A Survey of Recent Scalability Improvements for Semidefinite Programming with Applications in Machine Learning, Control, and Robotics

Anirudha Majumdar,[1] Georgina Hall,[2] and Amir Ali Ahmadi[3]

[1]Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, USA, 08544; email: ani.majumdar@princeton.edu
[2]Decision Sciences, INSEAD, Fontainebleau, France, 77300; email: georgina.hall@insead.edu
[3]Department of Operations Research and Financial Engineering, Princeton University, Princeton, USA, 08544; email: a_a_a_@princeton.edu

Xxxx. Xxx. Xxx. Xxx. YYYY. AA:1–33

https://doi.org/10.1146/((please add article doi))

Keywords

semidefinite programming, sum of squares programming, machine learning, control, robotics

Abstract

Historically, scalability has been a major challenge to the successful application of semidefinite programming in fields such as machine learning, control, and robotics. In this paper, we survey recent approaches for addressing this challenge including (i) approaches for exploiting structure (e.g., sparsity and symmetry) in a problem, (ii) approaches that produce low-rank approximate solutions to semidefinite programs, (iii) more scalable algorithms that rely on augmented Lagrangian techniques and the alternating direction method of multipliers, and (iv) approaches that trade off scalability with conservatism (e.g., by approximating semidefinite programs with linear and second-order cone programs). For each class of approaches we provide a high-level exposition, an entry-point to the corresponding literature, and examples drawn from machine learning, control, or robotics. We also present a list of software packages that implement many of the techniques discussed in the paper. Our hope is that this paper will serve as a gateway to the rich and exciting literature on scalable semidefinite programming for both theorists and practitioners.

Contents

1. INTRODUCTION

A *semidefinite program* (SDP) is an optimization problem of the form

$$\begin{aligned} \min_{X \in S_n} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_i X) = b_i, \ i = 1, \ldots, m \\ & X \succeq 0, \end{aligned} \tag{1}$$

where S_n denotes the set of $n \times n$ real symmetric matrices, Tr(.) stands for the trace of a matrix, $C, A_1, \ldots, A_m \in S_n$ and $b_1, \ldots, b_m \in \mathbb{R}$ are input data to the problem, and the notation $X \succeq 0$ denotes that the matrix X is constrained to be *positive semidefinite* (psd); i.e., to have nonnegative eigenvalues. The *dual* to Problem 1 takes the form

$$\begin{aligned} \max_{y \in \mathbb{R}^m} \quad & \sum_{i=1}^{m} y_i b_i \\ \text{s.t.} \quad & C - \sum_{i=1}^{m} y_i A_i \succeq 0 \end{aligned} \tag{2}$$

and will also feature in this paper.

SDPs have numerous fundamental applications throughout applied and computational mathematics. In combinatorics and discrete optimization for instance, some of the most powerful relaxations for prominent problems of the field rely on semidefinite programming. For example, the current best approximation algorithm for the maximum-cut problem [54], or the only known polynomial-time algorithm for finding maximum stable sets in perfect graphs [73] make fundamental use of semidefinite programming. In machine learning, one

is often faced with the task of learning a signal with a known structure from data (e.g., a low-rank matrix, a sparse eigenvector, a monotone polynomial, etc.). The design of an appropriate convex optimization problem which induces such structure in a signal often leads to a semidefinite programming problem; see, e.g., [97], [37], [56, Chap. 8]. In polynomial optimization, SDP-based hierarchies based on the notion of sum of squares polynomials (see Section 1.3 below) are among the most promising approaches for obtaining global solutions in the absence of any convexity assumptions [67, 85]. Finally, in control and robotics, various notions of safety, performance, stability, and robustness of dynamical systems can be certified by searching for Lyapunov functions that satisfy appropriate nonnegativity constraints over the state space. When the candidate functions are parameterized as polynomials or other semialgebraic functions, this search can be automated by semidefinite programming via the sum of squares approach; see, again, Section 1.3 below and [84]. Overall, semidefinite programming has proven to be one of the most expressive classes of convex optimization problems, subsuming many other important classes such as linear programming, convex quadratic programming, and second-order cone programming [114].

In addition to its expressiveness, another contributing factor to the popularity of semidefinite programming lies in its strong theoretical and computational properties, which to certain extent parallel that of linear programming [114]. For instance, strong duality between Problem 1 and Problem 2 holds under mild assumptions. Moreover, many algorithms for linear programs, such as the ellipsoid method and interior point methods, can and have been adapted to SDPs. While these algorithms solve SDPs to arbitrary accuracy in polynomial time, in practice, they suffer from one serious impediment: *scalability*. When the problems considered have large n or m (as defined in Problem 1), solving time and memory requirements tend to explode, making SDPs prohibitive to work with. Indeed, most solvers rely on interior point methods as their algorithm of choice for SDPs and interior point methods produce iterates that are (as their name suggests) in the interior of the feasible set. Computing these large dense matrix iterates and storing them lead to the issues described above. Devising methods to make SDPs more scalable is consequently a very important and active area of research.

1.1. The goal and outline of this survey

In this paper, we review recent literature on scalability improvements for semidefinite programming with a particular focus on methods that have proved useful in machine learning, control theory, and robotics. We consider these representative application areas since they are timely and because many problems that arise in them have straightforward semidefinite programming-based relaxations or reformulations, but involve very large problem instances. This paper is a follow-up to a workshop at the 2016 Conference on Decision and Control on the same topic and titled *"Solving Large SDPs in Control, Machine Learning, and Robotics"*. Eleven lectures were given at this workshop whose content can be found at `http://aaa.princeton.edu/largesdps`.

The outline of this paper is as follows. In Section 1.3, we provide a brief introduction to sum of squares proofs of nonnegativity of polynomials. This background material is relevant to many of the applications of semidefinite programming we discuss in this paper (e.g., in Section 2.1.1 and Section 5.1). Section 2 presents approaches that enhance scalability by exploiting structure (e.g., symmetry or sparsity) in a problem. Section 3 presents methods for generating low-rank solutions to SDPs in order to reduce computation time

and storage requirements. Section 4 reviews more scalable alternatives to interior-point methods for solving SDPs based on augmented Lagrangian techniques and the alternating direction method of multipliers. Section 5 presents approaches that trade off scalability with conservatism of solutions (e.g., by approximating SDPs with linear or second-order cone programs). In Section 6, we present a list of software packages that implement many of the techniques described in this paper. Section 7 concludes the paper and highlights promising directions for future work.

1.1.1. Relationship between approaches discussed in the paper. The approaches for improving scalability we discuss in Sections 2–5 are largely complementary to each other. These sections in the paper can thus be read independently of each other. We provide the following rough guide to the different sections (which may be used, e.g., by a practitioner interested in a specific application where scalability is a challenge). The interested reader may also use the guide below for exploring the software packages listed in Section 6, which are also organized according to sections in the paper.

- If the SDP of interest (corresponding to Problem 1) has special *structure* (e.g., symmetry, sparsity, or degeneracy) → Section 2.
- If either (i) the SDP has *low-rank* (optimal) solutions, or (ii) one desires good-quality low-rank feasible points to the SDP → Section 3.
- If *approximately feasible* solutions with good objective values are of interest (i.e., slight violation of the constraints can be tolerated) → Section 4.
- If *convservative feasible* solutions to the SDP (i.e., points that satisfy all constraints, but are potentially suboptimal) are valuable → Section 5.

In principle, the approaches presented in the different sections may be combined with each other to further enhance scalability (e.g., in order to tackle a problem exhibiting sparsity for which slightly suboptimal feasible solutions are valuable). We highlight such combinations of approaches where possible in Sections 2–5.

1.2. Related work not covered by this survey

The number of algorithms that have been proposed as alternatives to interior point methods for semidefinite programming is large and rapidly growing. While we have attempted to present some of the major themes, our survey is certainly not exhaustive. Some of the interesting contributions that this paper does not cover due to space limitations include the recent first-order methods developed by Renegar [98], the multiplicative weights update algorithm of Arora, Hazan, and Kale [15], the storage-optimal algorithm of Ding et al. [41], the iterative projection methods of Henrion and Malick [38], the conditional gradient-based augmented Lagrangian framework of Yurtsever, Fercoq, and Cevher [123], the regularization methods of Nie and Wang for SDP relaxations of polynomial optimization [80], the accelerated first-order method of Bertsimas, Freund, and Sun for the sum of squares relaxation of unconstrained polynomial optimization [20], and the spectral algorithms of Hopkins et al. for certain sum of squares problems arising in machine learning [60].

We also remark that there has been a relatively large recent literature at the intersection of optimization and machine learning where SDP relaxations of certain nonconvex problems are avoided altogether and instead the nonconvex problem is tackled directly with local descent algorithms. These algorithms can often scale to larger problems compared to the

SDP approach (at least when the SDPs are solved by interior point methods). Moreover, under certain technical caveats and statistical assumptions on problem data, it is sometimes possible to prove similar guarantees for these algorithms as those of the SDP relaxation. We do not cover this literature in this paper but point the interested reader to [103] for a list of references.

1.3. A brief review of sum of squares proofs of nonnegativity

Sum of squares constraints on multivariate polynomials are responsible for a substantial fraction of recent applications of semidefinite programming. For the convenience of the reader, we briefly review the basics of this concept and the context in which it arises. This will be relevant e.g. for a better understanding of Section 2.1.1 and Section 5.1.

Recall that a *closed basic semialgebraic set* is a subset of the Euclidean space of the form

$$\Omega := \{x \in \mathbb{R}^n |\ g_i(x) \geq 0, i = 1, \ldots, m\},$$

where $g_1, \ldots, g_m$ are polynomial functions. It is relatively well known by now that many fundamental problems of computational mathematics can be cast as optimization problems where the decision variables are the coefficients of a multivariate polynomial p, the objective function is a linear function of these coefficients, and the constraints are twofold: (i) affine constraints in the coefficients of p, and (ii) the constraint that

$$p(x) \geq 0 \ \forall x \in \Omega, \tag{3}$$

where Ω is a given closed basic semialgebraic set. For example, in a *polynomial optimization problem*—i.e., the problem of minimizing a polynomial function f on Ω—the optimal value is equal to the largest constant γ for which $p(x) := f(x) - \gamma$ satisfies Constraint 3. See [21],[57],[10, Sect. 1.1] for numerous other applications. Unfortunately, imposing Constraint 3 (or even asking a decision version of it for a fixed polynomial p) is NP-hard already when p is a quartic polynomial and $\Omega = \mathbb{R}^n$, or when p is quadratic and Ω is a polytope.

An idea pioneered to a large extent by Lasserre [67] and Parrilo [85] has been to write algebraic sufficient conditions for Constraint 3 based on the concept of sum of squares polynomials. We say that a polynomial h is a *sum of squares* (sos) if it can be written as $h = \sum_i q_i^2$ for some polynomials q_i. Observe that if we succeed in finding sos polynomials $\sigma_0, \sigma_1, \ldots, \sigma_m$ such that the polynomial identity

$$p(x) = \sigma_0(x) + \sum_{i=1}^{m} \sigma_i(x) g_i(x) \tag{4}$$

holds, then, clearly, Constraint 3 must be satisfied. Conversely, a celebrated result in algebraic geometry [94] states that if $g_1, \ldots, g_m$ satisfy the so-called "Archimedean property" (a condition slightly stronger than compactness of the set Ω), then positivity of p on Ω guarantees existence of sos polynomials $\sigma_0, \sigma_1, \ldots, \sigma_m$ that satisfy the algebraic identity in Equation 4.

The computational appeal of the sum of squares approach stems from the fact that the search for sos polynomials $\sigma_0, \sigma_1, \ldots, \sigma_m$ of a given degree that verify the polynomial identity in Equation 4 can be automated via semidefinite programming. This is true even when some coefficients of the polynomial p are left as decision variables. This claim is a

straightforward consequence of the following well-known fact (see, e.g., [84]): A polynomial h of degree $2d$ is a sum of squares if and only if there exists a symmetric matrix Q which is positive semidefinite and verifies the identity

$$h(x) = z(x)^T Q z(x),$$

where $z(x)$ here denotes the vector of all monomials in x of degree less than or equal to d. Note that the size of the semidefinite constraint in an SDP that imposes a sum of squares constraint on an n-variate polynomial of degree $2d$ is $\binom{n+d}{d} \times \binom{n+d}{d} \approx n^d \times n^d$. This is the reason why *sum of squares optimization problems*—i.e., SDPs arising from sum of squares constraints on polynomials—often run into scalability limitations quickly.

2. ENHANCING SCALABILITY BY EXPLOITING STRUCTURE

A major direction for improving the scalability of semidefinite programming has been towards the development of methods for exploiting problem-specific structure. Existing literature in this area has focused on two kinds of structure that appear frequently in applications in control theory, robotics, and machine learning: (i) sparsity, and (ii) symmetry.

2.1. Exploiting sparsity

Many problems in control theory give rise to SDPs that exhibit structure in the form of *chordal sparsity*. We first give a brief introduction to the general notion of chordal sparsity and then highlight applications in control theory that exploit this structure to achieve significant gains in scalability. We begin by introducing a few relevant graph-theoretic notions.

Definition 1 (Cycles and chords)**.** *A* cycle *of length $k \geq 3$ in a graph $\mathcal{G}$ is a sequence of distinct vertices $(v_1, v_2, \ldots, v_k)$ such that (v_k, v_1) is an edge and (v_i, v_{i+1}) is an edge for $i = 1, \ldots, k-1$. A* chord *(associated with a cycle) is an edge that is not part of the cycle but connects two vertices of the cycle.*

Definition 2 (Chordal graph)**.** *An undirected graph $\mathcal{G}$ is* chordal *if every cycle of length $k \geq 4$ has a chord.*

Definition 3 (Maximal clique)**.** *A* clique *of a graph $\mathcal{G}$ is a subset $\mathcal{C}$ of vertices such that for any distinct vertices $i, j \in \mathcal{C}$, (i, j) is an edge in $\mathcal{G}$. A clique is* maximal *if it is not a subset of another clique.*

The graph theoretic notions introduced above can be used to capture sparsity patterns of matrices. Let $\mathcal{E}$ (resp. $\mathcal{V}$) denote the set of edges (resp. vertices) of an undirected graph $\mathcal{G}$. Let S_n denote the set of symmetric $n \times n$ matrices as before, and $S_n(\mathcal{E})$ denote the set of matrices with a sparsity pattern defined by $\mathcal{G}$ as follows:

$$S_n(\mathcal{E}) \coloneqq \{X \in S_n \mid X_{ij} = 0 \text{ if } (i, j) \notin \mathcal{E} \text{ for } i \neq j\}. \tag{5}$$

The following proposition is the primary result that allows one to exploit chordal sparsity (i.e., sparsity induced by a chordal graph) in order to improve scalability of a semidefinite program.

Proposition 1 (see [6]). *Let $\mathcal{G}$ be a chordal graph with vertices $\mathcal{V}$, edges $\mathcal{E}$, and a set of maximal cliques $\Gamma = \{\mathcal{C}_1, \ldots, \mathcal{C}_p\}$. Then $X \in S_n(\mathcal{E})$ is positive semidefinite if and only if there exists a set of matrices $X_k \in S(\mathcal{C}_k) := \{X \in S_n \mid X_{ij} = 0 \text{ if } (i,j) \notin \mathcal{C}_k \times \mathcal{C}_k\}$ for $k = 1, \ldots, p$, such that $X_k \succeq 0$ and $X = \sum_{k=1}^{p} X_k$.*

The above statement allows one to equivalently express a large semidefinite constraint as a set of smaller semidefinite constraints (and additional equality constraints). This can lead to a significant increase in computational efficiency since the computational cost of solving an SDP is primarily determined by the size of the largest semidefinite constraint. We note that for a chordal graph, the problem of listing maximal cliques is amenable to efficient (i.e., polynomial time) algorithms (see [127] and references therein).

Proposition 1 (and a related dual result due to Grone et al. [55]) has been leveraged by several researchers in optimization to improve the efficiency of SDPs with chordal sparsity [50, 63, 49]. These results have also been exploited by researchers in the context of control theory. In [77], the authors apply these results to the problem of finding Lyapunov functions for sparse linear dynamical systems. In particular, by parameterizing Lyapunov functions with a chordal structure (i.e., a structure that allows one to apply Proposition 1), the resulting SDP finds a Lyapunov function significantly faster (up to a factor of ~ 80 faster in terms of running time for instances where the largest maximal clique size is less than 15).

In [126, 127], the authors extend Proposition 1 to the case of matrices with block-chordal sparsity by demonstrating that such matrices can be decomposed into an equivalent set of smaller block-positive semidefinite matrices (again, with additional equality constraints). This result allows for applications to networked systems since it reasons about the sparsity of interconnections between subsystems. The authors apply these results to the problem of designing structured feedback controllers to stabilize large-scale networked systems.

In the work highlighted above, Proposition 1 is used to decompose a large semidefinite constraint into smaller ones with additional equality constraints. The resulting SDP can then be solved using standard techniques (e.g., standard interior point methods). One challenge with this approach (as noted in [128]) is that the additional equality constraints can sometimes nullify the computational benefits of dealing with smaller semidefinite constraints. A set of strategies that seek to overcome this issue involve exploiting chordal sparsity *directly in the solver*, e.g., directly in an interior point method [50, 29, 12] or in a first-order method [107, 62, 74, 106, 128]. In the context of control theory, [13] leverages sparse solvers [12, 50, 19] based on this idea to tackle robust stability analysis for large-scale sparsely interconnected systems.

2.1.1. Exploiting sparsity in polynomial optimization problems. The existence of structure in the form of sparsity can also be exploited for polynomial optimization problems (cf. Section 1.3). In [118], the authors propose the Sparse-BSOS hierarchy (which is based on the Bounded-SOS (BSOS) hierarchy for polynomial optimization problems presented in [68]). The Sparse-BSOS hierarchy leverages the observation that for many polynomial optimization problems that arise in practice, the polynomials that define the constraints of the problem exhibit sparsity in their coefficients. This observation is used to split the variables in the problem into smaller blocks of variables such that (i) each monomial of the polynomial objective function only consists of variables from one of the blocks, and (ii) each polynomial that defines the constraints also only depends on variables in only one of the blocks. Under the assumption that the sparsity in the objective and constraints satisfy the

Running Intersection Property (RIP) [118], the Sparse-BSOS approach provides a hierarchy of SDPs whose optimal values converge to the global optimum of the original polynomial optimization problem. In addition, for the so-called class of "SOS-convex polynomial optimization problems" (see [118] for a definition) that satisfy RIP, the hierarchy converges at the very first step. In contrast to the (standard) BSOS hierarchy (which also provides these guarantees), Sparse-BSOS results in semidefinite constraints of smaller block size. Numerical experiments in [118] demonstrate that exploiting sparsity in the problem can result in the ability to solve large-scale polynomial optimization problems that are beyond the reach of the standard (i.e., non-sparse) BSOS approach.

In [76], the authors utilize the Sparse-BSOS hierarchy to tackle the problem of *simultaneous localization and mapping (SLAM)* in robotics. SLAM [109] refers to the problem of simultaneously estimating the location (or trajectory of locations) of a robot and a model of the robot's environment. Two important versions of the SLAM problem are (i) Landmark SLAM, and (ii) Pose-graph SLAM. The Landmark SLAM problem involves simultaneously estimating the position and orientation of the robot and the location of landmarks in the environment. The Pose-graph SLAM problem is a more restricted version that only requires estimating the position and orientation of the robot at each time-step. Traditionally, these problems have been posed as maximum likelihood estimation (MLE) problems and tackled using general nonlinear optimization methods (which do not come with guarantees on finding globally optimal solutions). In [76], the authors formulate the planar pose-graph and landmark SLAM problems as polynomial optimization problems and demonstrate that they are SOS-convex. This allows [76] to apply the Sparse-BSOS hierarchy with the guarantee that the hierarchy converges to the global optimum at the very first step. The approach in [76] contrasts with most prior works on pose-graph/landmark SLAM, which apply general-purpose nonlinear optimization algorithms to these problems. The numerical results in [76] compare the Sparse-BSOS approach to SLAM with nonlinear optimization techniques based on the Levenberg-Marquardt algorithm and demonstrate that the latter often yield suboptimal solutions while the former finds the globally optimal solution. We also note that the approach in [76] contrasts with the SE-Sync approach [100] we mention in Section 3.1 since [76] relaxes the requirement of limited measurement noise imposed by [100].

2.2. Exploiting symmetry

Beyond sparsity, another general form of structure that arises in applications of semidefinite programming to control theory, robotics, and machine learning is *symmetry*. Mathematically, symmetry refers to a transformation of the variables in a problem that does not change the underlying problem. As a concrete example, consider a multi-agent robotic system composed of a large number of identical agents. The agents in the system can be interchanged while leaving the overall capabilities of the multi-agent system invariant.

Symmetry reduction [52, 113, 39, 90] is a powerful set of approaches for exploiting symmetries in SDPs. We provide a brief overview of symmetry reduction here and refer the reader to [90] for a thorough exposition. Let S_n^+ denote the cone of $n \times n$ symmetric positive semidefinite matrices. Given an SDP of the form of Problem 1, symmetry reduction performs the following two steps:

1. Find an appropriate (see [90]) affine subspace $\mathcal{S} \subset S_n$ containing feasible solutions to the problem;

2. Express $\mathcal{S} \cap S_n^+$ as a linear transformation of a "simpler" cone $\mathcal{C} \subset S_m$, where $m < n$:

$$\mathcal{S} \cap S_n^+ = \{\Phi Z \mid Z \in \mathcal{C}\}. \tag{6}$$

Symmetry reduction techniques work by taking $\mathcal{S}$ equal to the *fixed point subspace* of the group action corresponding to the underlying symmetry group (see [90, 52] for details).

Once steps 1 and 2 above have been accomplished, one can solve the following problem instead of the original SDP:

$$\begin{array}{ll} \min\limits_{Z \in S_m} & \mathrm{Tr}(C\Phi Z) \\ \text{s.t.} & \mathrm{Tr}(A_i \Phi Z) = b_i, \ i = 1, \ldots, m \\ & Z \in \mathcal{C}. \end{array} \tag{7}$$

The key advantage here is that Problem 7 is formulated over a lower-dimensional space S_m, and can thus be more efficient to solve.

Symmetry reduction techniques have been fruitfully applied in many different applications of semidefinite programming to control theory [59, 36, 14]. As an example, [36] exploits symmetry in linear dynamical systems; more precisely, it exploits invariances of the state space realizations of linear systems under unitary coordinate transformations to obtain significant computational gains for control synthesis problems. In [14, Chapter 7], symmetry reduction techniques are used to reduce the computational burden of certifying stability of interconnected subsystems. Specifically, [14] exploits permutation symmetries in a network of dynamical systems (i.e., invariances to exchanging subsystems) in order to reduce the complexity of the resulting SDPs.

2.3. Facial reduction

In certain cases, there may be additional structure beyond sparsity and symmetry that one can exploit in order to reduce the size of SDPs arising in practice. As an example, consider the search for a Lyapunov function $V : \mathbb{R}^n \to \mathbb{R}$ that proves stability of the equilibrium $x_0 \in \mathbb{R}^n$ of a polynomial dynamical system $\dot{x} = f(x)$. In this case, one would like to search for functions V that satisfy the following "degeneracy" conditions: $V(x_0) = 0, \dot{V}(x_0) = 0$. This imposes structure on the resulting optimization problem by restricting the space of possible functions V. One powerful set of approaches for leveraging this kind of "degeneracy" structure is referred to as *facial reduction* [90, 42, 22, 87, 66]. The general procedure behind facial reduction is identical to that of symmetry reduction (Steps 1 and 2 in Section 2.2). The primary difference is that facial reduction techniques identify the subspace $\mathcal{S}$ in a different way than symmetry reduction techniques. In particular, $\mathcal{S}$ is found by exploiting a certain kind of geometric degeneracy condition (see [90] for details).

Facial reduction techniques have been used to improve scalability of SDPs arising in robotics, control theory, and ML applications [90, 115, 64]. We highlight [90], which presents methods for *automating* the process of performing facial (and symmetry) reduction. As an example, [90] considers the problem of analyzing the stability of a *rimless wheel*, which is a simple hybrid dynamical walking robot model. The facial reduction techniques presented in [90] reduce computation times by roughly a factor of 60 for this problem. Similar gains in terms of scalability are also obtained for many other problems. Another recent facial reduction algorithm which stands out in terms of its simplicity of implementation is "*Sieve-SDP*" [129]. This technique can detect redundancies in the constraints of an SDP (or

sometimes infeasibility) by checking positive definiteness of some sub-blocks of the SDP data matrices. In [129], the authors demonstrate the performance of this preprocessing strategy on several benchmark examples, including many arising from SDP relaxations of polynomial optimization problems.

3. ENHANCING SCALABILITY BY PRODUCING LOW-RANK SOLUTIONS

In this section, we focus on methods for producing low-rank feasible solutions to SDP 1 at relatively large scales. One may think of the techniques we present here as leveraging a very specific kind of "structure" (i.e., low-rank structure), similar to the approaches considered in Section 2. However, the technical approaches for leveraging low-rank structure are quite different in nature from the approaches presented in Section 2. Moreover, the literature on low-rank methods is quite vast. We thus treat these methods distinctly from the ones in Section 2. At a high level, one utilizes such methods in two different settings, which we describe now.

Case 1: There is a low-rank optimal solution to Problem 1. In this setting, we can establish a priori that among the optimal solutions to Problem 1, there must be one of low rank. An important case where this is known to hold is when Problem 1 does not have too many constraints. More specifically (see [17, 86, 69]), if Problem 1 has m constraints and an optimal solution, then it also has a rank-r optimal solution with

$$\frac{r(r+1)}{2} \leq m. \tag{8}$$

Note that we need $m < \frac{n(n+1)}{2}$ for this property to be useful.

In this setting, one may not particularly care about obtaining a low-rank (optimal) solution to the problem. Rather, searching for a low-rank solution (whose existence is guaranteed) can lead to significant gains in computation time and storage. To see why, recall that a $n \times n$ positive semidefinite matrix X has rank $r \leq n$ if and only if it can be written as $X = UU^T$, where $U \in \mathbb{R}^{n \times r}$. By searching for a low-rank solution, one can hope to work in the lower-dimensional space $\mathbb{R}^{n \times r}$ corresponding to U, rather than the higher-dimensional space S_n corresponding to X. Doing so would lead to much cheaper algorithms as storage space needs would drop from $O(n^2)$ to $O(nr)$ and, e.g., the important operation of matrix-vector multiplication would require $O(nr)$ flops rather than $O(n^2)$.

Case 2: Good-quality low-rank feasible solutions to Problem 1 are of interest. In this setting, one considers SDPs whose optimal solutions may all have high rank. However, the goal is to find low-rank feasible solutions with good objective values. By making this compromise, one can again expect to work in the lower-dimensional space of low-rank feasible solutions and make gains in terms of computation time and storage as explained previously.

Problems such as these occur in a variety of applications, most notably machine learning and combinatorial optimization. Consider for instance the problem of low-rank matrix completion [32] from machine learning, which is a cornerstone problem in user recommendation systems (see, e.g., the Netflix competition [18]). In this problem, we have access to a matrix $Z \in \mathbb{R}^{m \times n}$, which is partially complete, i.e., some of its entries are specified but others are unknown. The goal is to fill in the unknown entries. Without assuming some structure on

Z, any completion of Z would be a valid guess and so the problem is ill-defined. This can be remedied by assuming that Z should have low rank, which is a natural assumption in the context of user recommender systems. Indeed, if one assumes that the matrix Z reflects the ratings a user i ($i = 1, \ldots, n$) gives a product j ($j = 1, \ldots, m$), then completing the matrix Z would involve infering how a user would have rated a product (s)he has not yet rated from existing ratings, provided by both him/herself and other users. Under the assumption that a consumer base can be segmented into r categories (with $r \ll n$), it would then make sense to constrain the matrix Z to have rank less than or equal r, which would reflect this segmentation. The matrix completion problem would then be written as

$$\begin{aligned} \min_{X \in \mathbb{R}^{m \times n}} \quad & \sum_{i,j} (X_{ij} - Z_{ij})^2 \\ \text{s.t.} \quad & \operatorname{rank}(X) \leq r, \end{aligned} \tag{9}$$

where Z is the partially complete matrix corresponding to past data. The indices i, j in the objective function range over known entries of the matrix Z. Problem (9) can be relaxed to an SDP by replacing the constraint on the rank of X by a constraint on its *nuclear norm* [97], which is a semidefinite-representable surrogate of the rank. The problem then becomes:

$$\begin{aligned} & \min_{X \in \mathbb{R}^{m \times n}, W_1 \in S_m, W_2 \in S_n} \sum_{i,j} (X_{ij} - Z_{ij})^2 \\ & \text{s.t. } \operatorname{Tr} \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix} \leq r, \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix} \succeq 0. \end{aligned} \tag{10}$$

The goal here is to obtain a feasible solution to the problem such that the matrix

$$\begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix}$$

has low rank and the objective value is small. This makes matrix completion a typical application of the algorithms we present next. Other machine learning problems which involve relaxing a rank-constrained SDP to an SDP via the use of the nuclear norm include certain formulations of clustering and maximum variance unfolding [65], and sparse principal component analysis [37].

The rest of this section is devoted to two of the most well-known approaches for computing low-rank feasible (and possibly optimal) solutions to SDP 1. The first is the Burer-Monteiro method and its variants (Section 3.1). The second includes methods that use the Frank-Wolfe algorithm as their computational backbone (Section 3.2).

3.1. Burer-Monteiro based methods

The key idea of the Burer-Monteiro algorithm [30] is to factor the decision variable X in Problem 1 as VV^T where $V \in \mathbb{R}^{n \times r}$. Here, r can either be chosen such that a solution of rank r is guaranteed to exist (e.g., such that Inequality 8 holds) (Case 1 in the previous segment) or such that it corresponds to the rank which we would like our feasible solution to have (Case 2). Typically, the algorithm comes with some theoretical guarantees for the first case and is simply a heuristic in the second. Using this factorization, we rewrite Problem 1 as:

$$\begin{aligned} \min_{V \in \mathbb{R}^{n \times r}} \quad & \operatorname{Tr}(CVV^T) \\ \text{s.t.} \quad & \operatorname{Tr}(A_i VV^T) = b_i, i = 1, \ldots, m. \end{aligned} \tag{11}$$

This reformulation has both pros and cons: on the upside, the problem to consider has smaller dimension than Problem 1 and has no conic constraint to contend with; on the downside, the problem has become nonconvex. Hence, local optimization methods may not always recover the global minimum of Problem 1 even if there is one of low rank. Research in this area has consequently sought to find algorithms to solve Problem 11 that (i) provably recover global low-rank solutions to the problem if they exist; (ii) do so reasonably fast in terms of both theoretical and empirical convergence rates; and (iii) work on a large class of inputs A_i, b_i, and C. We review existing algorithms through the lens of these criteria.

3.1.1. Augmented Lagrangian approaches. The original method recommended by Burer and Monteiro [30] to solve Problem 11 is a method based on an augmented Lagrangian procedure (see Section 4.2 for another appearance of this general approach). The idea is to rewrite Problem 11 as an unconstrained optimization problem by adding terms to the objective that penalize infeasible points. This is done by constructing an augmented Lagrangian function, given here by

$$L_{y,\sigma}(V) = \mathrm{Tr}(CVV^T) - \sum_{i=1}^{m} y_i(\mathrm{Tr}(A_iVV^T) - b_i) + \frac{\sigma}{2}\sum_{i=1}^{m}(\mathrm{Tr}(A_iVV^T) - b_i)^2.$$

If y and $\sigma > 0$ are fixed appropriately, then minimizing $L_{y,\sigma}(\cdot)$ will yield an optimal solution to Problem 11. To obtain an appropriate pair (y, σ), the following iterative procedure is followed: fix (y, σ) and minimize $L_{y,\sigma}(\cdot)$ to obtain V, then construct a new (y, σ) from V, and repeat; see [30] for the exact details. Note that this procedure involves the minimization of $L_{y,\sigma}$ with respect to V: choosing an appropriate algorithm for this subproblem is also an important step in the method. The authors in [30] suggest using a limited-memory BFGS (LM-BFGS) algorithm [71] to do so. The LM-BFGS algorithm is a quasi-Newton algorithm for unconstrained optimization problems, where the objective is differentiable. Unlike Newton methods, it does not require computing the Hessian of the objective function, constructing instead an approximate representation of it from past gradients. Given that gradients of $L_{y,\sigma}$ can be computed quite quickly under some conditions (see [30] for more details), this is a reasonably efficient algorithm to use. One can replace the LM-BFGS subroutine with a subgradient approach if Problem 11 has additional constraints that are non-differentiable. Furthermore, the augmented Lagrangian approach has been tailored to directly handle inequality constraints of the type $\mathrm{Tr}(A_iX) \leq b_i$ in Problem 1; see [65]. Though there are no theoretical guarantees regarding convergence to global solutions, the authors of [30] experimentally observe that the algorithm tends to return global minima of SDP 1, doing so in a much smaller amount of time than interior point or bundle methods. The follow-up paper [31] to [30] slightly modifies the augmented Lagrangian function and shows convergence of the algorithm to a global minimum of SDP 1 (given that one exists of rank $\leq r$) if, among other things, each minimization of the modified $L_{y,\sigma}(\cdot)$ gives rise to a local minimum, a condition that is hard to test in practice.

3.1.2. Riemannian optimization approaches. Riemannian optimization concerns itself with the optimization of a smooth objective function over a smooth Riemannian manifold. We remind the reader that a *Riemannian* manifold $\mathcal{M}$ is a manifold that can be linearized at each point $x \in \mathcal{M}$ by a tangent space $T_x\mathcal{M}$, which is equipped with a Riemannian metric. We say that a Riemannian manifold is *smooth* if the Riemannian metric varies smoothly as the point x varies; see e.g. [23, Appendix A] for a short introduction to such

concepts. In the Riemannian setting, one can define notions of both gradients and Hessians for manifolds [23, Appendix A]. Using these notions, one can extend the concepts of first-order and second-order necessary conditions of optimality to optimization over manifolds. These are almost identical to the ones for unconstrained optimization over $\mathbb{R}^n$, except that they involve the analogs of gradient and Hessians for manifolds. Similarly, one can generalize many classical algorithms for nonlinear unconstrained optimization (such as, e.g., trust-region and gradient-descent methods) to the manifold setting. We refer the reader to [4, 5] for more information on the specifics of these algorithms which we will not cover here.

Though it may not be immediately apparent why, Riemannian optimization methods have become a popular tool for tackling problems of the form 11 [61, 25]. Indeed, the set

$$\mathcal{M} = \{V \in \mathbb{R}^{n\times r} \mid \mathrm{Tr}(A_i V V^T) = b_i, i = 1, \ldots, m\}$$

is a smooth Riemannian manifold under certain conditions on A_i (see [27, Assumption 1]), and the objective function $V \mapsto \mathrm{Tr}(CVV^T)$ is smooth. Hence, Problem 11 is exactly a Riemannian optimization problem. It can be shown that if $m < \frac{r(r+1)}{2}$, the conditions on $\mathcal{M}$ that make it smooth hold, and $\mathcal{M}$ is compact, then for almost all $C \in S_n$, any second-order critical point of Problem 11 is globally optimal to Problem 1 [25, Theorem 2]. (Note that an optimal solution of rank $\leq k$ is guaranteed to exist here from the assumptions and that we use the manifold definition of a second-order critical point.) If an algorithm which returns second-order critical points can be exhibited, then, in effect, we will have found a way of solving Problem 11. It turns out that Riemannian trust-region methods return such a point—under some conditions—regardless of initialization [26]. In fact, it is shown in [26] that an ϵ-accurate second-order critical point (i.e., a point $V \in \mathcal{M}$ with $||\mathrm{grad}(\mathrm{Tr}(CVV^T))|| \leq \epsilon$ and $\mathrm{Hessian}(\mathrm{Tr}(CVV^T)) \succeq -\epsilon I$) can be obtained in $O(1/\epsilon^2)$ iterations. Other Riemannian methods which come with some theoretical guarantees are the Riemannian steepest-descent algorithm and the Riemannian conjugate gradient algorithm, both of them being known to converge to a critical point of Problem 11 [26, 101]. In practice however, many more algorithms for nonlinear optimization can be and have been generalized to manifold settings (though their convergence properties have not necessarily been studied) and are used to solve problems such as Problem 11 (see, e.g., the list of solvers implemented in Manopt [24], one of the main toolboxes for optimization over manifolds).

We conclude this section by mentioning a few applications where Riemannian optimization methods have recently been used. These include machine learning problems such as synchronization [78], phase recovery [105], dictionary learning [104], and low-rank matrix completion [23]; but also robotics problems such as the simultaneous localization and mapping (SLAM) problem (cf. Section 2.1.1). In [34, 35, 100], the authors propose semidefinite programming relaxations for the maximum likelihood estimation (MLE) problem arising from SLAM. These relaxations provide exact solutions to the MLE problem assuming that noise in the measurements made by the robot are below a certain threshold (see [100] for exact conditions). In [100], the authors tackle the challenge of scalability by demonstrating that the resulting SDP admits low-rank solutions and using a Burer-Monteiro factorization combined with Riemannian optimization techniques (e.g., a Riemannian trust-region approach) to solve the SDP. Thus, assuming that the conditions on the measurement noise are satisfied, one can find globally optimal solutions to the MLE problems arising from Pose-graph SLAM (and other related MLE problems arising from camera motion estimation and sensor network localization problems) via this method.

3.1.3. Coordinate descent approaches. One of the more recent trends for solving problems of the form of Problem 11 has been to use coordinate descent (or block-coordinate descent) methods [45, 116]. They have been proposed to solve diagonally-constrained problems, i.e., problems of the type

$$\begin{aligned} \min_{V \in \mathbb{R}^{n \times r}} \quad & \mathrm{Tr}(CVV^T) \\ \text{s.t.} \quad & ||v_i^T|| = b_i, i = 1, \ldots, n. \end{aligned} \tag{12}$$

where $||\cdot||$ is the 2-norm and v_i^T is the i^{th} row of V. This subset of SDPs cover machine learning applications such as certain formulations of graphical model inference and community detection [45]. Coordinate-descent methods are easy to explain for this case. Let us assume that $b_i = 1$ for $i = 1, \ldots, m$ for clarity of exposition. Initialization consists in randomly choosing $v_i, i = 1, \ldots, n$ on the sphere. Then, at each iteration, an index i is picked and fixed and the goal is to find a feasible vector v_i such that $\mathrm{Tr}(CVV^T)$ is minimized. We update the objective by replacing the previous v_i by this new v_i and proceed to the next iteration. This algorithm can be slightly modified to consider descent on blocks of coordinates, rather than one single coordinate at each time-step [45]. In that setting, it can be shown that if the indices are picked appropriately, then coordinate block-descent methods converge to an ϵ-accurate critical point of Problem 12 in time $O(\frac{1}{\epsilon})$. Each iteration of coordinate descent methods is much less costly (by an order of n approximately) than its Riemannian trust-region counterpart however. As a consequence, numerical results tend to show that as n increases, using coordinate descent methods can be preferable [45].

3.2. Frank-Wolfe based methods

We start this subsection with a quick presentation of the Frank-Wolfe algorithm [47](also known as the conditional gradient algorithm), a version of which will be used in the approaches we present. This is an algorithm for constrained optimization problems of the type $\min_{x \in S} f(x)$, where S is a compact and convex set and f is a convex and differentiable function. At iteration k, a linear approximation of f (obtained using the first-order Taylor approximation of f around the current iterate x_k) is minimized over S and a minimizer s_k is obtained. The next iterate x_{k+1} is then constructed by taking a convex combination of the previous iterate x_k and the minimizer s_k before repeating the process. Intuitively, the reason why Frank-Wolfe algorithms are popular for obtaining low-rank feasible solutions for SDPs is due to the following fact: if the algorithm is initialized with a rank-1 matrix and if each minimizer is also a rank-1 matrix, then the k^{th} iterate is of rank at most k. This enables us to control the rank of the solution that is given as output. We first present an algorithm developed by Hazan [58] in Section 3.2.1 and follow up with some extensions of Hazan's algorithm in Section 3.2.2.

3.2.1. Hazan's algorithm. Hazan's algorithm is designed to produce low-rank solutions to SDPs of the type

$$\begin{aligned} \min_{X \in S_n} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_i X) = b_i, \ i = 1, \ldots, m \\ & X \succeq 0 \text{ and } \mathrm{Tr}(X) = 1. \end{aligned} \tag{13}$$

The additional constraint $\mathrm{Tr}(X) = 1$ is required by the algorithm, but can be slightly relaxed to the constraint $\mathrm{Tr}(X) \leq t$, where t is fixed [51, Chapter 5]. We consider the

version with the $\mathrm{Tr}(X) = 1$ constraint here. A key subroutine of Hazan's algorithm is solving the following optimization problem:

$$\begin{aligned} \min_{X \in S_n} \quad & f(X) \\ \text{s.t.} \quad & \mathrm{Tr}(X) = 1 \text{ and } X \succeq 0, \end{aligned} \tag{14}$$

where f is a convex function. This is of interest in itself as some problems can be cast in the form 14, an example being given in [58]. Let f^* be the optimal value of this problem. It is for this subroutine that a Frank-Wolfe type iterative algorithm is used. It can be described as follows: initalize X_1 to be a rank-1 matrix with trace one. Then, at iteration k, compute the eigenvector v_k corresponding to the maximum eigenvalue of $\nabla f(X_k)$ written in matrix form. Finally, for a step size α_k, set $X_{k+1} = (1 - \alpha_k)X_k + \alpha_k(-v_k v_k^T)$ and iterate. This algorithm returns a $\frac{1}{k}$-approximate solution to (14) (i.e., a matrix X such that $\mathrm{Tr}(X) \leq 1 + \frac{1}{k}$ and $f(X) \leq f^* + \Omega(\frac{1}{k})$) with rank at most k in k iterations.

In order to see how this subroutine can be used to solve Problem 13, first note that one can reduce Problem 13 to a sequence of SDP feasibility problems via the use of bisection. It is enough as a consequence to explain how one can leverage the Frank-Wolfe type algorithm described above to solve an SDP feasibility problem of the type:

$$\mathrm{Tr}(A_i X) \leq b_i, i = 1, \ldots, m, X \succeq 0, \mathrm{Tr}(X) = 1.$$

This can be done by letting $f(X) = \frac{1}{M} \log \left(\sum_{i=1}^m e^{M \cdot (\mathrm{Tr}(A_i X) - b_i)} \right)$ where $M = k \log(m)$, $\frac{1}{k}$ being the desired accuracy; see [58]. The algorithm is then guaranteed to return a $\frac{1}{k}$-approximate solution of rank at most k in $O(k^2)$ iterations.

It is interesting to note that this latter algorithm is nearly identical to the multiplicative weights update algorithm for SDPs which also produces low-rank solutions [15] and has exactly the same guarantees. However it is derived in a completely different fashion.

3.2.2. Other methods based on Frank-Wolfe. A caveat of Hazan's algorithm is that the rank of the solution returned is linked to its accuracy. One may think that, if the solution is known to be low-rank and the Frank-Wolfe algorithm is initialized with a low-rank matrix, then all iterates of the algorithm produce a low-rank matrix. Unfortunately this is not always the case. In fact, what can be observed in practice is that the Frank-Wolfe algorithm typically returns iterates with increasing rank up to a certain point, where the rank decreases again, with the final solution returned being low-rank; see, e.g., the numerical experiments in [122]. As a consequence, if one requires an accurate solution, one will likely have to deal with high-rank intermediate iterates, with the storage and computational problems that this entails.

In this section, we present two different methods designed to avoid this issue for optimization problems of the type:

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times d}} \quad & f(\mathcal{A}X) \\ \text{s.t.} \quad & ||X||_* \leq 1, \end{aligned} \tag{15}$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is a convex and differentiable function, $||\cdot||_*$ denotes the nuclear norm, and $\mathcal{A} : \mathbb{R}^{n \times d} \to \mathbb{R}^m$ is a linear operator that maps X to $(\mathrm{Tr}(A_1 X), \ldots, \mathrm{Tr}(A_m X))^T$. The relaxation of the low-rank matrix completion problem given in 10 e.g. fits into this format. Similarly to Problem 14, the Frank-Wolfe algorithm can be applied to Problem 15

quite readily. Initialization is done as before by setting X_0 to be some rank-1 matrix. At iteration k, one computes an update direction by finding a pair (u_k, v_k) of singular vectors corresponding to the maximum singular value of $\mathcal{A}^*(\nabla f(\mathcal{A}X_k))$ where $\mathcal{A}^* : \mathbb{R}^m \mapsto \mathbb{R}^{n\times d}$ is the adjoint operator to $\mathcal{A}$ given by $\mathcal{A}^* z = \sum_{i=1}^m z_i A_i$. We then construct a new iterate $X_{k+1} = (1-\alpha)X_k - \alpha u_k v_k^T$ and repeat.

The first method, described in [48], is a modification of the Frank-Wolfe algorithm as described in the previous paragraph. The key idea is that at every iteration, the algorithm chooses between two types of steps. One type is the step described above, which is derived from the singular vectors of the matrix $\mathcal{A}^*(\nabla f(\mathcal{A}X_k))$, (a "regular step" in the paper); the other is what is called an "in-face step". One way to take such a step is to minimize $\mathrm{Tr}((\mathcal{A}^*\nabla f(AX_k))^T X)$ over X in the minimal face of the nuclear norm unit ball to which X_k belongs. The matrix Z_k obtained from this minimization is then used to produce the next iterate: $X_{k+1} = (1-\alpha)X_k + \alpha Z_k$. We refer the reader to [48] to see how this can be done efficiently. The advantage of such a step is that the matrix X_{k+1} constructed has rank no larger than X_k. If the choice between a regular step and an in-face step is appropriately made, it can still be the case that one has a $\frac{1}{k}$-approximate solution after k iterations, though the rank of the k^{th} iterate could be much less than k depending on the number of "in face" steps used.

The second method given in [122] describes an alternative way of dealing with possibly high rank of the intermediate iterates X_k produced by the Frank-Wolfe algorithm applied to (15). To get around this problem, the authors devise a "dual" formulation of the Frank-Wolfe algorithm described above where the primal iterates $X_k \in \mathbb{R}^{n\times d}$ are replaced by "dual" iterates $z_k \in \mathbb{R}^m$, which are of smaller size. There is a need however to keep track of the iterates X_k in some sense as the ultimate goal is to recover a solution to (15) at the end of the algorithm. To do this, the authors suggest a procedure based on *sketching*. The procedure is as follows: two random matrices $\Omega \in \mathbb{R}^{d\times j}$ and $\Psi \in \mathbb{R}^{l\times n}$ are generated such that j and l are of order of the rank r of the solution to Problem 15, and two new matrices Y and W are obtained as $Y = X\Omega$ and $W = \Psi X$. Note that the matrices (Y, W) typically have smaller rank than the iterates X_k that they represent. Any update that is made in the algorithm is then reflected on (Y, W) rather than on the matrix X. At the end of the algorithm, one can reconstruct a rank r solution to the problem from the updated pair (Y, W); see [122] for details. Note that with this way of proceeding, one need never store or utilize high-rank intermediate iterates.

4. SCALABILITY VIA ADMM AND AUGMENTED LAGRANGIAN METHODS

An attractive alternative to using interior-point methods for solving SDPs is to use *first-order* methods. Compared to interior-point methods, first-order methods can scale to significantly larger problem sizes, while trading off the accuracy of the resulting output (more precisely, first-order methods can require more time to achieve similar accuracy to interior-point methods). First-order methods are thus particularly attractive for applications that demand scalability, but do not require extremely accurate feasible solutions (e.g., some machine learning applications highlighted below). Here, we describe two recent approaches that involve first-order methods: (i) an approach based on the Alternating Direction Method of Multipliers (ADMM) [81], and (ii) an approach based on the augmented Lagrangian method [124, 121].

4.1. Solving SDPs using ADMM

In [81], the authors present a first-order method for solving general cone programs, with SDPs as a special case. Here, we first describe the basic idea behind ADMM and then describe how this is applied to SDPs. We refer the reader to [28] for a thorough exposition to ADMM.

Consider a convex optimization problem of the form:

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{s.t.} \quad & x = z. \end{aligned} \tag{16}$$

Here, the convex functions f and g can be nonsmooth and take on infinite values (e.g., to encode additional constraints). ADMM is a method for solving problems of the form 16 (more generally, ADMM can solve problems where x and z have an affine relationship). ADMM operates by iteratively updating solutions x^k, z^k, and a dual variable λ^k (associated with the constraint $x = z$) via the following steps:

1. $x^{k+1} = \underset{x}{\text{argmin}} \left(f(x) + \frac{\rho}{2} \|x - z^k - \lambda^k\|_2^2 \right)$
2. $z^{k+1} = \underset{z}{\text{argmin}} \left(g(z) + \frac{\rho}{2} \|x^{k+1} - z - \lambda^k\|_2^2 \right)$
3. $\lambda^{k+1} = \lambda^k - x^{k+1} + z^{k+1}$,

where $\rho > 0$ is a parameter. The initializations z^0 and λ^0 are usually taken to be 0, but can be arbitrary. Under mild technical conditions (see [28]), the steps above converge. In particular, the cost $f(x^k) + g(z^k)$ converges to the optimal value of Problem 16, the residual $x^k - z^k$ converges to zero, and λ^k converges to an optimal dual variable.

In [81], the general ADMM procedure above is applied to SDPs. A key idea is to apply ADMM to the *homogeneous self-dual embedding* of the primal SDP (Problem 1) and its dual (Problem 2). The homogeneous self-dual embedding converts the primal-dual pair of optimization problems into a single convex *feasibility* problem. This is done by embedding the Karush-Kuhn-Tucker (KKT) optimality conditions associated with the primal and dual SDPs into a single system of equations and semidefinite constraints. If the original primal-dual pair has a feasible solution, it can be recovered from a solution to the embedding. If on the other hand, the original primal-dual pair is not solvable, the embedding allows one to produce a certificate of infeasibility. The homogeneous self-dual embedding is very commonly used in interior-point methods [102, 3]. We refer the reader to [81] for more details on this technique.

To summarize, the approach presented in [81] consists of the following three steps:

1. Form the homogeneous self-dual embedding associated with the primal-dual SDP pair;
2. Express this embedding in the form 16 required by ADMM;
3. Apply the ADMM steps to this problem.

In [81], the authors also present techniques for efficiently implementing Step 3. These include techniques for (i) efficiently performing the projections required to implement the ADMM steps, (ii) scaling/preconditioning the problem data in order to improve convergence, and (iii) choosing stopping criteria. The resulting approach achieves significant speedups as compared to interior-point methods for a number of large-scale conic optimization problems (with some loss in accuracy of the solution). As an example, [81] considers

the problem of *robust principal component analysis* [33] in machine learning. This problem corresponds to recovering the principal components of a data matrix even when a fraction of its entries are arbitrarily corrupted by noise. More specifically, suppose one is given a (large) data matrix M that is known to be decomposable as

$$M = L + S,$$

where L is a low-rank matrix and S is sparse (but nothing further is known, e.g., the locations or number of nonzero entries of S). Interestingly, this problem can be solved using semidefinite programming under relatively mild conditions (see [33]). For this problem, the ADMM approach presented in [81] is able to tackle large-scale instances where standard interior-point solvers (SeDuMi [102] and SDPT3 [111]) run out of memory. Moreover, the approach provides significant (order of magnitude) speedups on smaller instances, while resulting in only a small loss in accuracy (less than 3×10^{-4} reconstruction error of the low-rank matrix L).

4.2. Augmented Lagrangian methods

Next, we discuss two closely-related approaches for solving SDPs using augmented Lagrangian methods: SDPNAL [124], and SDPNAL+ [121]. SDPNAL operates on the dual SDP 2 by defining an *augmented Lagrangian* function $L_\sigma : \mathbb{R}^m \times S_n \rightarrow \mathbb{R}$ as:

$$L_\sigma(y, X) = -b^T y + \frac{1}{2\sigma}(\|\Pi(X - \sigma(C - \sum_{i=1}^{m} y_i A_i))\|^2 + \|X\|^2), \tag{17}$$

where $\sigma > 0$ is a penalty parameter, $\|\cdot\|$ corresponds to the Frobenius norm, and $\Pi(\cdot)$ is the metric projection operator onto the set S_n^+ of $n \times n$ symmetric psd matrices. More precisely, $\Pi(Y)$ is the (unique) optimal solution to the following convex problem:

$$\min_{Z \in S_n^+} \quad \frac{1}{2}\|Z - Y\|. \tag{18}$$

The augmented Lagrangian function is continuously differentiable (since $\|\Pi(\cdot)\|^2$ is continuously differentiable) and may be viewed as the (usual) Lagrangian function associated with a "regularized" version of the SDP 2 (see [99, 28] for a more thorough exposition).

The augmented Lagrangian method iteratively updates solutions to the dual Problem 2 and the primal Problem 1 using the following steps:

1. $y^{k+1} = \underset{y}{\text{argmin}}\ L_{\sigma_k}(y, X^k)$;
2. $X^{k+1} = \Pi(X^k - \sigma_k(C - \sum_{i=1}^{m} y_i A_i))$;
3. $\sigma_{k+1} = \rho\sigma_k$ for $\rho \geq 1$.

In order to implement this method, we thus require the ability to solve the "inner" optimization problems in Steps 1 and 2 above. The objective functions in these inner optimization problems are convex and continuously differentiable, but not twice continuously differentiable (see [124]). The SDPNAL method presented in [124] works by applying a semismooth Newton method combined with the conjugate gradient method to approximately solve the inner problems. Conditions under which the resulting iterates converge to the optimal solution are established in [124]. In [121], the authors present SDPNAL+, which builds on the approach presented in [124]. However, for certain kinds of SDPs (*degenerate* SDPs; see

[121] for a formal definition), SDPNAL can encounter numerical difficulties. In [121], the authors present SDPNAL+, which tackles this issue by directly working with the primal SDP and using a different technique to solve the inner optimization problems that arise from the augmented Lagrangian method (specifically, a "majorized" semismooth Newton method).

In [121], the authors compare the performance of SDPNAL+ with two other first order methods on various numerical examples. In particular, [121] considers SDP relaxations for the machine learning problem of k-means clustering. Specifically, the goal here is to partition a given set of data points into k clusters such that the total sum-of-squared Euclidean distances from each data point to its assigned cluster center is minimized. In [88], the authors present semidefinite relaxations for finding approximate solutions to this NP-hard problem. The numerical experiments presented in [121] demonstrate that SDPNAL+ is able to solve the resulting SDPs with higher accuracy as compared to the other first-order methods while also resulting in faster running times (e.g., order of magnitude gains in running time on large instances of the clustering problem).

5. TRADING OFF SCALABILITY WITH CONSERVATISM

In this section, we discuss approaches that afford gains in scalability, but are potentially *conservative*. We use the word "conservative" to refer to guaranteed feasible solutions that may be suboptimal (in contrast to approaches that may produce points that slightly violate some constraints). In particular, here we discuss approaches that provide the user with a tuning parameter for *trading off* scalability with conservatism. Conceptually, we classify such approaches into two: (i) "special-purpose" approaches that leverage domain knowledge associated with the target application, and (ii) "general-purpose" approaches that apply broadly across application domains.

As an example of an approach that falls under the first category, we highlight recent work on *neural network verification* using semidefinite programming. Over the last few years, a large body of work has explored the susceptibility of neural networks to adversarial inputs (or more generally, uncertainty in the inputs) [108, 83, 125, 79]. For example, in the context of image classification, neural networks can be made to switch their classification labels by adding very small perturbations (imperceptible to humans) to the input image. Motivated by the potentially serious consequences of such fragilities in safety-critical applications (e.g., autonomous cars), there has been a recent explosion of work on verifying the robustness of neural networks to perturbations to the input [110, 119, 70, 93, 120, 117] (see [70] for a more comprehensive literature review). Recently, approaches based on semidefinite programming have been proposed to tackle this challenge [96, 95, 46]. The challenge of scalability for this application is particularly pronounced since the number of decision variables in the resulting SDPs grows with the number of neurons in the network. In [46], the authors propose an approach for verifying neural networks by capturing their input-output relationships using quadratic constraints. This allows one to *certify* that for a given set of inputs (e.g., a set of images obtained by making small perturbations to a training image), the output is contained within a specified set (e.g., outputs that assign the same label). The authors consider neural networks with different types of activation functions (e.g., ReLU, sigmoid, etc.) and propose different sets of quadratic constraints for each one. Importantly, by including or excluding different kinds of quadratic constraints, the approach allows one to *trade off* scalability with conservatism. Moreover, by exploiting the modular structure of neural networks, the

approach scales to neural networks with roughly five thousand neurons. The scalability afforded by the modular approach, however, comes at the cost of conservatism.

Next, we highlight approaches that fall under the second category mentioned above, i.e., "general-purpose" approaches for trading off scalability with conservatism.

5.1. DSOS and SDSOS optimization

In [10], the authors introduce "DSOS and SDSOS optimization" as linear programming and second-order cone programming-based alternatives to sum of squares and semidefinite optimization that allow for a trade off between computation time and solution quality. The following definitions are central to their framework.

Definition 4 (dd and sdd matrices)**.** *A symmetric matrix $A = (a_{ij})$ is* diagonally dominant *(dd) if*

$$a_{ii} \geq \sum_{j \neq i} |a_{ij}|$$

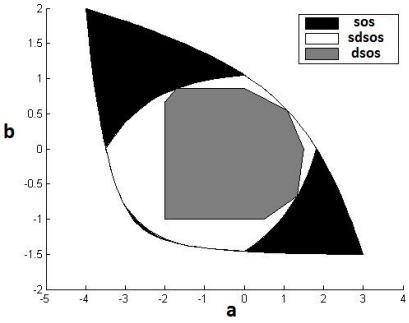
for all i. A symmetric matrix A is scaled diagonally dominant *(sdd) if there exists a diagonal matrix D, with positive diagonal entries, such that DAD is dd.*

Definition 5 (dsos and sdsos polynomials)**.** *A polynomial $p := p(x)$ is said to be* diagonally-dominant-sum-of-squares *(dsos) if it can be written as $p(x) = \sum_i \alpha_i q_i^2(x)$, for some scalars $\alpha_i \geq 0$ and some polynomials $q_i(x)$ that each involve at most two monomials with a coefficient of ± 1. We say that a polynomial p is* scaled-diagonally-dominant-sum-of-squares *(sdsos) if it can be written as $p(x) = \sum_i q_i^2(x)$, for some polynomials q_i that involve at most two monomials with an arbitrary coefficient.*

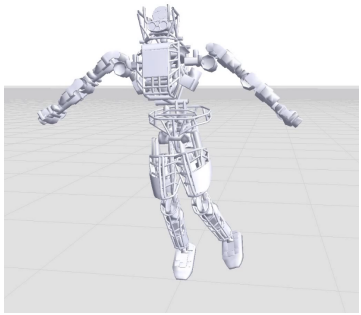
The following implications readily follow: dsos $\Rightarrow$ sdsos $\Rightarrow$ sos. See Figure 1(a) for a comparison of the three notions on a parametric family of bivariate quartic polynomials. Similarly, in view of Gershgorin's circle theorem [53], the implications dd $\Rightarrow$ sdd $\Rightarrow$ psd are straightforward to establish. In [10], the authors connect the above definitions via the following statement.

Theorem 1. *A polynomial p of degree $2d$ is dsos (resp. sdsos) if and only if it admits a representation as $p(x) = z^T(x)Qz(x)$, where $z(x)$ is the standard monomial vector of degree $\leq d$ and Q is a dd (resp. sdd) matrix.*

By combining this theorem with some linear algebraic observations, the authors show in [10] that optimization of a linear objective function over the intersection of the cone of dsos (resp. sdsos) polynomials of a given degree with an affine subspace can be carried out via linear programming (resp. second-order cone programming). These are two mature classes of convex optimization problems that can be solved significantly faster than semidefinite programs. The linear and second-order cone programs that arise from dsos/sdsos constraints on polynomials are termed *"DSOS and SDSOS optimization problems"*. They are used to produce feasible, but possibly suboptimal, solutions to sum of squares optimization problems quickly. In the special case where the polynomials involved have degree two, DSOS and SDSOS optimization problems can be used to produce approximate solutions to semidefinite programs. We also remark that in applications where sum of squares programming is used as a *relaxation*, i.e. to provide an outer approximation of a (typically



(a) The set of parameters a and b for which the polynomial $p(x_1, x_2) = x_1^4 + x_2^4 + ax_1^3x_2 + (1 - \frac{1}{2}a - \frac{1}{2}b)x_1^2x_2^2 + 2bx_1x_2^3$ is dsos/sdsos/sos.



(b) Reference [75] uses SDSOS optimization to balance a model of a humanoid robot with 30 states and 14 control inputs on one foot.

Figure 1: Dsos and sdsos polynomials form structured subsets of sos polynomials which can prove useful when optimization over sos polynomials is too expensive.

intractable) feasible set, this approach replaces the semidefinite constraint with a membership constraint in the *dual* of the cone of dsos or sdsos polynomials. See, e.g., [7, Sect. 3.4] for more details.

Impact on applications. A key practical benefit of the DSOS/SDSOS approach is that it can be used as a "plug-in" in *any* application area of SOS programming. The software package (cf. Section 6) that accompanies reference [10] facilitates this procedure. In [10, Sect. 4], it is shown via numerical experiments from diverse application areas—polynomial optimization, statistics and machine learning, derivative pricing, and control theory—that with reasonable tradeoffs in accuracy, one can achieve noticeable speedups and handle problems at scales that are currently beyond the reach of traditional sum of squares approaches.[1] For the problem of sparse principal component analysis in machine learning for example, the experiments in [10] show that the SDSOS approach is over 1000 times faster than the standard SDP approach on 100×100 input instances while sacrificing only about 2 to 3% in optimality; see [10, Sect. 4.5]. As another example, Figure 1(b) illustrates a humanoid robot with 30 state variables and 14 control inputs that SDSOS optimization is able to stabilize on one foot [75]. A nonlinear control problem of this scale is beyond the reach of standard solvers for sum of squares programs at the moment. In a different paper [9], the authors show the potential of DSOS and SDSOS optimization for real-time applications. More specifically, they use these techniques to compute, every few milliseconds, certificates of collision avoidance for a simple model of an unmanned vehicle that navigates through a cluttered environment.

Some guarantees of the DSOS/SDSOS approach. On the theoretical front, DSOS and SDSOS optimization enjoy some of the same guarantees as those enjoyed by SOS optimization. For example, classical theorems in algebraic geometry can be utilized to conclude that any even positive definite homogeneous polynomial is the ratio of two dsos polynomials [10, Sect. 3.2]. From this observation alone, one can design a hierarchy of

[1] Comparisons are made in [10] with SDP solvers such as MOSEK, SeDuMi, and SDPNAL+.

linear programming relaxations that can solve any copositive program [44] to arbitrary accuracy [10, Sect. 4.2]. This idea can be extended to achieve the same result for any polynomial optimization problem with a compact feasible set; see [8, Sect. 4.2].

5.2. Adaptive improvements to DSOS and SDSOS optimization

While DSOS and SDSOS techniques result in significant gains in terms of solving times and scalability, they inevitably lead to some loss in solution accuracy when compared to the SOS approach. In this subsection, we briefly outline two possible strategies to mitigate this loss. These strategies solve a sequence of adaptive linear programs (LPs) or second-order cone programs (SOCPs) that inner approximate the feasible set of a sum of squares program in a direction of interest. For brevity of exposition, we explain how the strategies can be applied to approximate the generic semidefinite program given in Problem 1. A treatment tailored to the case of sum of squares programs can be found in the references we provide.

5.2.1. Iterative change of basis. In [7], the authors build on the notions of diagonal and scaled diagonal dominance to provide a sequence of improving inner approximations to the cone P_n of psd matrices in the direction of the objective function of an SDP at hand. The idea is simple: define a family of cones[2]

$$DD(U) := \{M \in S_n \mid M = U^T Q U \text{ for some dd matrix } Q\},$$

parametrized by an $n \times n$ matrix U. Optimizing over the set $DD(U)$ is an LP since U is fixed, and the defining constraints are linear in the coefficients of the two unknown matrices M and Q. Furthermore, the matrices in $DD(U)$ are all psd; i.e., $\forall U$, $DD(U) \subseteq P_n$.

The proposal in [7] is to solve a sequence of LPs, indexed by k, by replacing the condition $X \succeq 0$ by $X \in DD(U_k)$:

$$\begin{aligned} DSOS_k := \min_{X \in S_n} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_i X) = b_i, \quad i = 1, \ldots, m, \\ & X \in DD(U_k). \end{aligned} \tag{19}$$

The sequence of matrices $\{U_k\}$ is defined as follows

$$\begin{aligned} U_0 &= I \\ U_{k+1} &= \mathrm{chol}(X_k), \end{aligned} \tag{20}$$

where X_k is an optimal solution to the LP in 19 and chol(.) denotes the Cholesky decomposition of a matrix (this can also be replaced with the matrix square root operation).

Note that the first LP in the sequence optimizes over the set of diagonally dominant matrices. By defining U_{k+1} as a Cholesky factor of X_k, improvement of the optimal value is guaranteed in each iteration. Indeed, as $X_k = U_{k+1}^T I U_{k+1}$, and the identity matrix I is diagonally dominant, we see that $X_k \in DD(U_{k+1})$ and hence is feasible for iteration $k+1$. This implies that the optimal value at iteration $k+1$ is at least as good as the optimal

[2] One can think of $DD(U)$ as the set of matrices that are dd after a change of coordinates via the matrix U.

value at the previous iteration; i.e., $DSOS_{k+1} \leq DSOS_k$ (in fact, the inequality is strict under mild assumptions; see [7]).

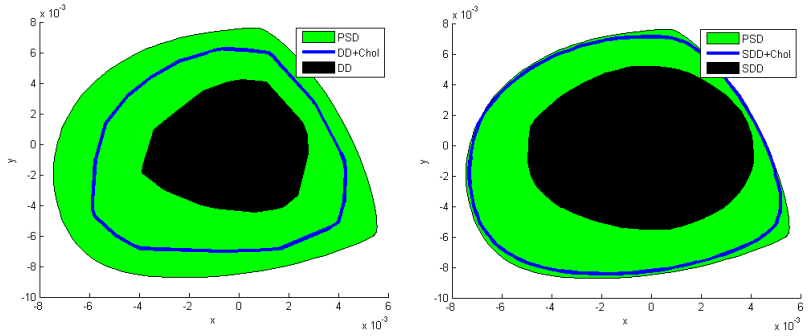
In an analogous fashion, one can construct a sequence of SOCPs that inner approximate P_n with increasing quality. This time, the authors define a family of cones

$$SDD(U) := \{M \in S_n \mid M = U^T Q U, \text{ for some sdd matrix } Q\},$$

parameterized again by an $n \times n$ matrix U. For any U, optimizing over the set $SDD(U)$ is an SOCP and we have $SDD(U) \subseteq P_n$. This leads us to the following iterative SOCP sequence:

$$\begin{aligned} SDSOS_k := \min_{X \in S_n} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_i X) = b_i, \quad i = 1, \ldots, m, \\ & X \in SDD(U_k). \end{aligned} \tag{21}$$

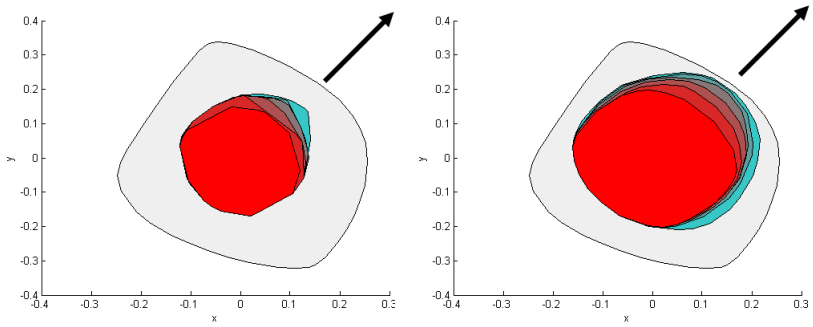
Assuming existence of an optimal solution X_k at each iteration, we can define the sequence $\{U_k\}$ iteratively in the same way as was done in Equation 20. Using similar reasoning, we have $SDSOS_{k+1} \leq SDSOS_k$. In practice, the sequence of upper bounds $\{SDSOS_k\}$ approaches faster to the SDP optimal value than the sequence of the LP upper bounds $\{DSOS_k\}$.



(a) LP inner approximations (b) SOCP inner approximations

Figure 2: Figure reproduced from [7] showing improvement (in all directions) after one iteration of the change of basis algorithm.

Figure 2 shows the improvement (in every direction) obtained just by a single iteration of this approach. The outer set in green in both subfigures is the feasible set of a randomly generated semidefinite program. The sets in black are the diagonally dominant (left) and the scaled diagonally dominant (right) inner approximations. What is shown in dark blue in both cases is the boundary of the improved inner approximation after one iteration. Note that the SOCP in particular fills up almost the entire spectrahedron in a single iteration.



(a) LP iterations (b) SOCP iterations

Figure 3: Figure reproduced from [11] showing the successive improvement on the dd (left) and sdd (right) inner approximation of the feasible set of a random SDP via five iterations of the column generation method.

5.2.2. Column generation. In [11], the authors design another iterative method for inner approximating the set of psd matrices using linear and second order cone programming. Their approach combines DSOS/SDSOS techniques with ideas from the theory of column generation in large-scale linear and integer programming. The high-level idea is to approximate the SDP in Problem 1 by a sequence of LPs (parameterized by t):

$$\begin{aligned} \min_{X \in S_n, \alpha_i} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_i X) = b_i, \quad i = 1, \ldots, m, \\ & X = \sum_{i=1}^{t} \alpha_i B_i, \\ & \alpha_i \geq 0, \quad i = 1, \ldots, t, \end{aligned} \tag{22}$$

where $B_1, \ldots, B_t$ are fixed psd matrices. These matrices are initialized to be the extreme rays of $n \times n$ dd matrices which turn out to be all rank one matrices $v_i v_i^T$, where the vector v_i has at most two nonzero components, each equal to ± 1 [16]. Once this initial LP is solved, one adds one (or sometimes several) new psd matrices B_j to Problem 22 and resolves. This process then continues. In each step, the new matrices B_j are picked carefully to bring the optimal value of the LP closer to that of the SDP. Usually, the construction of B_j involves solving a "*pricing subproblem*" (in the terminology of the column generation literature), which adds appropriate cutting planes to the dual of Problem 22; see [11] for more details.

The SOCP analog of this process is similar. The SDP in Problem 1 is inner approximated

by a sequence of SOCPs (parameterized by t):

$$\begin{aligned} \min_{X \in S_n, \Lambda_i \in S_2} \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_i X) = b_i, \quad i = 1, \ldots, m, \\ & X = \sum_{i=1}^{t} V_i \Lambda_i V_i^T, \\ & \Lambda_i \succeq 0, \quad i = 1, \ldots, t, \end{aligned} \tag{23}$$

where $V_1, \ldots, V_t$ are fixed $n \times 2$ matrices. They are initialized as the set of matrices that have zeros everywhere, except for a 1 in the first column in position j and a 1 in the second column in position $k \neq j$. This gives exactly the set of $n \times n$ sdd matrices [10, Lemma 3.8]. In subsequent steps, one (or sometimes several) appropriately-chosen matrices V_i are added to Problem 23. These matrices are obtained by solving pricing subproblems and help bring the optimal value of the SOCP closer to that of the SDP in each iteration.

Figure 3 shows the improvement obtained by five iterations of this process on a randomly generated SDP, where the objective is to maximize a linear function in the northeast direction.

6. SOFTWARE: SOLVERS AND MODELING LANGUAGES

In this section, we list software tools that allow a user to specify and solve SDPs. This list is not meant to be exhaustive; rather, the goal is to provide the interested practitioner with a starting point in terms of software for implementing some of the approaches reviewed in this paper.

SOFTWARE TOOLS

- Software for specifying SDPs:
  YALMIP [72], CVXPY [40], PICOS [91], SPOT [2], JuMP [43].
  These software packages allow one to easily specify SDPs and provide interfaces to various solvers listed above. YALMIP and SPOT are MATLAB packages, CVXPY and PICOS are Python packages, and JuMP is a Julia package.
- General-purpose SDP solvers that implement interior-point methods:
  MOSEK [3], SeDuMi [102], SDPT3 [111].
  The MOSEK solver provides an interface with MATLAB, C, Python, and Java. SeDuMi and SDPT3 interface with MATLAB.
- Solvers based on ADMM and augmented Lagrangian methods (cf. Section 4):
  SCS [82], SDPNAL/SDPNAL+ [121].
  SDPNAL/SDPNAL+ provide a MATLAB interface, while SCS provides interfaces to C, C++, Python, MATLAB, R, and Julia.
- Software for Riemannian Optimization (cf. Section 3.1):
  Manopt [24], Pymanopt [112].
  Manopt is a MATLAB-based toolbox for optimization on manifolds, while Pymanopt is Python-based.
- Software for exploiting chordal sparsity in SDPs (cf. Section 2.1):
  SparseCoLO [49], CDCS [128].
  SparseCoLO is a MATLAB toolbox for converting SDPs with chordal sparsity to equivalent SDPs with smaller-sized SDP constraints. CDCS is a solver that exploits chordal sparsity and provides a MATLAB interface.
- Software for exploiting structure via facial reduction (cf. Section 2.3):
  frlib [89].
  This is a MATLAB toolbox and interfaces with the YALMIP and SPOT packages mentioned above.
- Parsers for SOS programs (cf. Section 1.3):
  YALMIP [72], SPOT [2], SOSTOOLS [92], SumofSquares.jl [1].
  These packages allow one to specify SOS programs and convert them to a form that can be solved using SDP solvers. YALMIP, SPOT, and SOSTOOLS are MATLAB packages, while SumofSquares.jl is a Julia package.
- Software for parsing DSOS and SDSOS programs (cf. Section 5.1):
  SPOT [2].
  This MATLAB toolbox allows one to parse DSOS and SDSOS programs. SPOT provides interfaces to LP and SOCP solvers including MOSEK. See the appendix of [10] for a tutorial on solving DSOS and SDSOS programs using SPOT.

7. CONCLUSIONS

Semidefinite programming is an exciting and active area of research with a vast number of applications in fields including machine learning, control theory, and robotics. Historically, the lack of scalability of semidefinite programming has been a major impediment to fulfilling the potential that it brings to these application domains. In this paper, we have reviewed recent approaches for addressing this challenge including (i) techniques that exploit structure such as sparsity and symmetry in a problem, (ii) approaches that produce low-rank feasible solutions to SDPs, (iii) methods for solving SDPs via augmented Lagrangian and ADMM techniques, and (iv) approaches that trade off scalability with conservatism, including techniques for approximating SDPs with LPs or SOCPs. We have also provided a list of software packages associated with these approaches. Our hope is that this paper will serve as an entry-point for both practitioners working in application domains that demand scalability, and researchers seeking to contribute to theory and algorithms for scalable semidefinite programming.

Exciting and significant work remains to be done on building upon the advancements reviewed here. Semidefinite programming is still far from being a mature technology similar to linear or quadratic programming. Potential directions for future work include: (i) better theoretical understanding of the convergence properties of algorithms for low-rank SDPs (cf. Section 3), (ii) identifying structure beyond chordal sparsity, symmetry, and degeneracy (cf. Section 2) that arise in practice and methods for exploiting such structure, (iii) exploring different first-order methods for solving SDPs and understanding their relative merits (cf. Section 4), (iv) understanding the power of LPs and SOCPs for approximating SDPs in an adaptive (as opposed to one-shot) fashion (cf. Section 5), (v) finding ways to combine the different approaches for improving scalability reviewed in this paper, and (vi) further developing mature software that allows practitioners to deploy semidefinite programming technology on future applications including ones that involve solving SDPs in real-time.

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

We thank Cemil Dibek for his constructive feedback on the first draft of this manuscript. This work is partially supported by the DARPA Young Faculty Award, the CAREER Award of the NSF, the Google Faculty Award, the Innovation Award of the School of Engineering and Applied Sciences at Princeton University, the Sloan Fellowship, the MURI Award of the AFOSR, the Office of Naval Research [Award Number: N00014-18-1-2873], the NSF CRII award [IIS-1755038], and the Amazon Research Award.

LITERATURE CITED

1. *Sum of Squares Programming for Julia*. URL `https://github.com/JuliaOpt/SumOfSquares.jl`.
2. *Systems Polynomial Optimization Toolbox (SPOT)*, 2013. URL `https://github.com/spot-toolbox/spotless`.

3. *Introducing the MOSEK Optimization Suite*, 2018. URL `https://docs.mosek.com/8.1/intro/index.html`.
4. P.-A. Absil, C. G. Baker, and K. A. Gallivan. Trust-region methods on Riemannian manifolds. *Foundations of Computational Mathematics*, 7(3):303–330, 2007.
5. P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2009.
6. J. Agler, W. Helton, S. McCullough, and L. Rodman. Positive semidefinite matrices with a given sparsity pattern. *Linear Algebra and its Applications*, 107:101–149, 1988.
7. A. A. Ahmadi and G. Hall. Sum of squares basis pursuit with linear and second order cone programming. *Contemporary Mathematics*, pages 27–53, 2017. Available at `https://arxiv.org/pdf/1510.01597.pdf`.
8. A. A. Ahmadi and G. Hall. On the construction of converging hierarchies for polynomial optimization based on certificates of global positivity. *Mathematics of Operations Research*, 2019. Available at `https://arxiv.org/pdf/1709.09307.pdf`.
9. A. A. Ahmadi and A. Majumdar. Some applications of polynomial optimization in operations research and real-time decision making. *Optimization Letters*, 10(4):709–729, 2016.
10. A. A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebraic Geometry*, 3(193), 2019.
11. A. A. Ahmadi, S. Dash, and G. Hall. Optimization over structured subsets of positive semidefinite matrices via column generation. *Discrete Optimization*, 24:129–151, 2017.
12. M. S. Andersen, J. Dahl, and L. Vandenberghe. Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones. *Mathematical Programming Computation*, 2(3-4):167–201, 2010.
13. M. S. Andersen, S. K. Pakazad, A. Hansson, and A. Rantzer. Robust stability analysis of sparsely interconnected uncertain systems. *IEEE Transactions on Automatic Control*, 59(8): 2151–2156, 2014.
14. M. Arcak, C. Meissen, and A. Packard. *Networks of Dissipative Systems: Compositional Certification of Stability, Performance, and Safety*. Springer, 2016.
15. S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 339–348, 2005.
16. G. Barker and D. Carlson. Cones of diagonally dominant matrices. *Pacific Journal of Mathematics*, 57(1):15–32, 1975.
17. A. I. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete & Computational Geometry*, 13(2):189–202, 1995.
18. J. Bennett, S. Lanning, et al. The Netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA., 2007.
19. S. J. Benson, Y. Ye, et al. DSDP5 user guide-software for semidefinite programming. Technical report, Argonne National Lab, Argonne, 2006.
20. D. Bertsimas, R. M. Freund, and X. A. Sun. An accelerated first-order method for solving SOS relaxations of unconstrained polynomial optimization problems. *Optimization Methods and Software*, 28(3):424–441, 2013.
21. G. Blekherman, P. A. Parrilo, and R. Thomas. *Semidefinite Optimization and Convex Algebraic Geometry*. SIAM Series on Optimization, 2013.
22. J. Borwein and H. Wolkowicz. Regularizing the abstract convex program. *Journal of Mathematical Analysis and Applications*, 83(2):495–530, 1981.
23. N. Boumal and P.-a. Absil. RTRMC: A Riemannian trust-region method for low-rank matrix completion. In *Advances in Neural Information Processing Systems*, pages 406–414, 2011.
24. N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a MATLAB toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014. URL

`http://www.manopt.org`.
25. N. Boumal, V. Voroninski, and A. Bandeira. The non-convex Burer-Monteiro approach works on smooth semidefinite programs. In *Advances in Neural Information Processing Systems*, pages 2757–2765, 2016.
26. N. Boumal, P.-A. Absil, and C. Cartis. Global rates of convergence for nonconvex optimization on manifolds. *IMA Journal of Numerical Analysis*, 39(1):1–33, 2018.
27. N. Boumal, V. Voroninski, and A. S. Bandeira. Deterministic guarantees for Burer-Monteiro factorizations of smooth semidefinite programs. *arXiv preprint arXiv:1804.02008*, 2018.
28. S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.
29. S. Burer. Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM Journal on Optimization*, 14(1):139–172, 2003.
30. S. Burer and R. D. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003.
31. S. Burer and R. D. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444, 2005.
32. E. J. Candès and Y. Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6): 925–936, 2010.
33. E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.
34. L. Carlone, D. M. Rosen, G. Calafiore, J. J. Leonard, and F. Dellaert. Lagrangian duality in 3D SLAM: verification techniques and optimal solutions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 125–132, 2015.
35. L. Carlone, G. C. Calafiore, C. Tommolillo, and F. Dellaert. Planar pose graph optimization: duality, optimal solutions, and verification. *IEEE Transactions on Robotics*, 32(3):545–565, 2016.
36. R. Cogill, S. Lall, and P. A. Parrilo. Structured semidefinite programs for the control of symmetric systems. *Automatica*, 44(5):1411–1417, 2008.
37. A. d Aspremont, L. E. Ghaoui, M. I. Jordan, and G. R. Lanckriet. A direct formulation for sparse PCA using semidefinite programming. *SIAM Review*, 49(3):434–448, 2007.
38. D. D. Henrion and J. Malick. Projection methods for conic feasibility problems: applications to polynomial sum-of-squares decompositions. *Optimization Methods & Software*, 26(1):23–46, 2011.
39. 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.
40. S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
41. L. Ding, A. Yurtsever, V. Cevher, J. A. Tropp, and M. Udell. An optimal-storage approach to semidefinite programming using approximate complementarity. *arXiv preprint arXiv:1902.03373*, 2019.
42. D. Drusvyatskiy, H. Wolkowicz, et al. The many faces of degeneracy in conic optimization. *Foundations and Trends® in Optimization*, 3(2):77–170, 2017.
43. I. Dunning, J. Huchette, and M. Lubin. JuMP: a modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.
44. M. Dür. Copositive programming–a survey. In *Recent Advances in Optimization and its Applications in Engineering*, pages 3–20. Springer, 2010.
45. M. A. Erdogdu, A. Ozdaglar, P. A. Parrilo, and N. D. Vanli. Convergence rate of block-coordinate maximization Burer-Monteiro method for solving large SDPs. *arXiv preprint arXiv:1807.04428*, 2018.
46. M. Fazlyab, M. Morari, and G. J. Pappas. Safety verification and robustness analysis of

neural networks via quadratic constraints and semidefinite programming. *arXiv preprint arXiv:1903.01287*, 2019.
47. M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.
48. R. M. Freund, P. Grigas, and R. Mazumder. An extended Frank–Wolfe method with in-face directions, and its application to low-rank matrix completion. *SIAM Journal on Optimization*, 27(1):319–346, 2017.
49. 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. *Research Report B-453, Dept. of Math. and Comp. Sci. Japan, Tech. Rep.*, pages 152–8552, 2009.
50. M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: General framework. *SIAM Journal on Optimization*, 11 (3):647–674, 2001.
51. B. Gärtner and J. Matousek. *Approximation Algorithms and Semidefinite Programming*. Springer Science & Business Media, 2012.
52. K. Gatermann and P. A. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. *Journal of Pure and Applied Algebra*, 192(1-3):95–128, 2004.
53. S. A. Gershgorin. Uber die Abgrenzung der Eigenwerte einer Matrix. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, (6):749–754, 1931.
54. M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6): 1115–1145, 1995.
55. R. Grone, C. R. Johnson, E. M. Sá, and H. Wolkowicz. Positive definite completions of partial Hermitian matrices. *Linear Algebra and its Applications*, 58:109–124, 1984.
56. G. Hall. *Optimization over nonnegative and convex polynomials with and without semidefinite programming*. PhD thesis, Princeton University, 2018.
57. G. Hall. Engineering and business applications of sum of squares polynomials. Available at `https://arxiv.org/pdf/1906.07961.pdf`, 2019.
58. E. Hazan. Sparse approximate solutions to semidefinite programs. In *Latin American Symposium on Theoretical Informatics*, pages 306–316. Springer, 2008.
59. D. Henrion and A. Garulli. *Positive polynomials in control*, volume 312. Springer Science & Business Media, 2005.
60. S. B. Hopkins, T. Schramm, J. Shi, and D. Steurer. Fast spectral algorithms from sum-of-squares proofs: tensor decomposition and planted sparse vectors. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*, pages 178–191, 2016.
61. M. Journée, F. Bach, P.-A. Absil, and R. Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization*, 20(5):2327–2351, 2010.
62. A. Kalbat and J. Lavaei. A fast distributed algorithm for decomposable semidefinite programs. In *Proceedings of the 54th IEEE Conference on Decision and Control*, pages 1742–1749, 2015.
63. S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion. *Mathematical programming*, 129(1):33–68, 2011.
64. N. Krislock and H. Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. *SIAM Journal on Optimization*, 20(5):2679–2708, 2010.
65. B. Kulis, A. C. Surendran, and J. C. Platt. Fast low-rank semidefinite programming for embedding and clustering. In *Artificial Intelligence and Statistics*, pages 235–242, 2007.
66. V. Kungurtsev and J. Marecek. A two-step pre-processing for semidefinite programming. *arXiv preprint arXiv:1806.10868*, 2018.
67. J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.
68. J. B. Lasserre, K.-C. Toh, and S. Yang. A bounded degree SOS hierarchy for polynomial

optimization. *EURO Journal on Computational Optimization*, 5(1-2):87–117, 2017.
69. A. Lemon, A. M.-C. So, Y. Ye, et al. Low-rank semidefinite programming: theory and applications. *Foundations and Trends® in Optimization*, 2(1-2):1–156, 2016.
70. C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.
71. D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.
72. J. Löfberg. Yalmip: a toolbox for modeling and optimization in matlab. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
73. L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, 1979.
74. R. Madani, A. Kalbat, and J. Lavaei. ADMM for sparse semidefinite programming with applications to optimal power flow problem. In *Proceedings of the 54th IEEE Conference on Decision and Control*, pages 5932–5939, 2015.
75. A. Majumdar, A. A. Ahmadi, and R. Tedrake. Control and verification of high-dimensional systems via DSOS and SDSOS optimization. In *Proceedings of the IEEE Conference on Decision and Control*, 2014.
76. J. G. Mangelson, J. Liu, R. M. Eustice, and R. Vasudevan. Guaranteed globally optimal planar pose graph and landmark SLAM via sparse-bounded sums-of-squares programming. *arXiv preprint arXiv:1809.07744*, 2018.
77. R. P. Mason and A. Papachristodoulou. Chordal sparsity, decomposing SDPs and the Lyapunov equation. In *Proceedings of the American Control Conference*, pages 531–537, 2014.
78. S. Mei, T. Misiakiewicz, A. Montanari, and R. I. Oliveira. Solving SDPs for synchronization and maxcut problems via the Grothendieck inequality. *arXiv preprint arXiv:1703.08729*, 2017.
79. S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1765–1773, 2017.
80. J. Nie and L. Wang. Regularization methods for SDP relaxations in large-scale polynomial optimization. *SIAM Journal on Optimization*, 22(2):408–428, 2012.
81. B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169 (3):1042–1068, 2016.
82. B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.1.0. `https://github.com/cvxgrp/scs`, Nov. 2017.
83. N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *Proceedings of the IEEE European Symposium on Security and Privacy*, pages 372–387, 2016.
84. P. A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, May 2000.
85. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2, Ser. B):293–320, 2003.
86. G. Pataki. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of Operations Research*, 23(2):339–358, 1998.
87. G. Pataki. Strong duality in conic linear programming: facial reduction and extended duals. In *Computational and Analytical Mathematics*, pages 613–634. Springer, 2013.
88. J. Peng and Y. Wei. Approximating k-means-type clustering via semidefinite programming. *SIAM Journal on Optimization*, 18(1):186–205, 2007.
89. F. Permenter and P. Parrilo. Partial facial reduction: simplified, equivalent SDPs via approximations of the PSD cone. *Mathematical Programming*, 171(1-2):1–54, 2018.
90. F. N. Permenter. *Reduction methods in semidefinite and conic optimization*. PhD thesis, Massachusetts Institute of Technology, 2017.

91. PICOS. *The PICOS Documentation*, 2018. URL `https://picos-api.gitlab.io/picos/`.
92. S. Prajna, A. Papachristodoulou, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2002-05. Available from `http://www.cds.caltech.edu/sostools` and `http://www.mit.edu/~parrilo/sostools`.
93. L. Pulina and A. Tacchella. Challenging SMT solvers to verify neural networks. *AI Communications*, 25(2):117–135, 2012.
94. M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, 42(3):969–984, 1993.
95. C. Qin, B. O'Donoghue, R. Bunel, R. Stanforth, S. Gowal, J. Uesato, G. Swirszcz, P. Kohli, et al. Verification of non-linear specifications for neural networks. *arXiv preprint arXiv:1902.09592*, 2019.
96. A. Raghunathan, J. Steinhardt, and P. S. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10877–10887, 2018.
97. B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.
98. J. Renegar. Accelerated first-order methods for hyperbolic programming. *Mathematical Programming*, pages 1–35, 2019.
99. R. T. Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1(2):97–116, 1976.
100. D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard. SE-Sync: a certifiably correct algorithm for synchronization over the special Euclidean group. *The International Journal of Robotics Research*, 38(2-3):95–125, 2019.
101. H. Sato and T. Iwai. A new, globally convergent Riemannian conjugate gradient method. *Optimization*, 64(4):1011–1031, 2015.
102. J. Sturm. *SeDuMi version 1.05*, Oct. 2001. Latest version available at `http://sedumi.ie.lehigh.edu/`.
103. J. Sun. Provable nonconvex methods and algorithms (collection of papers). References available at `https://sunju.org/research/nonconvex/`, 2019.
104. J. Sun, Q. Qu, and J. Wright. Complete dictionary recovery over the sphere II: recovery by Riemannian trust-region method. *IEEE Transactions on Information Theory*, 63(2):885–914, 2016.
105. J. Sun, Q. Qu, and J. Wright. A geometric analysis of phase retrieval. *Foundations of Computational Mathematics*, 18(5):1131–1198, 2018.
106. Y. Sun and L. Vandenberghe. Decomposition methods for sparse matrix nearness problems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1691–1717, 2015.
107. Y. Sun, M. S. Andersen, and L. Vandenberghe. Decomposition in conic optimization with partially separable structure. *SIAM Journal on Optimization*, 24(2):873–897, 2014.
108. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.
109. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT press, 2005.
110. V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.
111. K. C. Toh, R. H. Tütüncü, and M. J. Todd. *SDPT3 - a MATLAB software package for semidefinite-quadratic-linear programming*. Available from `http://www.math.cmu.edu/~reha/sdpt3.html`.
112. J. Townsend, N. Koep, and S. Weichwald. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *The Journal of Machine Learning Research*, 17(1): 4755–4759, 2016.
113. F. Vallentin. Symmetry in semidefinite programs. *Linear Algebra and its Applications*, 430 (1):360–369, 2009.

114. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, Mar. 1996.
115. H. Waki, Y. Ebihara, and N. Sebe. Reduction of SDPs in H-infinity control of SISO systems and performance limitations analysis. In *Proceedings of the 55th IEEE Conference on Decision and Control*, pages 646–651, 2016.
116. P.-W. Wang, W.-C. Chang, and J. Z. Kolter. The mixing method: coordinate descent for low-rank semidefinite programming. *arXiv preprint arXiv:1706.00476*, 2017.
117. S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6367–6377, 2018.
118. T. Weisser, J. B. Lasserre, and K.-C. Toh. Sparse-BSOS: a bounded degree SOS hierarchy for large scale polynomial optimization with sparsity. *Mathematical Programming Computation*, 10(1):1–32, 2018.
119. E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.
120. W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989*, 2018.
121. L. Yang, D. Sun, and K.-C. Toh. SDPNAL+: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints. *Mathematical Programming Computation*, 7(3):331–366, 2015.
122. A. Yurtsever, M. Udell, J. A. Tropp, and V. Cevher. Sketchy decisions: Convex low-rank matrix optimization with optimal storage. *arXiv preprint arXiv:1702.06838*, 2017.
123. A. Yurtsever, O. Fercoq, and V. Cevher. A conditional gradient-based augmented Lagrangian framework. *arXiv preprint arXiv:1901.04013*, 2019.
124. X.-Y. Zhao, D. Sun, and K.-C. Toh. A Newton-CG augmented Lagrangian method for semidefinite programming. *SIAM Journal on Optimization*, 20(4):1737–1765, 2010.
125. S. Zheng, Y. Song, T. Leung, and I. Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4480–4488, 2016.
126. Y. Zheng, R. P. Mason, and A. Papachristodoulou. A chordal decomposition approach to scalable design of structured feedback gains over directed graphs. In *Proceedings of the 55th IEEE Conference on Decision and Control*, pages 6909–6914, 2016.
127. Y. Zheng, R. P. Mason, and A. Papachristodoulou. Scalable design of structured controllers using chordal decomposition. *IEEE Transactions on Automatic Control*, 63(3):752–767, 2017.
128. Y. Zheng, G. Fantuzzi, A. Papachristodoulou, P. Goulart, and A. Wynn. Chordal decomposition in operator-splitting methods for sparse semidefinite programs. *Mathematical Programming*, pages 1–44, 2019.
129. Y. Zhu, G. Pataki, and Q. Tran-Dinh. Sieve-SDP: a simple facial reduction algorithm to preprocess semidefinite programs. *Mathematical Programming Computation*, 11(3):503–586, 2019.