A Subgradient Projection Method with Outer Approximation for Solving Semidefinite Programming Problems

Nagisa Sugishita[1] and Miguel F. Anjos[1*]

[1*]School of Mathematics, University of Edinburgh, James Clerk Maxwell Building, Peter Guthrie Tait Road, Edinburgh, EH9 3FD, UK.

*Corresponding author(s). E-mail(s): anjos@stanfordalumni.org;
Contributing authors: nsugishi@ed.ac.uk;

Abstract

We explore the combination of subgradient projection with outer approximation to solve semidefinite programming problems. We compare several ways to construct outer approximations using the problem structure. The resulting approach enjoys the strengths of both subgradient projection and outer approximation methods. Preliminary computational results on the semidefinite programming relaxations of graph partitioning and max-cut show that our approach is competitive for solving large-scale instances.

Keywords: Semidefinite Programming, Subgradient Projection Method, Outer Approximation

1 Introduction

In the last few decades, semidefinite programming (SDP) has become useful in modelling various important applications, ranging from optimal control [6] to combinatorial optimization [13]. A comprehensive list of applications of SDP can be found in the survey paper by Vandenberghe and Boyd [32] and the handbook by Anjos and Lasserre [3].

The main workhorse to solve SDP problems has been the interior point method (IPM) approach [22]. Given a tolerance, an IPM finds a near-optimal solution within a polynomial number of iterations. However, IPMs require the solution of large linear

systems of equations, which makes it challenging to solve a large-scale problem in practice.

To overcome this difficulty, many alternative algorithms have been proposed [19]. First-order methods are typical examples in this direction. The Burer-Monteiro method [8] reformulates an SDP problem by factorising the decision variable (matrix) as the product of low-rank matrices. The reformulated problem is non-convex and is solved heuristically using an augmented Lagrangian method and a quasi-Newton method. Helmberg and Rendl [15] study the use of a bundle method. When feasible solutions have a constant, known trace, the SDP problem can be reformulated as an unconstrained eigenvalue optimization problem, which can be solved using a bundle method. Also, there is an emerging interest in applying augmented Lagrangian methods to SDP, such as the boundary point studied by Povh et al [27]. O'Donoghue et al [25] and Garstka et al [12] use an alternating direction method of multipliers (ADMM) to solve SDP. Under mild conditions, the objective value of the iterates converges to the optimal objective value, which follows from standard results for ADMM [7].

Another family of approaches uses an outer approximation of the feasible set. The positive definiteness constraint is replaced with cheaper constraints that are often referred to as cuts. The cutting-plane method of Krishnan and Mitchell [16] is an example of how to replace the feasible set with a polyhedral cone, namely a system of linear inequalities. Similarly, Ahmadi et al [1] use second-order cones as an outer approximation of the feasible set. In those methods, the outer approximation is tightened successively by adding new cuts. A major drawback of these approaches is the lack of a convergence guarantee unless the number of cuts can grow arbitrarily. Analytic centre cutting-plane methods (ACCPM) are variants of cutting-plane methods that compute new cuts based on the analytic centre of the current outer approximation. ACCPM was first developed for linear programming by [14], and extended by Oskoorouchi and Mitchell [24] to solve SDP. Unlike other cutting-plane methods, ACCPM only requires a polynomial number of cuts. However, before generating each cutting-plane, the analytic centre of the current outer approximation must be computed. This can be expensive, particularly when the number of constraints is large, as discussed in [24].

In this paper, we study the combination of subgradient projection methods and outer approximation. Subgradient methods have played a crucial role in convex optimization [4]. These methods can handle a simple constraint by means of projection. By simple we mean that it is easy to project a point onto the feasible set. More precisely, after each iteration, the subgradient step is projected onto the feasible set and the projection is used as the next iterate.

However, when it is difficult to find the projection onto the feasible set, this approach becomes impractical. Recently, there has been some work to overcome this difficulty. For instance, Nedic [20], Neto and de Pierro [23] and Nesterov [21] study the use of subgradient projection methods [26] together with subgradient methods. In their methods, the projection in subgradient methods is replaced with a cheaper operation, the subgradient projection.

We propose to combine subgradient projection methods with outer approximation of the feasible set. We examine effective ways to construct outer approximations of the

feasible set of a SDP problem to significantly improve the performance of subgradient projection methods. The performance of the proposed approach is shown on SDP relaxations of SDPLIB instances and of large-scale instances of the graph partitioning problem and the max-cut problem.

The remainder of the paper is structured as follows. In Section 2.1 we introduce our notation and subgradient projection methods. In Section 2.2 we present our idea to combine subgradient projection methods with outer approximation. In Section 2.3 we state the convergence analysis of the algorithm using diminishing step sizes. In Section 2.4, we consider the application of the algorithm to SDP and examine approaches for constructing outer approximations of the feasible region. In Section 2.5, we present our proposed algorithms for constructing outer approximations. In Section 3 we describe the setup and report the results of our computational experiments to evaluate the performance of our method. Section 4 concludes the paper.

2 Existing and Proposed Approaches

2.1 Subgradient Projection Methods

Consider the general convex optimization problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \ & f(x) \\ \text{s.t. } & g(x) \leq 0, \end{aligned} \tag{1}$$

where f and g are real-valued, convex functions. We assume that problem (1) has an optimal solution.

Subgradient methods are iterative methods that, at each iteration, take a subgradient step to improve the objective value followed by a projection step to restore feasibility [4]. Let

$$F := \{x \in \mathbb{R}^n : g(x) \leq 0\}$$

be the feasible set of problem (1). Subgradient methods are defined as:

$$\begin{aligned} y^{(k)} &= x^{(k)} - \alpha^{(k)} t^{(k)}, \\ x^{(k+1)} &= \Pi_F(y^{(k)}), \qquad \text{for } k \geq 0, \end{aligned} \tag{2}$$

where $t^{(k)} \in \partial f(x^{(k)})$, $\Pi_F(y^{(k)})$ is the projection of $y^{(k)}$ onto F and $\alpha^{(k)}$ is the step size. Although algorithm (2) is conceptually simple, finding $\Pi_F(y^{(k)})$ can be computationally expensive. For instance, in the application we consider later, F is a spectrahedron and computing the projection onto F is as hard as solving (1).

In such a case, it is natural to replace the projection in algorithm (2) with the subgradient projection [26]. Let $[a]^+ = \max\{a, 0\}$ for $a \in \mathbb{R}$. The method of Nedic [20], Neto and de Pierro [23] is

$$y^{(k)} = x^{(k)} - \alpha^{(k)} t^{(k)},$$

$$x^{(k+1)} = y^{(k)} - \frac{[g(y^{(k)})]^+}{\|d^{(k)}\|^2} d^{(k)}, \qquad \text{for } k \geq 0, \tag{3}$$

with $d^{(k)}$ chosen as $d^{(k)} \in \partial g(y^{(k)})$ if $g(y^{(k)}) > 0$ and $d^{(k)} = d$ for some $d \neq 0$ if $g(y^{(k)}) \leq 0$. Nesterov [21] studies a method which takes only the subgradient projection step when the constraint violation $g(y^{(k)})$ is large, and the subgradient step otherwise.

2.2 Subgradient Projection Methods with Outer Approximations

Algorithm (3) restores feasibility solely by the subgradient projection. If we have outer approximations $P^{(k)}$ and $Q^{(k)}$ of F for $k = 0, 1, \ldots$, we can use them to reduce infeasibility:

$$\begin{aligned} y^{(k)} &= \Pi_{P^{(k)}}(x^{(k)} - \alpha^{(k)} t^{(k)}), \\ x^{(k+1)} &= \Pi_{Q^{(k)}} \left(y^{(k)} - \frac{[g(y^{(k)})]^+}{\|d^{(k)}\|^2} d^{(k)} \right), \quad \text{for } k \geq 0. \end{aligned} \tag{4}$$

Here, $P^{(k)}$ and $Q^{(k)}$ can be constructed using intermediate results obtained during the execution of algorithm (3). For example, let $\bar{y} \in \mathbb{R}^n$, $\bar{s} \in \partial g(\bar{y})$ and $x \in F$. Since g is convex, we have $g(\bar{y}) + \bar{s}^T(x - \bar{y}) \leq g(x)$. Furthermore, since x is feasible, $g(x) \leq 0$. Combining these two relations, we obtain

$$g(\bar{y}) + \bar{s}^T(x - \bar{y}) \leq 0 \qquad \text{for } x \in F. \tag{5}$$

Thus, for $k = 0, 1, \ldots$, we can obtain valid outer approximations of F as

$$\begin{aligned} Q^{(k)}_{\text{subgrad}} &:= \{x \in \mathbb{R}^n : g(y^{(i)}) + (s^{(i)})^T(x - y^{(i)}) \leq 0,\ s^{(i)} \in \partial g(y^{(i)}),\ 0 \leq i \leq k\}, \\ P^{(k)}_{\text{subgrad}} &:= Q^{(k-1)}_{\text{subgrad}}, \end{aligned} \tag{6}$$

where we define $Q^{(-1)}_{\text{subgrad}}$ as $\mathbb{R}^n$.

2.3 Convergence Analysis

Algorithm (4) maintains the overall structure of algorithm (3). The difference is the additional projections onto outer approximations $P^{(k)}$ and $Q^{(k)}$. Intuitively, since the projection onto an outer approximation does not move the iterate further from the feasible region, algorithm (4) inherits many convergence properties from (3). For example, the convergence of the iterate with a diminishing step size can be shown using a similar argument to that used by Nedic [20], under the following assumptions.

Assumption 1. *Let the following hold:*

1. *f and g are real-valued, convex functions.*

2. The subgradients of f and g are uniformly bounded:

$$\begin{aligned} \|t\| &\le C_f, \qquad \text{for all } t \in \partial f(x), x \in \mathbb{R}^n, \\ \|s\| &\le C_g, \qquad \text{for all } s \in \partial g(x), x \in \mathbb{R}^n. \end{aligned}$$

3. There exists a constant $c > 0$ such that

$$\operatorname{dist}(x, F) \le c[g(x)]^+ \qquad \text{for all } x \in \mathbb{R}^n. \tag{7}$$

4. (1) has at least one optimal solution.

Condition (7) is known as the global error bound [18]. Under these assumptions, we have the following result with diminishing step sizes.

Proposition 2. *Suppose that Assumption 1 holds. If the step size is positive and satisfies*

$$\sum_{k=0}^{\infty} \alpha^{(k)} = \infty, \sum_{k=0}^{\infty} (\alpha^{(k)})^2 < \infty,$$

then $\{x^{(k)}\}$ converges to some optimal solution.

The proof of Proposition 2 is given in Appendix A.

2.4 Applications to SDP

In the remainder of the paper, we consider the application of algorithm (4) to the following SDP problem:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \ b^T x \\ &\text{s.t. } S(x) := \sum_{i=1}^{n} A_i x_i - C \succeq 0, \end{aligned} \tag{8}$$

where C, $A_1, \ldots, A_n$ are m by m symmetric matrices and $b \in \mathbb{R}^n$. For $i = 1, \ldots, n$, x_i denotes the ith element of the vector x. Here, $f(x) = b^T x$ and $g(x)$ is the negative of the minimum eigenvalue of $S(x)$, which is a convex function. In this case, we can use $C_f = \|b\|$ and $C_g = \sqrt{n} \max_{1 \le i \le n} \{\|A_i\|_F\}$, where $\|A_i\|_F$ is the Frobenius norm of A_i [11]. The following proposition is helpful to find the constant c in the global error bound [10].

Lemma 3. *Let $g(x)$ be the negative of the minimum eigenvalue of $S(x)$. If there exists a unit vector x such that*

$$\lambda_{\min} \left(\sum_{i=1}^{n} A_i x_i \right) = \tau > 0, \tag{9}$$

then (7) holds with $c = \frac{1}{\tau}$.

In particular, the SDP relaxations of the max-cut and graph partitioning problems satisfy Assumption 1. Thus, if the step size is decreased properly, algorithm (4) finds an optimal solution in the limit.

A subgradient of $g(x)$ can be obtained relatively easily. Let $x \in \mathbb{R}^n$ and $v_1(x)$ be a normalised eigenvector associated with the minimum eigenvalue of $S(x)$. Then, we have

$$\big(-v_1(x)^T A_1 v_1(x), \ldots, -v_1(x)^T A_n v_1(x)\big)^T \in \partial g(x),$$

for any $x \in \mathbb{R}^n$ [4].

Figure 1 shows the behaviour of algorithm (4) with outer approximations $P^{(k)}_{\text{subgrad}}$ and $Q^{(k)}_{\text{subgrad}}$ (6) on an SDP problem given by $n = 2$, $m = 3$, $c = (0, 1)^T$, $C = I$ and randomly generated A_1 and A_2 with entries independently sampled from the normal distribution with mean 0 and standard deviation $1/\sqrt{m}$. The feasible set is shown in a thick line while the solid and open circles indicate the iterates $x^{(k)}$ and $y^{(k)}$, respectively. The outer approximations are drawn in thin lines. We see that as algorithm (4) makes progress, the outer approximation improves, and this helps the algorithm find a near-optimal solution more quickly.

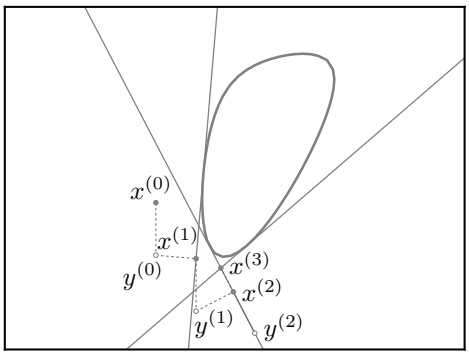

Fig. 1 Behaviour of algorithm (4) on a simple example

We now focus on approaches for the construction of outer approximations of F when algorithm (4) is applied to (8). Let $x \in \mathbb{R}^n$ be such that $S(x) \succeq 0$. Then, for any $B \succeq 0$,

$$B \bullet S(x) \geq 0, \tag{10}$$

where $B \bullet S(x)$ is the Hadamard product of B and $S(x)$. In particular, for any $b \in \mathbb{R}^n$,

$$b^T S(x) b = (bb^T) \bullet S(x) \geq 0.$$

By enforcing (10) for a finite number of matrices $B_1, \ldots, B_r \succeq 0$, we obtain an outer approximation of F:

$$\{x \in \mathbb{R}^n : B_\ell \bullet S(x) \geq 0,\ \ell = 1, \ldots, r\} \supset F.$$

We consider two alternatives to obtain a suitable set of B_ℓ matrices.

Alternative 1: For $j = 1, \ldots, m$, denote by $\lambda_j(x)$ the jth smallest eigenvalue of $S(x)$ and by $v_j(x)$ a normalised eigenvector associated with $\lambda_j(x)$. Let $x^{(k)} \in \mathbb{R}^n$ be the kth iterate. If the smallest eigenvalue $\lambda_1(x^{(k)})$ is negative, then

$$v_1(x^{(k)})^T S(x^{(k)}) v_1(x^{(k)}) = \lambda_1(x^{(k)}) < 0,$$

that is, the constraint

$$v_1(x^{(k)})^T S(x) v_1(x^{(k)}) \geq 0 \tag{11}$$

cuts off $x^{(k)}$. These cuts are used in [1, 16].

We note that cut (11) is obtained from (5) with $\bar{y} = x^{(k)}$ and

$$\bar{s} = \left(-v_1(x^{(k)})^T A_1 v_1(x^{(k)}), \ldots, -v_1(x^{(k)})^T A_n v_1(x^{(k)})\right)^T \in \partial g(x^{(k)}),$$

where $v_1(x^{(k)})$ is a normalised eigenvector associated with $\lambda_1(x^{(k)})$. In fact, it follows that

$$\begin{aligned} g(\bar{y}) + \bar{s}^T(x - \bar{y}) &= -\lambda_1(x^{(k)}) + \bar{s}^T(x - x^{(k)}) \\ &= -\lambda_1(x^{(k)}) - v_1(x^{(k)})^T(S(x) - S(x^{(k)}))v_1(x^{(k)}) \\ &= -\lambda_1(x^{(k)}) - v_1(x^{(k)})^T S(x) v_1(x^{(k)}) + v_1(x^{(k)})^T S(x^{(k)}) v_1(x^{(k)}) \\ &= -\lambda_1(x^{(k)}) - v_1(x^{(k)})^T S(x) v_1(x^{(k)}) + \lambda_1(x^{(k)}) \\ &= -v_1(x^{(k)})^T S(x) v_1(x^{(k)}) \leq 0, \end{aligned}$$

where we used $g(\bar{y}) = g(x^{(k)}) = -\lambda_1(x^{(k)})$ in the first line, and the definition of $S(x)$ in the second line.

Alternative 2: Let $\mathbb{S}_+^{m\times m}$ be the set of $m \times m$ symmetric positive semidefinite matrices. The projection $\Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)}))$ of $S(x^{(k)})$ onto $\mathbb{S}_+^{m\times m}$ is given by

$$\Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)})) = \sum_{j=1}^{m} [\lambda_j(x^{(k)})]^+ v_j(x^{(k)}) v_j(x^{(k)})^T.$$

Let $N(x^{(k)}) := \Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)})) - S(x^{(k)})$. We have

$$N(x^{(k)}) = \sum_{j=1}^{m} [-\lambda_j(x^{(k)})]^+ v_j(x^{(k)}) v_j(x^{(k)})^T, \tag{12}$$

and hence $N(x^{(k)}) \succeq 0$. When $S(x^{(k)}) \not\succeq 0$, $N(x^{(k)})$ defines a supporting hyperplane of $\mathbb{S}_+^{m\times m}$ at $\Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)}))$. That is, for any $S \succeq 0$,

$$N(x^{(k)}) \bullet (S - \Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)}))) \geq 0,$$

and equality holds if $S = \Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)}))$, for example. Because $N(x^{(k)}) \bullet \Pi_{\mathbb{S}_+^{m\times m}}(S(x^{(k)})) = 0$, we have equivalently

$$N(x^{(k)}) \bullet S \geq 0.$$

Thus, for any $x \in F$, we have

$$N(x^{(k)}) \bullet S(x) \geq 0. \tag{13}$$

Constraint (13) cuts off the current iterate $x^{(k)}$ if $S(x^{(k)}) \not\succeq 0$, because in this case $\lambda_1(x^{(k)}) < 0$ and

$$N(x^{(k)}) \bullet S(x^{(k)}) = \sum_{j=1}^{m} \lambda_j(x^{(k)})[-\lambda_j(x^{(k)})]^+ \leq -\lambda_1(x^{(k)})^2 < 0.$$

2.5 Proposed Approaches

We consider three alternative approaches for refining outer approximations $P^{(k)}$ and $Q^{(k)}$. The first approach is based on (11):

$$\begin{aligned} P_{\min}^{(k)} &:= Q_{\min}^{(k-1)} \cap \{x \in \mathbb{R}^n : v_1(x^{(k)})^T S(x) v_1(x^{(k)}) \geq 0\}, \\ Q_{\min}^{(k)} &:= P_{\min}^{(k-1)} \cap \{x \in \mathbb{R}^n : v_1(y^{(k)})^T S(x) v_1(y^{(k)}) \geq 0\}, \qquad k = 0, 1, \ldots, \end{aligned} \tag{14}$$

where we define $Q_{\min}^{(-1)} := \mathbb{R}^n$.

The second approach uses (12):

$$\begin{aligned} P_{\text{comb}}^{(k)} &:= Q_{\text{comb}}^{(k-1)} \cap \{x \in \mathbb{R}^n : N_k(x^{(k)}) \bullet S(x) \geq 0\}, \\ Q_{\text{comb}}^{(k)} &:= P_{\text{comb}}^{(k)} \cap \{x \in \mathbb{R}^n : N_k(y^{(k)}) \bullet S(x) \geq 0\}, \qquad k = 0, 1, \ldots, \end{aligned} \tag{15}$$

where $Q_{\text{comb}}^{(-1)} := \mathbb{R}^n$.

The third approach is a combination of the other two approaches:

$$\begin{aligned} P_{\text{min+comb}}^{(k)} &:= P_{\min}^{(k)} \cap P_{\text{comb}}^{(k)}, \\ Q_{\text{min+comb}}^{(k)} &:= Q_{\min}^{(k)} \cap Q_{\text{comb}}^{(k)}, \qquad k = 0, 1, \ldots. \end{aligned} \tag{16}$$

3 Computational Experiments

We consider two families of SDP problems: the SDP relaxations of the graph partitioning problem and of the max-cut problem. We use instances from the SDPLIB benchmark set [5] and larger instances based on graphs generated using rudy[1], a graph generator by Giovanni Rinaldi. We note that we only solve the relaxations of the problems and that we do not seek integer solutions as in [17, 28].

We ran algorithm (4) until we found a near-optimal solution x:

$$|f^* - b^T x| \leq \varepsilon |f^*|, \qquad |g(x)| \leq 10^{-3}, \tag{17}$$

where ε was set to 10^{-2} or 10^{-3} and f^* was the optimal objective value of problem (8). We tested these conditions using the optimal objective values f^* known for the benchmark instances. The output solution x almost satisfies the constraint of (8) in the sense that the minimum eigenvalue of $S(x)$ is -10^{-3} or greater.

We first compare the performance of algorithm (4) with the three approaches to construct the outer approximations given in (14), (15) and (16). $P_k^{\min}$ and $Q_k^{\min}$ only require eigenvectors associated with the minimum eigenvalues of $S(x^{(k)})$ and $S(y^{(k)})$. We could compute these using the Lanczos method [30]. However, we observed that as the iterate approaches the solution, multiple eigenvalues of $S(x^{(k)})$ and $S(y^{(k)})$ get close to 0 and the performance of the Lanczos method degrades considerably (the convergence rate of the Lanczos method depends on the gap between the smallest and second smallest eigenvalues). Thus, we instead used the `dsyevd` routine in LAPACK [2] to compute all the eigenvectors of the iterates. Our method is implemented in Python.[2] The projection onto an outer approximation is formulated as a quadratic programming problem and solved using the Gurobi Optimizer.[3] The workstation used to run the experiments has two Intel® Xeon® Gold 6226 Processors and 12 modules of 32GB of RAM.

The step size $\alpha^{(k)}$ was adjusted based on the progress of the algorithm. For each iteration, we measured the constraint violation. If $g(y^{(k)}) > 10^{-3}$, call iteration k an infeasible iteration, and otherwise a feasible iteration. If the current iteration was an infeasible iteration, the step size was decreased by a factor of 0.8. If the iteration was a feasible iteration, the objective value $f(y^{(k)})$ was compared with those of the previous feasible iterations. If the current iteration gave the best objective value, the step size was increased by a factor of 1.2. Otherwise, it was decreased by a factor of 0.8. We used $\alpha^{(0)} = 1$. The convergence analysis in Section 2.3 does not cover this adaptive step size rule. However, adjusting the step size is crucial for ensuring the method's robustness and reducing its sensitivity to the initial step size choice. If necessary, one can decrease the step size after a certain number of iterations to enforce the convergence of the algorithm.

---

[1] https://www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz

[2] The source code to run the experiments is available at https://github.com/nsugishita/subgradient_projection_for_sdp.

[3] https://www.gurobi.com/

3.1 Results on SDPLIB Instances

We first report results on the instances from SDPLIB. The sizes of these test instances are given in Table 1.

Table 1 Size of the SDPLIB test instances

Instances	n	m
gpp250-1, gpp250-2, gpp250-3, gpp250-4	250	250
gpp500-1, gpp500-2, gpp500-3, gpp500-4	500	500
mcp250-1, mcp250-2, mcp250-3, mcp250-4	251	250
mcp500-1, mcp500-2, mcp500-3, mcp500-4	501	500

The computational time (all computational times in this paper are in wallclock time) and the number of iterations required to solve the test instances are reported in Tables 2 and 3, respectively. The best results in each line are highlighted in bold. The outer approximations (14), (15) and (16) are labelled `min`, `comb` and `min+comb`, respectively. `min` is worse than `comb` and `min+comb` by a large margin on all the test instances both in terms of the computational time and the number of iterations. In many cases, `min+comb` requires as many or slightly fewer iterations than `comb`. This is expected since `min+comb` uses better approximations than `comb`. However, each step of `min+comb` is more computationally expensive than that of `comb` since `min+comb` uses more cuts than `comb`. The difference in the number of iterations is not large enough and, as a result, `min+comb` often requires slightly more time to solve the test instances.

Table 2 Computational time (s) of algorithm (4) with the three different outer approximations given in (14), (15) and (16)

	$\varepsilon = 10^{-2}$			$\varepsilon = 10^{-3}$		
Instance	min	comb	min+comb	min	comb	min+comb
gpp250-1	31.2	**1.6**	1.8	31.9	**2.2**	9.0
gpp250-2	34.5	**1.2**	1.4	37.3	**1.5**	1.7
gpp250-3	24.8	**1.0**	1.4	28.6	**1.1**	1.3
gpp250-4	10.4	**1.1**	1.2	15.3	**1.1**	1.2
gpp500-1	109.9	5.1	**4.5**	105.3	**6.1**	6.5
gpp500-2	115.5	**3.6**	4.1	121.8	**3.9**	4.4
gpp500-3	108.9	**3.0**	3.4	119.6	**3.3**	3.5
gpp500-4	63.6	**2.9**	3.3	79.0	**2.9**	3.3
mcp250-1	26.1	**1.6**	1.7	28.5	2.0	**1.9**
mcp250-2	35.7	**1.2**	1.3	39.2	**1.3**	**1.3**
mcp250-3	37.6	**1.1**	1.2	44.0	**1.1**	1.2
mcp250-4	41.4	**1.0**	1.2	48.9	**1.0**	1.3
mcp500-1	94.0	**4.4**	5.0	103.0	5.1	**4.9**
mcp500-2	111.5	4.0	**3.8**	127.8	3.9	**3.6**
mcp500-3	130.3	**3.1**	3.8	150.4	**3.0**	3.7
mcp500-4	145.7	**3.2**	3.4	182.4	**3.0**	3.3

Next, we compare the performance of algorithm (4) with the commercial IPM solver MOSEK[4], the open-source ADMM solver COSMO [12] and the open-source

[4] https://www.mosek.com/

Table 3 The number of iterations of algorithm (4) with the three different outer approximations given in (14), (15) and (16)

	$\varepsilon = 10^{-2}$			$\varepsilon = 10^{-3}$		
Instance	min	comb	min+comb	min	comb	min+comb
problem	min	comb	both	min	comb	both
gpp250-1	530	32	**28**	534	**46**	141
gpp250-2	580	**23**	**23**	596	29	**27**
gpp250-3	421	**20**	23	454	**22**	23
gpp250-4	183	22	**21**	255	22	**21**
gpp500-1	772	37	**29**	775	43	**39**
gpp500-2	831	25	**24**	846	28	**27**
gpp500-3	790	**22**	**22**	862	24	**23**
gpp500-4	466	**21**	22	567	**21**	22
mcp250-1	477	33	**30**	511	40	**31**
mcp250-2	621	25	**23**	677	25	**23**
mcp250-3	640	22	**21**	745	22	**21**
mcp250-4	710	**22**	**22**	821	**22**	**22**
mcp500-1	723	**32**	**32**	766	36	**32**
mcp500-2	832	28	**24**	904	28	**24**
mcp500-3	965	**22**	24	1058	**22**	24
mcp500-4	1061	**22**	**22**	1250	**22**	**22**

solver SDPNAL+ [31], which utilizes the augmented Lagrangian method. We set the optimality tolerance to either 10^{-2} or 10^{-3}, and the feasibility tolerance to 10^{-3}. SDPNAL+ was run until it found a solution meeting stopping criterion (17). When we applied stopping criterion (17) to COSMO, it required a significantly longer computational time. We could not use (17) for MOSEK due to the lack of access to the iterate. Therefore, COSMO and MOSEK were run using their default stopping criterion (with suboptimality and infeasibility tolerances adjusted to ε and 10^{-3}, respectively). The computational time is reported in Table 4. MOSEK is the fastest across almost all instances, followed by our method. SDPNAL+ consistently required more time compared to our method. COSMO is faster than our method on the max-cut problem when the tolerance is large ($\varepsilon = 10^{-2}$). In other setups, however, it often requires more computational time. It is particularly slower when it is applied to the graph partitioning problem.

Interestingly, COSMO requires more iterations than our method, even on the max-cut problem with $\varepsilon = 10^{-2}$, where COSMO outperformed our method. For instance, on `mcp250-1` with $\varepsilon = 10^{-2}$, COSMO requires 301 iterations while our method requires only 33 iterations. Both COSMO and our method require the projection of the iterate $S(x^{(k)})$ onto the cone of the positive semidefinite matrices, which typically accounts for a significant portion of the computational time. Therefore, it is somewhat surprising that COSMO requires more iterations but less computational time. This superior computational efficiency of COSMO is partly attributed to the chordal decomposition. When we ran COSMO without the chordal decomposition feature, it often took more time than our method. For example, COSMO without the chordal decomposition required 2.1 seconds to solve `mcp250-1` with $\varepsilon = 10^{-2}$.

Table 4 Computational time (s) of the three methods on the test instances

	$\varepsilon = 10^{-2}$				$\varepsilon = 10^{-3}$			
Instance	comb	MOSEK	COSMO	SDPNAL+	comb	MOSEK	COSMO	SDPNAL+
gpp250-1	1.6	**0.5**	39.1	3.0	2.2	**0.5**	106.8	3.0
gpp250-2	1.2	**0.5**	18.6	3.5	1.5	**0.5**	123.5	3.5
gpp250-3	1.0	**0.5**	29.3	3.4	1.1	**0.5**	77.8	3.4
gpp250-4	1.1	**0.5**	32.8	3.8	1.1	**0.5**	82.8	3.8
gpp500-1	5.1	**3.4**	154.7	12.6	6.1	**3.4**	1565.1	17.1
gpp500-2	**3.6**	**3.6**	294.7	14.4	3.9	**3.6**	766.4	14.4
gpp500-3	**3.0**	3.3	209.9	12.1	**3.3**	**3.3**	497.0	12.1
gpp500-4	**2.9**	3.3	81.2	12.1	**2.9**	3.3	632.5	12.1
mcp250-1	1.6	**0.2**	1.1	2.3	2.0	**0.2**	1.1	3.6
mcp250-2	1.2	**0.2**	0.8	2.6	1.2	**0.2**	2.4	2.6
mcp250-3	1.1	**0.2**	1.0	3.3	1.1	**0.2**	4.9	3.3
mcp250-4	1.0	**0.2**	0.7	3.1	1.0	**0.2**	5.9	3.1
mcp500-1	4.4	1.1	**0.9**	13.1	5.1	**1.2**	2.4	13.1
mcp500-2	4.0	**0.9**	1.7	11.1	4.0	**1.2**	8.8	11.1
mcp500-3	3.1	**0.9**	2.4	11.6	3.1	**1.1**	17.2	11.6
mcp500-4	3.2	**0.9**	2.9	11.9	3.2	**1.2**	18.3	11.9

3.2 Results on Larger Instances

The next experiments use larger instances based on graphs generated using rudy. We varied the graph sizes (number of nodes) from 1000 to 5000, while setting the edge density to either 5% or 10%. For each combination of size and density, we generated four graphs using different random seeds. Both the graph partitioning and max-cut problems were formulated using the same graphs. The performance of our method and of MOSEK are reported in Tables 5 and 6. The best results are highlighted in bold.

Since the two methods used different stopping criteria, we cannot compare their computational times directly. However, across many configurations, we observe that MOSEK's computational time tends to grow more rapidly than that of our method. An exception is the max-cut problem with a density of 10% for which the computational time of our method to find a solution with a small suboptimality (0.1%) increases more rapidly than that of MOSEK.

4 Conclusions

In this paper, we have studied the combination of the subgradient projection method with outer approximations for semidefinite programming problems. We explored several possibilities to construct the outer approximations. Our computational experiments compared the performance of the proposed method with MOSEK, COSMO and SDPNAL+. Our method finds near-optimal solutions to the SDP relaxations of graph partitioning and max-cut problem in SDPLIB more quickly than COSMO and SDPNAL+. However, MOSEK was even faster than our method. The results on larger instances showed the seemingly superior scalability of our method compared with MOSEK.

Although the proposed method uses significantly less memory than interior point methods, it still requires the projection of each iterate onto the cone of positive

Table 5 Computational time (s) of our method and MOSEK on the SDP relaxations of the graph partitioning problems (note: our method and MOSEK use different stopping criteria)

		Density: 5				Density: 10			
		$\varepsilon = 10^{-2}$		$\varepsilon = 10^{-3}$		$\varepsilon = 10^{-2}$		$\varepsilon = 10^{-3}$	
Size	Instance	comb	MOSEK	comb	MOSEK	comb	MOSEK	comb	MOSEK
1000	1	**13.3**	31.1	54.0	**31.1**	**16.2**	31.2	**16.2**	31.2
1000	2	**12.9**	29.4	**12.9**	29.4	**16.5**	30.9	**16.5**	30.9
1000	3	**12.5**	31.2	50.9	**31.2**	**16.8**	31.4	**16.8**	31.4
1000	4	**13.0**	31.2	60.5	**31.2**	**16.9**	30.7	**16.9**	30.7
2000	1	**100.7**	273.2	**100.7**	273.2	**124.2**	250.2	**124.2**	250.2
2000	2	**99.5**	259.2	**99.5**	259.2	**136.7**	263.0	**136.7**	263.0
2000	3	**99.2**	270.0	**99.2**	270.0	**124.3**	268.8	**124.3**	268.8
2000	4	**101.0**	263.4	**101.0**	263.4	**134.9**	264.7	**134.9**	264.7
3000	1	**395.5**	1006.9	**395.5**	1006.9	**504.4**	1211.2	**504.4**	1211.2
3000	2	**409.3**	1073.0	**409.3**	1073.0	**508.0**	1241.1	**508.0**	1241.1
3000	3	**426.7**	1116.5	**426.7**	1116.5	**472.7**	1129.1	**472.7**	1129.1
3000	4	**420.0**	1026.7	**420.0**	1026.7	**514.4**	1179.8	**514.4**	1179.8
4000	1	**1102.5**	2740.1	**1102.5**	2740.1	**1290.7**	2784.1	**1290.7**	2784.1
4000	2	**1042.8**	2736.8	**1042.8**	2736.8	**1270.7**	2631.7	**1270.7**	2631.7
4000	3	**1034.0**	2682.2	**1034.0**	2682.2	**1298.0**	2725.7	**1298.0**	2725.7
4000	4	**1034.2**	2737.1	**1034.2**	2737.1	**1204.7**	2866.4	**1204.7**	2866.4
5000	1	**2195.7**	5784.2	**2195.7**	5784.2	**2460.7**	5462.0	**2460.7**	5462.0
5000	2	**2195.7**	5460.1	**2195.7**	5460.1	**2440.8**	5719.2	**2440.8**	5719.2
5000	3	**2023.7**	5990.1	**2023.7**	5990.1	**2380.0**	5760.6	**2380.0**	5760.6
5000	4	**2204.2**	5187.3	**2204.2**	5187.3	**2464.1**	5763.7	**2464.1**	5763.7

Table 6 Computational time (s) of our method and MOSEK on the SDP relaxations of the max-cut problems (note: our method and MOSEK use different stopping criteria)

		Density: 5				Density: 10			
		$\varepsilon = 10^{-2}$		$\varepsilon = 10^{-3}$		$\varepsilon = 10^{-2}$		$\varepsilon = 10^{-3}$	
Size	Instance	comb	MOSEK	comb	MOSEK	comb	MOSEK	comb	MOSEK
1000	1	12.1	**8.0**	12.1	**8.9**	11.3	**7.6**	11.3	**8.4**
1000	2	11.9	**8.1**	11.9	**8.8**	11.7	**7.6**	11.7	**8.3**
1000	3	12.4	**8.1**	12.4	**8.8**	11.3	**6.5**	11.3	**8.3**
1000	4	11.8	**8.1**	11.8	**8.8**	10.4	**6.7**	10.4	**8.6**
2000	1	**63.3**	65.5	**63.3**	68.8	66.7	**57.3**	66.7	**66.5**
2000	2	65.0	**62.1**	**65.0**	69.5	67.0	**62.8**	**67.0**	67.5
2000	3	**65.8**	67.5	**65.8**	69.8	64.3	**61.9**	**64.3**	68.0
2000	4	65.6	**62.0**	**65.6**	71.5	64.5	**63.2**	64.5	**62.8**
3000	1	**215.9**	238.0	**215.9**	258.0	**224.7**	262.4	**224.7**	289.4
3000	2	**235.9**	240.8	**235.9**	254.5	**228.5**	263.0	**228.5**	278.8
3000	3	**224.8**	231.5	**224.8**	249.6	**231.8**	268.0	**231.8**	277.6
3000	4	241.4	**236.0**	**241.4**	249.3	**222.3**	258.0	**222.3**	284.7
4000	1	**543.2**	571.3	**543.2**	652.2	**546.3**	664.4	**546.3**	711.0
4000	2	**479.8**	562.6	**479.8**	595.4	**528.0**	661.9	**528.0**	684.0
4000	3	**504.4**	567.6	**504.4**	618.5	**539.4**	658.7	**539.4**	703.7
4000	4	**564.9**	566.3	**564.9**	636.6	**545.5**	659.5	**545.5**	719.7
5000	1	**1062.7**	1136.0	**1062.7**	1223.1	**1007.8**	1293.4	1629.2	**1424.8**
5000	2	**934.3**	1190.3	**934.3**	1207.7	**1019.8**	1348.3	1783.1	**1397.3**
5000	3	**1061.5**	1125.8	**1061.5**	1248.6	**1064.2**	1338.9	1739.3	**1370.9**
5000	4	**992.5**	1133.0	**992.5**	1221.8	**990.1**	1342.4	**1088.6**	1415.3

semidefinite matrices. This could be prohibitively expensive when solving large-scale problems. A similar limitation applies to ADMM, which requires the projection of the iterate at each iteration [12]. Rontsis et al [29] show that the use of a faster, warm-started routine to compute the projection of the iterate can speed up ADMM. It would be of interest to consider a similar extension of the proposed method to solve very large instances.

Another limitation of the proposed method is the lack of reliable stopping criteria. Our computational experiments used the optimal objective values computed a priori. However, in practice, these values are normally unknown, making it essential to develop practical stopping criteria.

Finally, the analysis presented in this paper is asymptotic. It is of interest to study the non-asymptotic convergence rate of the method.

Declarations

Data availability: All data generated or analysed during this study are included in [5] and at https://github.com/nsugishita/subgradient_projection_for_sdp.

Acknowledgments

The authors gratefully acknowledge two reviewers for their constructive feedback, which greatly contributed to the improvement of the paper.

Appendix A Proof of Proposition 2

The proof of Proposition 2 is similar to the proofs in [20]. To make this paper self-contained, it is provided in this appendix.

Throughout this appendix, we suppose Assumption 1 holds. In particular, Assumption 1.2 implies C_f-Lipschitz continuity of f and C_g-Lipschitz continuity of g (Theorem 3.61 in [4]).

Lemma 4. *For any $\bar{x} \in F$, $z \in F$ and $\eta > 0$,*

$$\|y^{(k)} - \bar{x}\|^2 \leq \|x^{(k)} - \bar{x}\|^2 + 2cC_f[g(x^{(k)})]^+\alpha^{(k)} - 2\alpha^{(k)}(f(z^{(k)}) - f(\bar{x})) + (\alpha^{(k)})^2 C_f^2,$$

where $z^{(k)} = \Pi_F(x^{(k)})$.

Proof. For any $\bar{x} \in F \in P^{(k)}$, it follows that

$$\begin{aligned}\|y^{(k)} - \bar{x}\|^2 &\leq \|x^{(k)} - \alpha^{(k)} t^{(k)} - \bar{x}\|^2 \\ &= \|x^{(k)} - \bar{x}\|^2 - 2\alpha^{(k)}(t^{(k)})^T(x^{(k)} - \bar{x}) + (\alpha^{(k)})^2\|t^{(k)}\|^2. \end{aligned} \tag{A1}$$

The convexity of f gives $(t^{(k)})^T(x^{(k)} - \bar{x}) \geq f(x^{(k)}) - f(\bar{x})$. Therefore,

$$\|y^{(k)} - \bar{x}\|^2 \leq \|x^{(k)} - \bar{x}\|^2 - 2\alpha^{(k)}(f(x^{(k)}) - f(\bar{x})) + (\alpha^{(k)})^2 C_f^2.$$

Let $z^{(k)}$ be the projection of $x^{(k)}$ onto F. Since f is C_f-Lipschitz continuous,

$$\begin{aligned}
-2\alpha^{(k)}(f(x^{(k)}) - f(\bar{x})) &= -2\alpha^{(k)}(f(x^{(k)}) - f(z^{(k)})) - 2\alpha^{(k)}(f(z^{(k)}) - f(\bar{x})) \\
&\leq 2\alpha^{(k)} C_f \|x^{(k)} - z^{(k)}\| - 2\alpha^{(k)}(f(z^{(k)}) - f(\bar{x})) \\
&\leq 2\alpha^{(k)} C_f \mathrm{dist}(x^{(k)}, F) - 2\alpha^{(k)}(f(z^{(k)}) - f(\bar{x})) \\
&\leq 2\alpha^{(k)} c C_f [g(x^{(k)})]^+ - 2\alpha^{(k)}(f(z^{(k)}) - f(\bar{x})),
\end{aligned}$$

where the last inequality follows from the global error bound. Combining with (A1) gives the desired result. □

Lemma 5. *For any $\bar{x} \in F$ and $k \geq 0$,*

$$\|x^{(k+1)} - \bar{x}\|^2 \leq \|x^{(k)} - \bar{x}\|^2 - 2\alpha^{(k)}(f(z^{(k)}) - f(\bar{x})) - \frac{1}{2C_g^2}([g(x^{(k)})]^+)^2 + D(\alpha^{(k)})^2,$$

where $D = C_f^2(2(cC_g + 1)^2 + 1)$ and $z^{(k)} = \Pi_F(x^{(k)})$.

Proof. For any $\bar{x} \in F$, the definition of $x^{(k+1)}$ gives

$$\begin{aligned}
\|x^{(k+1)} - \bar{x}\|^2 &\leq \left\| y^{(k)} - \frac{[g(y^{(k)})]^+}{\|d^{(k)}\|^2} d^{(k)} - \bar{x} \right\|^2 \\
&\leq \|y^{(k)} - \bar{x}\|^2 - \frac{([g(y^{(k)})]^+)^2}{\|d_k\|^2} \\
&\leq \|y^{(k)} - \bar{x}\|^2 - \frac{([g(y^{(k)})]^+)^2}{C_g^2}.
\end{aligned}$$

We have

$$\begin{aligned}
([g(y^{(k)})]^+)^2 &= ([g(y^{(k)})]^+ - [g(x^{(k)})]^+ + [g(x^{(k)})]^+)^2 \\
&\geq 2([g(y^{(k)})]^+ - [g(x^{(k)})]^+)[g(x^{(k)})]^+ + ([g(x^{(k)})]^+)^2.
\end{aligned}$$

Using C_g-Lipschitz continuity of g and the definition of $y^{(k)}$, we get

$$\begin{aligned}
([g(y^{(k)})]^+)^2 &\geq -2C_g \|y^{(k)} - x^{(k)}\| [g(x^{(k)})]^+ + ([g(x^{(k)})]^+)^2 \\
&\geq -2C_g \alpha^{(k)} C_f [g(x^{(k)})]^+ + ([g(x^{(k)})]^+)^2.
\end{aligned}$$

Thus,

$$\|x^{(k+1)} - \bar{x}\|^2 \leq \|y^{(k)} - \bar{x}\|^2 + \frac{2C_f}{C_g}\alpha^{(k)}[g(x^{(k)})]^+ - \frac{1}{C_g^2}([g(x^{(k)})]^+)^2.$$

From Lemma 4 it follows that

$$\begin{aligned}\|x^{(k+1)}-\bar{x}\|^2 &\le \|x^{(k)}-\bar{x}\|^2 + 2\alpha^{(k)}C_f c[g(x^{(k)})]^+ - 2\alpha^{(k)}(f(z^{(k)})-f(\bar{x})) + (\alpha^{(k)})^2 C_f^2 \\ &\quad + \frac{2C_f}{C_g}\alpha^{(k)}[g(x^{(k)})]^+ - \frac{1}{C_g^2}([g(x^{(k)})]^+)^2 \\ &\le \|x^{(k)}-\bar{x}\|^2 - 2\alpha^{(k)}(f(z^{(k)})-f(\bar{x})) + (\alpha^{(k)})^2 C_f^2 \\ &\quad - \frac{1}{C_g^2}([g(x^{(k)})]^+)^2 + 2\frac{C_f}{C_g}(cC_g+1)\alpha^{(k)}[g(x^{(k)})]^+.\end{aligned}$$

We can get an upper bound of the last term as

$$\begin{aligned}&2\frac{C_f}{C_g}(cC_g+1)\alpha^{(k)}[g(x^{(k)})]^+ \\ &\quad \le 2\frac{C_f}{C_g}(cC_g+1)\alpha^{(k)}[g(x^{(k)})]^+ + \left(\sqrt{2}C_f(cC_g+1)\alpha^{(k)} - \frac{1}{\sqrt{2}C_g}[g(x^{(k)})]^+\right)^2 \\ &\quad = 2C_f^2(cC_g+1)^2(\alpha^{(k)})^2 + \frac{1}{2C_g^2}([g(x^{(k)})]^+)^2.\end{aligned}$$

Combining the two inequalities gives the desired result. □

Proof of Proposition 2. Let $x^* \in F$ be any optimal solution to (1). Invoking Lemma 5 repeatedly with $\bar{x} = x^*$ we get

$$\|x^{(k)}-x^*\| \le \|x^{(0)}-x^*\| + D\sum_{i=0}^{k-1}(\alpha^{(i)})^2 \le \|x^{(0)}-x^*\| + D\sum_{i=0}^{\infty}(\alpha^{(i)})^2.$$

Therefore, $\{x^{(k)}\}$ is bounded. Since $z^{(k)} := \Pi_F(x^{(k)})$, $\{z^{(k)}\}$ is also bounded.

Similarly, use Lemma 5 with $\bar{x} = x^*$ to get

$$\sum_{i=0}^{\infty} 2\alpha^{(i)}(f(z^{(i)})-f^*) \le \|x^{(0)}-x^*\|^2 + D\sum_{i=0}^{\infty}(\alpha^{(i)})^2 < \infty,$$

where $f^* = f(x^*)$ is the optimal objective value of (1). In the light of $\sum_{i=0}^{\infty}\alpha^{(i)} = \infty$, it follows that

$$\liminf_{i\to\infty}(f(z^{(i)})-f^*) = 0.$$

With the boundedness of $\{z^{(k)}\}$, $\{z^{(k)}\} \subset F$ and the continuity of f, this implies that $z^{(k)}$ has a subsequence converging to some optimal solution.

Use Lemma 5 again to get

$$\sum_{i=0}^{\infty}\frac{1}{2C_g^2}([g(x^{(i)})]^+)^2 \le \|x^{(0)}-x^*\|^2 + D\sum_{i=0}^{\infty}(\alpha^{(i)})^2 < \infty.$$

Therefore,

$$\lim_{i\to\infty} [g(x^{(i)})]^+ = 0.$$

With the global error bound, we obtain that $\|x^{(k)} - z^{(k)}\| = \text{dist}(x^{(k)}, F) \to 0$ as $k \to \infty$. Thus, $\{x^{(k)}\}$ also has a subsequence converging to some optimal solution. Now invoke Proposition 1.3 of Correa and Lemaréchal [9] to complete the proof. □

References

[1] Ahmadi AA, Dash S, Hall G (2017) Optimization over structured subsets of positive semidefinite matrices via column generation. Discrete Optimization 24:129–151. https://doi.org/https://doi.org/10.1016/j.disopt.2016.04.004

[2] Anderson E, Bai Z, Bischof C, et al (1999) LAPACK users' guide, 3rd edn. Software, environments, tools ; 9, Society for Industrial and Applied Mathematics SIAM, Philadelphia, https://doi.org/https://doi.org/10.1137/1.9780898719604

[3] Anjos MF, Lasserre JB (2012) Handbook on Semidefinite, Conic and Polynomial Optimization, International Series in Operations Research & Management Science, vol 166. Springer Nature, New York, NY, https://doi.org/https://doi.org/10.1007/978-1-4614-0769-0

[4] Beck A (2017) First-Order Methods in Optimization. Society for Industrial and Applied Mathematics, Philadelphia, https://doi.org/10.1137/1.9781611974997

[5] Borchers B (1999) SDPLIB 1.2, a library of semidefinite programming test problems. Optimization Methods and Software 11(1-4):683–690. https://doi.org/10.1080/10556789908805769

[6] Boyd S, El Ghaoui L, Feron E, et al (1994) Linear Matrix Inequalities in System and Control Theory. SIAM studies in applied mathematics; vol. 15, Society for Industrial and Applied Mathematics, Philadelphia, https://doi.org/https://doi.org/10.1137/1.9781611970777

[7] Boyd S, Parikh N, Chu E, et al (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning 3(1):1–122. https://doi.org/https://doi.org/10.1007/s12532-020-00178-3

[8] Burer S, Monteiro RDC (2003) A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. Mathematical programming 95(2):329–357. https://doi.org/https://doi.org/10.1007/s10107-002-0352-8

[9] Correa R, Lemaréchal C (1993) Convergence of some algorithms for convex minimization. Mathematical Programming 62(1-3):261–275

[10] Deng S, Hu H (1999) Computable error bounds for semidefinite programming. Journal of Global Optimization 14(2):105–

[11] Fletcher R (1985) Semi-definite matrix constraints in optimization. SIAM Journal on Control and Optimization 23(4):493–513

[12] Garstka M, Cannon M, Goulart P (2021) COSMO: A conic operator splitting method for convex conic problems. Journal of optimization theory and applications 190(3):779–810. https://doi.org/https://doi.org/10.1007/s10957-021-01896-x

[13] Goemans M, Williamson D (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM 42(6):1115–1145. https://doi.org/https://doi.org/10.1145/227683.227684

[14] Goffin JL, Vial JP (1993) On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. Mathematical Programming 60(1):81–92

[15] Helmberg C, Rendl F (2000) A spectral bundle method for semidefinite programming. SIAM Journal on Optimization 10(3):673–696. https://doi.org/10.1137/S1052623497328987

[16] Krishnan K, Mitchell J (2006) A unifying framework for several cutting plane methods for semidefinite programming. Optimization Methods & Software 21(1):57–74. https://doi.org/https://doi.org/10.1080/10556780500065283

[17] Krislock N, Malick J, Roupin F (2017) Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems. ACM Transactions on Mathematical Software 43(4). https://doi.org/10.1145/3005345, URL https://doi.org/10.1145/3005345

[18] Lewis AS, Pang J (1998) Error Bounds for Convex Inequality Systems, Springer US, Boston, MA, pp 75–110. https://doi.org/10.1007/978-1-4613-3341-8_3, URL https://doi.org/10.1007/978-1-4613-3341-8_3

[19] Majumdar A, Hall G, Ahmadi A (2020) Recent scalability improvements for semidefinite programming with applications in machine learning, control, and robotics. Annual Review of Control, Robotics, and Autonomous Systems 3(1):331–360. https://doi.org/10.1146/annurev-control-091819-074326

[20] Nedic A (2011) Random algorithms for convex minimization problems. Mathematical Programming 129(2):225–253. https://doi.org/https://doi.org/10.1007/s10107-011-0468-9

[21] Nesterov Y (2018) Lectures on Convex Optimization, 2nd edn. Springer Optimization and Its Applications, 137, Springer International Publishing, Cham, https://doi.org/https://doi.org/10.1007/978-3-319-91578-4

[22] Nesterov Y, Nemirovskii A (1994) Interior-point polynomial algorithms in convex programming. SIAM studies in applied and numerical mathematics ; 13, Society for Industrial and Applied Mathematics, Philadelphia, https://doi.org/https://doi.org/10.1137/1.9781611970791

[23] Neto ESH, de Pierro AR (2009) Incremental subgradients for constrained convex optimization: A unified framework and new methods. SIAM Journal on Optimization 20(3):1547–1572. https://doi.org/https://doi.org/10.1137/070711712

[24] Oskoorouchi MR, Mitchell JE (2009) A second-order cone cutting surface method: complexity and application. Computational Optimization and Applications 43(3):379–409

[25] O'Donoghue B, Chu E, Parikh N, et al (2016) Conic optimization via operator splitting and homogeneous self-dual embedding. Journal of Optimization Theory and Applications 169(3):1042–1068. https://doi.org/https://doi.org/10.1007/s10957-016-0892-3

[26] Polyak BT (2001) Random algorithms for solving convex inequalities. In: Butnariu D, Censor Y, Reich S (eds) Inherently Parallel Algorithms in Feasibility and Optimization and their Applications, Studies in Computational Mathematics, vol 8. Elsevier, Amsterdam, p 409–422, https://doi.org/https://doi.org/10.1016/S1570-579X(01)80024-0

[27] Povh J, Rendl F, Wiegele A (2006) A boundary point method to solve semidefinite programs. Computing 78(3):277–286. https://doi.org/https://doi.org/10.1007/s00607-006-0182-2

[28] Rendl F, Rinaldi G, Wiegele A (2010) Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. Math Programming 121(2):307. https://doi.org/https://doi.org/10.1007/s10107-008-0235-8

[29] Rontsis N, Goulart P, Nakatsukasa Y (2022) Efficient semidefinite programming with approximate ADMM. Journal of Optimization Theory and Applications 192(1):292–320. https://doi.org/https://doi.org/10.1007/s10957-021-01971-3

[30] Saad Y (2011) Numerical Methods for Large Eigenvalue Problems. Society for Industrial and Applied Mathematics, Philadelphia, https://doi.org/10.1137/1.9781611970739

[31] Sun D, Kim-Chuan T, Yuan Y, et al (2020) SDPNAL+: A matlab software for semidefinite programming with bound constraints (version 1.0). Optimization Methods and Software 35(1):87–115. https://doi.org/10.1080/10556788.

2019.1576176

[32] Vandenberghe L, Boyd S (1996) Semidefinite programming. SIAM review 38(1):49–95. https://doi.org/https://doi.org/10.1137/1038003