\mathsf{T}} & c \\ A & I & -b \\ -c^{\mathsf{T}} & b^{\mathsf{T}} & I \end{bmatrix} \begin{bmatrix} w_x \\ w_y \\ w_\tau \end{bmatrix} = \begin{bmatrix} \xi_x \\ \xi_y \\ \xi_\tau \end{bmatrix},$$

where ξ_x is a vectorized symmetric matrix, one needs to verify that w_x is so too. Now, the first block of equations implies that

$$w_x = A^{\mathsf{T}} w_y - c w_\tau + \xi_x,$$

which—recalling the definitions of c and A from (6)—is satisfied if and only if

$$\begin{aligned} \operatorname{vec}^{-1}(w_x) &= \operatorname{vec}^{-1}(\xi_x) - \operatorname{vec}^{-1}(c w_\tau) + \operatorname{vec}^{-1}(A^{\mathsf{T}} w_y) \\ &= \operatorname{vec}^{-1}(\xi_x) - w_\tau C + \sum_{i=1}^{m} (w_y)_i A_i. \end{aligned}$$

In the last expression, C, A_1, …, A_m are the symmetric data matrices of the original SDP and $(w_y)_i$ denotes the i-th entry of the vector w_y. Thus, the vector w_x must be a vectorized symmetric matrix, as required.

Remark 2. Algorithm (16a)–(16c) is not the only one available to solve the convex feasibility problem (15), and other methods could be employed. In particular, one could utilise other operator-splitting first-order methods, including Douglas-Rachford iterations [27], Spingarn's method [35], and Dykstra's method [6]. We have not tried these, but in light of the equivalence between ADMM and many operator-splitting methods [10], it is not unreasonable to expect that the ADMM-based algorithm (16a)–(16c) will perform at least as efficiently as many other alternatives.

4 Cone decomposition in sparse SDPs

Let us consider the case in which SDPs (1) and (2) have an aggregate sparsity pattern defined by the graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, meaning that

$$C \in \mathbb{S}^n(\mathcal{E},0) \quad \text{and} \quad A_i \in \mathbb{S}^n(\mathcal{E},0), \quad i=1,\dots,m.$$

Without loss of generality, we assume that $\mathcal{G}(\mathcal{V},\mathcal{E})$ is chordal with a set of maximal cliques $\mathcal{C}_1,\mathcal{C}_2,\dots,\mathcal{C}_p$ (for sparse nonchordal SDPs, we assume that a suitable sparse chordal extension has been found).

Aggregate sparsity implies that the dual variable Z in (2) satisfies $Z \in \mathbb{S}^n(\mathcal{E},0)$. As for the primal SDP (1), although the variable X can be dense, only the entries X_{ij} defined by the extended edge set $\mathcal{E}^*$ appear in the equality constraints and the cost function, while the remaining entries only ensure that X is PSD. Consequently, it suffices to consider $X \in \mathbb{S}^n_+(\mathcal{E},?)$. We can then apply Theorems 1 and 2 to rewrite (1) and (2), respectively, as

$$\begin{aligned}
\min_{X,X_1,\dots,X_p} \quad & \langle C, X\rangle \\
\text{subject to} \quad & \langle A_i, X\rangle = b_i, && i=1,\dots,m,\\
& X_k = E_{\mathcal{C}_k} X E_{\mathcal{C}_k}^{\mathsf{T}}, && k=1,\dots,p,\\
& X_k \in \mathbb{S}_+^{|\mathcal{C}_k|}, && k=1,\dots,p,
\end{aligned} \tag{17}$$

and

$$\begin{aligned}
\max_{y,Z_1,\dots,Z_p,V_1,\dots,V_p} \quad & \langle b, y\rangle \\
\text{subject to} \quad & \sum_{k=1}^{p} E_{\mathcal{C}_k}^{\mathsf{T}} V_k E_{\mathcal{C}_k} + \sum_{i=1}^{m} y_i A_i = C,\\
& Z_k = V_k, && k=1,\dots,p,\\
& Z_k \in \mathbb{S}_+^{|\mathcal{C}_k|}, && k=1,\dots,p.
\end{aligned} \tag{18}$$

We refer to (17) and (18) as the *cone decomposition* of a primal-dual pair of SDPs with chordal sparsity. Note that this cone decomposition is similar to the domain- and range-space decompositions developed in [19] but for one key feature: We do not eliminate the variables X in (17) and we introduce slack variables V_k, $k = 1, 2, \ldots, p$ in (18). This is essential if the conic and the affine constraints are to be separated effectively when using operator-splitting algorithms; see [49–51] for more detailed discussions. Also, a standard argument based on the Lagrange function and Lagrange multipliers [11, Chapter 5] reveals that the decomposed problems (17) and (18) are the dual of each other. This fact, which may seem surprising at first, is a consequence of the duality between the original problems (1) and (2), and the more subtle dual relationship between Grone's and Agler's theorems (cf. Ref. [40]).

As in Section 3, to formulate and solve the HSDE of problems (17) and (18) we use vectorized variables. Precisely, we let

$$x := \mathrm{vec}(X),\ x_k := \mathrm{vec}(X_k),\ z_k := \mathrm{vec}(Z_k),\ v_k := \mathrm{vec}(V_k),\ k = 1,\ \ldots,\ p,$$

and define matrices

$$H_k := E_{\mathcal{C}_k} \otimes E_{\mathcal{C}_k}, \quad k = 1, \ldots, p,$$

such that

$$x_k = \mathrm{vec}(X_k) = \mathrm{vec}(E_k X E_k^{\mathsf{T}}) = H_k x.$$

In other words, the matrices $H_1,\ \ldots,\ H_p$ project x onto the subvectors $x_1,\ \ldots,\ x_p$, respectively. Moreover, we denote the constraints $X_k \in \mathbb{S}_+^{|\mathcal{C}_k|}$ by $x_k \in \mathcal{S}_k$ (the formal definition of $\mathcal{S}_k$ is analogous to that of the set $\mathcal{S}$; see (7) in Section 3.1). We then group the vectorized variables according to

$$\hat{x} = \begin{bmatrix} x \\ x_1 \\ \vdots \\ x_p \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} y \\ v_1 \\ \vdots \\ v_p \end{bmatrix}, \quad \hat{z} = \begin{bmatrix} 0 \\ z_1 \\ \vdots \\ z_p \end{bmatrix},$$

define $H^{\mathsf{T}} := \begin{bmatrix} H_1^{\mathsf{T}} & \ldots & H_p^{\mathsf{T}} \end{bmatrix}$ and $\hat{\mathcal{S}} := \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_p$, and augment the problem data matrices according to

$$\hat{c} = \begin{bmatrix} c \\ 0 \end{bmatrix}, \quad \hat{b} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad \hat{A} = \begin{bmatrix} A & 0 \\ H & -I \end{bmatrix}.$$

With these definition, we can rewrite (17) and (18) in the compact vectorized forms

$$\begin{array}{lll} \min\limits_{\hat{x}} & \hat{c}^{\mathsf{T}}\hat{x} & \\ \text{subject to} & \hat{A}\hat{x} = \hat{b}, & \\ & \hat{x} \in \mathbb{R}^{n^2} \times \hat{\mathcal{S}}, & \end{array} \qquad \begin{array}{lll} \max\limits_{\hat{y},\hat{z}} & \hat{b}^{\mathsf{T}}\hat{y} & \\ \text{subject to} & \hat{z} + \hat{A}^{\mathsf{T}}\hat{y} = \hat{c}, & \\ & \hat{z} \in \{0\}^{n^2} \times \hat{\mathcal{S}}. & \end{array} \tag{19a,b}$$

These are a standard pair of primal-dual conic programs, and can be solved using the ADMM algorithm (16). Of course, the same could be done for the original pair of SDPs, but at a higher computational cost as we shall now demonstrate.

First, consider step (16a). An apparent difficulty is that the size of matrix Q is much larger for the decomposed problem (19) than that for the original problem, due to the introduction of the variables X_k and V_k in (17) and (18). However, Q is also highly structured and sparse. Using block elimination and the matrix inversion lemma [11, Appendix C.4.3], it is shown in Ref. [50] that step (16a) for the decomposed problem requires a set of relatively inexpensive matrix-vector multiplications and the solution of a linear system of equations with coefficient matrix

$$I + A\left(I + \frac{1}{2}D\right)^{-1} A^{\mathsf{T}} \in \mathbb{S}^m, \tag{20}$$

where $D = \sum_{k=1}^{p} H_k^{\mathsf{T}} H_k$ is diagonal. Note that the matrix in (20) only depends on the problem data and, consequently, its preferred factorization can be computed and cached before iterating the ADMM algorithm. Applying the ADMM algorithm of Ref. [30] directly to (1)–(2) also requires the solution of a linear system of the same size. It is reasonable to assume that solving the linear system (including the factorization step) bears the most computational burden, while further matrix-vector products are comparatively inexpensive. Step (16a) therefore has the same leading-order cost irrespective of whether algorithm (16a)–(16c) is applied to the original pair of SDPs (1)–(2) or their decomposed counterparts (17)–(18).

Consider now the conic projection (16b). When the ADMM algorithm described in Section (3.2) is applied to the original pair of SDPs (1)–(2), the operator $\mathbb{P}_{\mathcal{K}}$ requires the projection of a (large) $n \times n$ matrix on the PSD cone. This can be achieved via an eigenvalue decomposition, which to leading order requires $O(n^3)$ flops. When applied to the decomposed problem (19), instead, one needs to project a vector onto the cone

$$\hat{\mathcal{K}} = \mathbb{R}^{n^2} \times \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_p \times \mathbb{R}^{m+n_d} \times \mathbb{R}_+,$$

where $n_d = \sum_{k=1}^{p} |\mathcal{C}_k|^2$ is the length of the vector v. Projecting onto the subcones $\mathbb{R}^{n^2}, \mathbb{R}^{m+n_d}, \mathbb{R}_+$ is trivial, and the computational burden rests mostly on the projections onto the vectorized PSD cones $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_p$. The size of each of these cones only depends on the size of the corresponding maximal clique of the chordal sparsity pattern of the problem data. Consequently, if the largest maximal clique is small compared to the original problem size n, each of the PSD projections is much less expensive computationally than projecting the original $n \times n$ matrix. Moreover, each projection is independent of the others, so the computation can be parallelized. Therefore, step (16b) can be carried out more efficiently when the ADMM is applied to the decomposed problem (19) instead of the original pair of SDPs (1)–(2).

In conclusion, SDPs with chordal aggregate sparsity can be solved very efficiently using the ADMM algorithm (16a)–(16c) thanks to the computational saving in the conic projection step (16b). Numerical results that demonstrate this in practice

will be presented in Section 6, and we refer the interested reader to Refs. [49–51] for more details.

5 Affine decomposition in SDPs with partial orthogonality

As anticipated in the introduction to this chapter, there are some large-scale SDPs whose aggregate sparsity patterns (after chordal extension if necessary) are almost full, so the chordal decomposition method presented in Section 4 brings little to no advantage. Notable examples are the SDPs arising from general SOS programming: In this case, the individual data matrices C, A_1, ..., A_m are extremely sparse, but their aggregate sparsity pattern is full. However, it is easily shown [47] that when SOS programs are formulated in the usual monomial basis the data matrices are partially orthogonal (this property is defined precisely below).

Motivated by applications in SOS programming, we therefore consider SDPs characterized by partial orthogonality and discuss how to improve the computational efficiency of the ADMM steps (16a)–(16c) for this class of SDPs. We will show that partial orthogonality allows computational improvements in step (16a). For reasons that will become apparent below, and with a slight abuse of terminology, we say that partial orthogonality enables an *affine decomposition* of (16a), which in some respects parallels the cone decomposition of (16b) allowed by the chordal sparsity.

Let us make these ideas more precise. Consider the coefficient matrix A in the vectorized problems (8)–(9). We say that an SDP satisfies partial orthogonality if there exists a column permutation matrix P such that $AP = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$ with $A_1 \in \mathbb{R}^{m\times t_1}$, $A_2 \in \mathbb{R}^{m\times t_2}$ and $A_2 A_2^{\mathsf{T}} = D$ a diagonal matrix. In this case,

$$AA^{\mathsf{T}} = APP^{\mathsf{T}}A^{\mathsf{T}} = \begin{bmatrix} A_1 & A_2 \end{bmatrix} \begin{bmatrix} A_1^{\mathsf{T}} \\ A_2^{\mathsf{T}} \end{bmatrix} = A_1 A_1^{\mathsf{T}} + D. \tag{21}$$

For the SDPs resulting from SOS representations, one usually has $t_1 \ll \min\{t_2, m\}$, so that the product AA^{T} is of the "diagonal plus low-rank" form. In this chapter, we refer to the partition (21) as *affine decomposition.*

Partial orthogonality can be exploited to gain significant computational savings in the ADMM step (16a). Recalling the definitions of Q, this step requires the solution of the linear system of equations

$$\begin{bmatrix} I & -A^{\mathsf{T}} & c \\ A & I & -b \\ -c^{\mathsf{T}} & b^{\mathsf{T}} & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}. \tag{22}$$

We denote

$$M := \begin{bmatrix} I & -A^{\mathsf{T}} \\ A & I \end{bmatrix}, \quad \zeta := \begin{bmatrix} c \\ -b \end{bmatrix}.$$

Then, by eliminating w_3 from the first and second block-equations in (22), we have

$$(M + \zeta\zeta^\mathsf{T}) \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} - \theta_3\zeta, \tag{23a}$$

$$w_3 = \theta_3 + c^\mathsf{T} w_1 - b^\mathsf{T} w_2. \tag{23b}$$

We then apply the matrix inversion lemma [11, Appendix C.4.3] to (23a), leading to

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \left[I - \frac{(M^{-1}\zeta)\zeta^\mathsf{T}}{1 + \zeta^\mathsf{T}(M^{-1}\zeta)} \right] M^{-1} \begin{bmatrix} \theta_1 - c\theta_3 \\ \theta_2 + b\theta_3 \end{bmatrix}. \tag{24}$$

The vector $(M^{-1}\zeta)/(1 + \zeta^\mathsf{T} M^{-1}\zeta)$ depends only on the problem data and can be computed before starting the ADMM algorithm, so multiplication by the first matrix on the right-hand side of (24) at each iteration can be implemented only using vector-vector operations. The core of the computation is therefore to solve a linear system of the form

$$\begin{bmatrix} I & -A^\mathsf{T} \\ A & I \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \end{bmatrix}. \tag{25}$$

Again, by eliminating σ_1 from the second block-equation in (25), we obtain

$$\sigma_1 = \hat{\theta}_1 + A^\mathsf{T}\sigma_2, \tag{26a}$$

$$(I + AA^\mathsf{T})\sigma_2 = -A\hat{\theta}_1 + \hat{\theta}_2. \tag{26b}$$

At this stage, we can use the property of partial orthogonality (21): There exists a diagonal matrix $J := I + D$ such that $I + AA^\mathsf{T} = J + A_1 A_1^\mathsf{T}$. In the context of typical SOS programs, we know that $A_1 \in \mathbb{R}^{m \times t_1}$ with $t_1 \ll m$. Therefore, it is convenient to apply the matrix inversion lemma again to (26b), resulting in

$$\begin{aligned} (I + AA^\mathsf{T})^{-1} &= (J + A_1 A_1^\mathsf{T})^{-1} \\ &= J^{-1} - J^{-1}A_1(I + A_1^\mathsf{T} J^{-1} A_1)^{-1} A_1^\mathsf{T} J^{-1}. \end{aligned}$$

Since J is diagonal, its inverse is trivial to compute. Then, σ_1 and σ_2 in (26) are available by solving a $t_1 \times t_1$ linear system with a coefficient matrix

$$I + A_1^\mathsf{T} J^{-1} A_1 \in \mathbb{S}^{t_1}, \tag{27}$$

plus relatively inexpensive matrix-vector, vector-vector, and scalar-vector operations. Furthermore, the matrix $I + A_1^\mathsf{T} J^{-1} A_1$ is the same at all iterations and its preferred factorization can be computed and cached before iterating steps (16a)–(16c). Once σ_1 and σ_2 have been computed, the solution of (22) can be recovered using vector-vector and scalar-vector operations.

In summary, for SDPs with partial orthogonality, we only need to invert or factorize a $t_1 \times t_1$ matrix shown in (27), in contrast to the usual $m \times m$ matrix (*e.g.*, $I + AA^\mathsf{T} \in \mathbb{R}^{m \times m}$, or see (20)). If $t_1 \ll m$, which is true for typical SOS pro-

grams, then the affine decomposition can yield significant computational saving in the ADMM step (16a). More details can be found in Ref. [47].

6 Numerical simulations

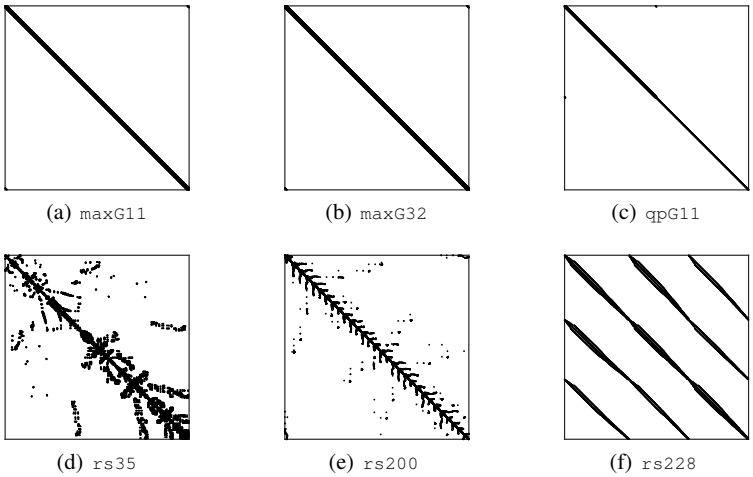
In this section we present numerical results for CDCS (cone decomposition conic solver), a MATLAB package that provides an efficient implementation of the cone decomposition and affine decomposition strategies described above. For the cone decomposition strategy, we show results on selected benchmark problems from SDPLIB [8] and some large and sparse SDPs with nonchordal sparsity patterns from [5]. For the affine decomposition strategy, we report results on the SDPs arising from SOS relaxations of constrained polynomial optimizations. We compared the results to the interior-point solver SeDuMi [36], and to the first-order solver SCS (the direct implementation was called) [31]. We called CDCS and SCS with termination tolerance 10^{-3} and limited the maximum number of iterations to $2\,000$. Default values were used for all other parameters. SeDuMi was called with its default parameters. All experiments were carried out on a PC with a 2.8 GHz Intel Core i7 CPU and 8GB of RAM.

6.1 CDCS

CDCS is the first open-source first-order conic solver that exploits chordal decomposition for the PSD cones and affine decomposition for the equality constraints. Infeasible problems can be detected thanks to the usage of homogeneous self-dual embedding. CDCS supports Cartesian products of the following standard cones: $\mathbb{R}^n$, non-negative orthant, second-order cones, and PSD cones. In order to save memory and increase computational efficiency, CDCS uses symmetric vectorization for all variables (cf. footnote 1 on page 7). The current implementation is written in MATLAB and can be downloaded from

`https://github.com/oxfordcontrol/CDCS` .

Different solver options are available. The default solver, `hsde`, implements the sparsity-exploiting algorithm described in Section 4. Changing the solver option to `sos` exploits partial orthogonality. We will refer to these two solver options as CDCS-hsde and CDCS-sos, respectively. CDCS also includes `primal` and `dual` options, which implement the primal-only and dual-only ADMM algorithms described in Ref. [49]. CDCS can be called directly from MATLAB's command window or through the optimization modeling toolboxes YALMIP [28] and SOSTOOLS [32].



(a) `maxG11` (b) `maxG32` (c) `qpG11`

(d) `rs35` (e) `rs200` (f) `rs228`

Fig. 2 Aggregate sparsity patterns of the large-scale sparse SDPs considered in this chapter; see Table 1 for the matrix dimensions.

Table 1 Summary of chordal decomposition for the chordal extensions of the large-scale sparse SDPs tested in this chapter. Problems are taken form Refs. [5, 8].

	`maxG11`	`maxG32`	`qpG11`	`rs35`	`rs200`	`rs228`
Original cone size, n	800	2 000	1 600	2 003	3 025	1 919
Affine constraints, m	800	2 000	800	200	200	200
Number of cliques, p	598	1 499	1 405	588	1635	783
Maximum clique size	24	60	24	418	102	92
Minimum clique size	5	5	1	5	4	3

6.2 Cone decomposition: the `hsde` *option*

To illustrate the benefits brought by the cone decomposition strategy, we consider three benchmark SDPs from SDPLIB [8] (`maxG11`, `maxG32` and `qpG11`) and three large-scale sparse SDPs from [5] (`rs35`, `rs200`, `rs228`). The SDPLIB problems are from practical applications (max cut problems and relaxations of box-constrained quadratic programs), while those of [5] are random SDPs with aggregate sparsity pattern coming from the University of Florida Sparse Matrix Collection [15].

Table 1 reports the dimensions of these problems and chordal decomposition details, while the aggregate sparsity patterns of these problems are illustrated in Fig. 2. Note that although the size of the PSD cone for problems `maxG32`, `qpG11`, `rs35`, `rs200` and `rs228` is over 1 000, the chordal extensions of the underlying sparsity patterns are very spare and the maximum clique size is much smaller than the original cone size. We expect CDCS-hsde to perform well on these problems since working with smaller PSD cones makes the conic projection step more efficient.

Table 2 Numerical results for described in Table 1.

	maxG11	maxG32	qpG11	rs35	rs200	rs228
SeDuMi						
Total CPU time (s)	49.7	954.3	458.2	1 120.9	4 003.7	1 291.2
Iterations	13	14	14	17	17	21
Objective	629.2	1 567.6	2 448.7	25.3	99.7	64.7
SCS						
Total CPU time (s)	91.3	2 414.6	1 070.4	2 425.5	9 259.1	2 187.9
Pre- time (s)	0.1	0.5	0.3	0.5	1.0	0.4
Iterations	1 080	†2 000	†2 000	†2 000	†2 000	†2 000
Objective	629.2	1 567.6	2 449.2	25.1	81.9	62.1
CDCS-hsde						
Total CPU time (s)	9.5	36.3	13.5	258.4	73.4	34.4
Pre- time (s)	2.2	12.8	2.2	5.2	8.2	3.4
Iterations	109	134	172	265	256	130
Objective	629.8	1 570.1	2 453.1	25.9	98.6	64.7

†: Maximum number of iterations reached.

Table 3 Average CPU time per ADMM iteration (in seconds) for the SDPs described in Table 1.

	maxG11	maxG32	qpG11	rs35	rs200	rs228
SCS	0.085	1.21	0.54	1.21	4.63	1.09
CDCS-hsde	0.067	0.18	0.066	0.96	0.25	0.24

The numerical results for these sparse SDPs are summarized in Tables 2 and 3. As shown in Table 2, CDCS-hsde was faster than either SeDuMi or SCS for all the problems we considered. In particular, CDCS-hsde is able to return an approximate solution of maxG11, maxG32, qpG11, rs200 or rs228 in less than $100s$, providing a speed up of approximately $10\times$, $67\times$, $79\times$, $126\times$ and $64\times$ over SCS for each problem respectively. Table 3 lists the average CPU time per iteration for CDCS-hsde and SCS, giving a fairer comparison of the performance of CDCS-hsde and SCS because any dependence on the exact stopping conditions used by each solver is removed. CDCS-hsde was faster than SCS for all problems, and—perhaps not unexpectedly—the computational gains become more significant when the size of the largest maximal clique is much smaller than the original PSD cone size. For problems maxG32, qpG11, rs200, and rs228 the average CPU time of CDCS-hsde improves on SCS by approximately $6.7\times$, $8.2\times$, $18.5\times$, and $4.5\times$, respectively. Finally, although first-order algorithms are only meant to provide solutions of moderate accuracy, the objective value returned by CDCS-hsde was always within 2.5% of the optimal value computed using SeDuMi. This gap may be considered acceptable in practical applications. Of course, as with all first-order algorithms accuracy could be improved further by setting tighter convergence tolerances at the expense of longer computation time.

Table 4 CPU time (in seconds) to solve the SDP relaxations of (28). N is the size of the largest PSD cone, m is the number of constraints, t is the size of the matrix factorized by CDCS-sos.

	Dimensions			CPU time (s)		
n	N	m	t	SeDuMi	SCS	CDCS-sos
10	66	1 000	66	2.7	0.4	0.4
15	136	3 875	136	162.8	2.4	2.1
20	231	10 625	120	2 730.8	10.4	8.2
25	351	23 750	351	**	45.8	28.8
30	496	46 375	496	**	156.1	85.3
35	666	82 250	666	**	455.4	222.2
40	861	135 750	861	**	1 182.8	526.6

**: The problem could not be solved due to memory limitations.

6.3 Affine decomposition: the `sos` *option*

To show the benefits brought by the exploitation of partial orthogonality, we next consider a series of SOS relaxations of constrained polynomial optimization problems (POPs). Note that partial orthogonality is an inherent structural property of SOS programs. In particular, we consider the constrained POP

$$\begin{aligned} \min_{x} \quad & \sum_{1\leq i<j\leq n} (x_i x_j + x_i^2 x_j - x_j^3 - x_i^2 x_j^2) \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i^2 \leq 1. \end{aligned} \tag{28}$$

We recast (28) into an SDP using the second Lasserre relaxation and GloptiPoly [24].

The numerical results are summarized in Tables 4 and 5. Table 4 reports the CPU time (in seconds) required by the solvers considered in this chapter to solve the SDP relaxations as the number of variables n in (28) was increased. As we can see, CDCS-sos is the fastest method for all the instances we tested thanks to partial orthogonality. For large-scale POPs ($n \geq 25$), the number of constraints in the resulting SDP is over $20\,000$, and the interior-point solver SeDuMi terminated prematurely due to RAM limitations. For all values on n the number t of non-orthogonal constraints is much smaller than the number m of constraints (see Table 4), so we expect CDCS-sos to perform well thanks to the affine decomposition described in Section 5. Indeed, for $n \geq 25$ CDCS-sos was approximately twice as fast as SCS. Finally, Table 5 shows that although first-order methods only aim to provide solutions of moderate accuracy, the optimal objective value returned by CDCS-sos and SCS was always within 0.5% of the high-accuracy optimal value computed using SeDuMi (when available). As remarked at the end of Section 6.2 such a small difference can be considered negligible in many practical applications.

Table 5 Terminal objective value by SeDuMi, SCS and CDCS-sos for the SDP relaxation of (28).

	SeDuMi	SCS		CDCS-sos	
n	Objective	Objective	Accuracy	Objective	Accuracy
10	-9.114	-9.124	0.10 %	-9.090	0.27 %
15	-14.118	-14.090	0.20 %	-14.138	0.14 %
20	-19.120	-19.165	0.24 %	-19.078	0.22 %
25	**	-24.174	—	-24.164	—
30	**	-29.087	—	-29.145	—
35	**	-34.053	—	-34.075	—
40	**	-39.146	—	-39.114	—

**: The problem could not be solved due to memory limitations.
—: No comparison is possible due to the lack of a reference accurate optimal value.

7 Conclusion

In this chapter, we have presented an overview of two recent approaches that exploit two structural properties of large-scale sparse semidefinite programs, namely *chordal aggregate sparsity* and *partial orthogonality*. Chordal sparsity is common to many SDPs encountered in practice, while partial orthogonality is an inherent characteristic of SOS programs. The first approach, developed in Refs. [49–51] and called *cone decomposition* in this chapter, allows one to decompose the original, large PSD cone into smaller ones. The second strategy, proposed in [47] and referred to as *affine decomposition* in this chapter, relies on a "diagonal plus low-rank" representation of a matrix related to the affine constraints of the SDP. By utilizing these two decompositions, the computational efficiency of the tailored ADMM algorithm for conic programs developed in Ref. [30] can be improved significantly. The proposed methods have been implemented in the MATLAB solver CDCS, and we have illustrated their efficiency on a set of benchmark test problems.

Looking ahead, further software development seems essential. Compared to the current version of CDCS, improvements at the implementation level are possible: Many steps of the algorithms presented in this chapter can be carried out in parallel, so one can take full advantage of distributed and/or parallel computing architectures. Algorithmic refinements to include acceleration techniques such as those proposed in Refs. [20, 39] also promise to improve the known poor convergence performance (in terms of numbers of iterations) of ADMM-based algorithms.

Finally, the computational gains obtained through the decomposition methods described in this chapter motivate the search for other structural properties that can be exploited. The identification of properties that characterize generic SDPs seems difficult, but one possible way forward is to focus on the special classes of problems arising in particular fields (*e.g.*, control theory, machine learning, combinatorics, fluid dynamics). In fact, it is reasonable to expect that discipline-specific structural properties exist, which can be exploited to develop advanced and efficient discipline-specific algorithms. This and the software developments mentioned above will be

essential to provide efficient and reliable computational tools to solve large-scale SDPs across all fields in which semidefinite programming has found applications.

Acknowledgements The authors would like to thank Prof. P. J. Goulart at the University of Oxford and Prof. A. Wynn at Imperial College London for their insightful comments. Y. Zheng is supported by the Clarendon Scholarship and the Jason Hu Scholarship. G. Fantuzzi was supported by an EPSRC studentship, Award Reference 1864077, and A. Papachristodoulou is supported by EPSRC grant EP/M002454/1.

References

1. J. Agler, W. Helton, S. McCullough, and L. Rodman. Positive semidefinite matrices with a given sparsity pattern. *Linear algebra Appl.*, 107:101–149, 1988.
2. A. A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization. *arXiv preprint arXiv:1706.02586*, 2017.
3. F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton. Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM J. Optim.*, 8(3):746–768, 1998.
4. M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe. Interior-point methods for large-scale cone programming. In *Optimization for machine learning*, pages 55–83. MIT Press, 2011.
5. M. S. Andersen, J. Dahl, and L. Vandenberghe. Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones. *Math. Program. Comput.*, 2(3-4):167–201, 2010.
6. H. H. Bauschke and J. M. Borwein. Dykstra's alternating projection algorithm for two sets. *Journal of Approximation Theory*, 79(3):418–443, 1994.
7. J. R. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.
8. B. Borchers. SDPLIB 1.2, a library of semidefinite programming test problems. *Optim. Methods Softw.*, 11(1-4):683–690, 1999.
9. S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.
10. S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends® Mach. Learn.*, 3(1):1–122, 2011.
11. S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
12. S. Burer. Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM J. Optim.*, 14(1):139–172, 2003.
13. S. I. Chernyshenko, P. J. Goulart, D. Huang, and A. Papachristodoulou. Polynomial sum of squares in fluid dynamics: a review with a look ahead. *Philos. Trans. R. Soc. A*, 372(2020):20130350, 2014.
14. T. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.
15. T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.
16. E. De Klerk. Exploiting special structure in semidefinite programming: A survey of theory and applications. *European Journal of Operational Research*, 201(1):1–10, 2010.
17. G. Fantuzzi and A. Wynn. Optimal bounds with semidefinite programming: An application to stress driven shear flows. *Phys. Rev. E*, 93(4):043308, 2016.
18. K. Fujisawa, S. Kim, M. Kojima, Y. Okamoto, and M. Yamashita. User's manual for SparseCoLO: Conversion methods for sparse conic-form linear optimization problems. Technical report, Research Report B-453, Tokyo Institute of Technology, Tokyo 152-8552, Japan, 2009.

19. M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: General framework. *SIAM J. Optim.*, 11(3):647–674, 2001.
20. P. Giselsson, M. Fält, and S. Boyd. Line search for averaged operator iteration. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 1015–1022. IEEE, 2016.
21. M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming. *Available: cvxr. com/cvx*, 2008.
22. R. Grone, C. R. Johnson, E. M. Sá, and H. Wolkowicz. Positive definite completions of partial hermitian matrices. *Linear Algebra Appl.*, 58:109–124, 1984.
23. C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. Optim.*, 6(2):342–361, 1996.
24. D. Henrion and J.-B. Lasserre. GloptiPoly: Global optimization over polynomials with MATLAB and SeDuMi. *ACM Tans. Math. Softw. (TOMS)*, 29(2):165–194, 2003.
25. S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion. *Math. Program.*, 129(1):33–68, 2011.
26. J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. Optim.*, 11(3):796–817, 2001.
27. P.-L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.
28. J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2005.
29. R. Madani, A. Kalbat, and J. Lavaei. ADMM for sparse semidefinite programming with applications to optimal power flow problem. In *Proc. 54th IEEE Conf. Decis. Control*, pages 5932–5939, Dec 2015.
30. B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *J. Optim. Theory Appl.*, 169(3):1042–1068, 2016.
31. B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 1.2.6. https://github.com/cvxgrp/scs, Apr. 2016.
32. A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. Parrilo. SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB. *arXiv preprint arXiv:1310.4716*, 2013.
33. A. Papachristodoulou and S. Prajna. A tutorial on sum of squares techniques for systems analysis. In *Am. Control Conf. (ACC)*, pages 2686–2700. IEEE, 2005.
34. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program. Ser. B*, 96(2):293–320, 2003.
35. J. E. Spingarn. Applications of the method of partial inverses to convex programming: decomposition. *Mathematical Programming*, 32(2):199–223, 1985.
36. J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, 11(1-4):625–653, 1999.
37. Y. Sun, M. S. Andersen, and L. Vandenberghe. Decomposition in conic optimization with partially separable structure. *SIAM J. Optim.*, 24(2):873–897, 2014.
38. R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing*, 13(3):566–579, 1984.
39. A. Themelis and P. Patrinos. SuperMann: a superlinearly convergent algorithm for finding fixed points of nonexpansive operators. *arXiv preprint arXiv:1609.06955*, 2016.
40. L. Vandenberghe and M. S. Andersen. Chordal graphs and semidefinite optimization. *Found. Trends® Optim.*, 1(4):241–433, 2014.
41. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.
42. Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented lagrangian methods for semidefinite programming. *Math. Program. Comput.*, 2(3-4):203–230, 2010.
43. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2:77–79, 1981.

44. Y. Ye, M. J. Todd, and S. Mizuno. An $\mathcal{O}\sqrt{n}l$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19(1):53–67, 1994.
45. R. Y. Zhang, C. Josz, and S. Sojoudi. Conic optimization theory: Convexification techniques and numerical algorithms. *arXiv preprint arXiv:1709.08841*, 2017.
46. Y. Zheng, G. Fantuzzi, and A. Papachristodoulou. Exploiting sparsity in the coefficient matching conditions in sum-of-squares programming using ADMM. *IEEE Control Systems Letters*, 1(1):80–85, 2017.
47. Y. Zheng, G. Fantuzzi, and A. Papachristodoulou. Fast ADMM for sum-of-squares programs using partial orthogonality. *arXiv preprint arXiv:1708.04174*, 2017.
48. Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. CDCS: Cone decomposition conic solver, version 1.1. https://github.com/OxfordControl/CDCS, Sept. 2016.
49. 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.
50. Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. Fast ADMM for homogeneous self-dual embedding of sparse SDPs. In *Proceedings of the 20th IFAC World Congress*, pages 8741–8746, 2017.
51. Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. Fast ADMM for semidefinite programs with chordal sparsity. In *American Control Conference (ACC)*, pages 3335–3340. IEEE, 2017.