SIAM J. MATH. DATA SCI.
Vol. 3, No. 1, pp. 171–200

Scalable Semidefinite Programming*

Alp Yurtsever†, Joel A. Tropp‡, Olivier Fercoq§, Madeleine Udell¶, and Volkan Cevher‖

Abstract. Semidefinite programming (SDP) is a powerful framework from convex optimization that has striking potential for data science applications. This paper develops a provably correct randomized algorithm for solving large, weakly constrained SDP problems by economizing on the storage and arithmetic costs. Numerical evidence shows that the method is effective for a range of applications, including relaxations of MaxCut, abstract phase retrieval, and quadratic assignment. Running on a laptop equivalent, the algorithm can handle SDP instances where the matrix variable has over 10^{14} entries.

Key words. augmented Lagrangian, conditional gradient method, convex optimization, dimension reduction, first-order method, randomized linear algebra, semidefinite programming, sketching

AMS subject classifications. 90C22, 65K05, 65F99

DOI. 10.1137/19M1305045

1. Motivation.

For a spectrum of challenges in data science, methodologies based on semidefinite programming offer remarkable performance both in theory and for small problem instances. Even so, practitioners often critique this approach by asserting that it is impossible to solve semidefinite programs (SDPs) at the scale demanded by real-world applications. We would like to argue against this article of conventional wisdom.

This paper proposes a new algorithm, called SketchyCGAL, that can solve very large SDPs to moderate accuracy. The algorithm marries a primal-dual optimization technique [102] to

*Received by the editors December 9, 2019; accepted for publication (in revised form) November 16, 2020; published electronically February 8, 2021.

https://doi.org/10.1137/19M1305045

Funding: The first author received support from Early Postdoc.Mobility Fellowship P2ELP2 187955 from Swiss National Science Foundation and partial postdoctoral support from NSF-CAREER grant IIS-1846088. The second author was supported by ONR awards N00014-11-1-0025, N00014-17-1-2146, and N00014-18-1-2363. The fourth author was supported by DARPA award FA8750-17-2-0101. The first and fifth authors have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement 725594 (time-data) and the Swiss National Science Foundation (SNSF) under grant 200021 178865/1.

†Laboratory for Information and Inference Systems, Department of Electrical Engineering, École Polytechnique Fédérale de Lausanne, Lausanne 1015, Switzerland, and Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (alp.yurtsever@epfl.ch, alpy@mit.edu, https://lions.epfl.ch/).

‡Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125-5000 USA (jtropp@cms.caltech.edu, http://users.cms.caltech.edu/~jtropp).

§Laboratoire Traitement et Communication d'Information, Télécom Paris, Institut Polytechnique de Paris, Palaiseau 91120, France (olivier.fercoq@telecom-paris.fr, https://perso.telecom-paristech.fr/ofercoq/).

¶Department of Operations Research and Information Engineering, Cornell University, Ithaca, NY 14850 USA (udell@cornell.edu).

‖Laboratory for Information and Inference Systems, Department of Electrical Engineering, École Polytechnique Fédérale de Lausanne, Lausanne 1015, Switzerland (volkan.cevher@epfl.ch)

a randomized sketch for low-rank matrix approximation [93]. In each iteration, the primary expense is one low-precision randomized eigenvector calculation [57].

For every standard-form SDP that satisfies strong duality, SketchyCGAL provably converges to a near-optimal low-rank approximation of a solution. The algorithm uses limited arithmetic and minimal storage. It is most effective for weakly constrained problems whose solutions are nearly low rank. In contrast, given the same computational resources, other methods for this class of problems may fail. In particular, SketchyCGAL needs far less storage than the Burer–Monteiro factorization heuristic [24, 20] for certain problem instances [97].

In addition to the theoretical guarantees, we offer evidence that SketchyCGAL is a practical optimization algorithm. For example, on a laptop equivalent, we can solve the MaxCut SDP for a sparse graph with over 20 million vertices, where the matrix variable has over 10^{14} entries. We also tackle large phase retrieval problems arising from Fourier ptychography [48], as well as relaxations [108, 23] of the quadratic assignment problem (QAP).

1.1. Example: The maximum cut in a graph.

To begin, we derive a fundamental SDP [34, 35, 44] that arises in combinatorial optimization. This example highlights why large SDPs are hard to solve, and it illustrates the potential of our approach.

1.1.1. MaxCut.

Consider an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ comprising a vertex set $\mathsf{V} = \{1, \dots, n\}$ and a set E of m edges. The combinatorial Laplacian of the graph is the real positive-semidefinite (psd) matrix

$$\boldsymbol{L} := \sum\nolimits_{\{i,j\} \in \mathsf{E}} (\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^* \in \mathbb{R}^{n \times n}, \tag{1.1}$$

where $\mathbf{e}_i \in \mathbb{R}^n$ denotes the ith standard basis vector and * refers to the (conjugate) transpose of a matrix or vector. We can search for a maximum-weight cut in the graph by solving

$$\text{maximize} \quad \boldsymbol{\chi}^* \boldsymbol{L} \boldsymbol{\chi} \quad \text{subject to} \quad \boldsymbol{\chi} \in \{\pm 1\}^n. \tag{1.2}$$

Unfortunately, the formulation (1.2) is NP-hard [55]. One remedy is to relax it to an SDP.

Consider the matrix $\boldsymbol{X} = \boldsymbol{\chi}\boldsymbol{\chi}^*$ where $\boldsymbol{\chi} \in \{\pm 1\}^n$. The matrix $\boldsymbol{X}$ is psd, its diagonal entries equal one, and it has rank one. We can express the MaxCut problem (1.2) in terms of the matrix $\boldsymbol{X}$ by rewriting the objective as a trace. Bringing forward the implicit constraints on $\boldsymbol{X}$ and dropping the rank constraint, we arrive at the MaxCut SDP:

$$\text{maximize} \quad \operatorname{tr}(\boldsymbol{L}\boldsymbol{X}) \quad \text{subject to} \quad \operatorname{diag}(\boldsymbol{X}) = \mathbf{1}, \quad \boldsymbol{X} \text{ is psd}. \tag{1.3}$$

As usual, diag extracts the diagonal of a matrix as a vector, and $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones.

The matrix solution $\boldsymbol{X}_\star$ of (1.3) does not immediately yield a cut. Let $\boldsymbol{x}_\star \boldsymbol{x}_\star^*$ be a best rank-one approximation of $\boldsymbol{X}_\star$ with respect to the Frobenius norm. Then the vector $\boldsymbol{\chi}_\star = \operatorname{sgn}(\boldsymbol{x}_\star)$ is a valid cut. In many cases, the cut $\boldsymbol{\chi}_\star$ yields an excellent solution to the discrete MaxCut problem (1.2). We can also use $\boldsymbol{X}_\star$ to compute a cut that is provably near-optimal via a more involved randomized rounding procedure [44].

1.1.2. What's the issue?

We specify an instance of the MaxCut SDP (1.3) by means of the Laplacian $\boldsymbol{L}$ of the graph, which has $O(m+n)$ nonzero entries. Our goal is to compute a best rank-one approximation of a solution to the SDP, which has $O(n)$ degrees of freedom. In other words, the total cost of representing the input and output of the problem is $O(m+n)$. Sadly, the matrix variable in (1.3) seems to require storage $O(n^2)$. For example, a graph G with one million vertices leads to an SDP (1.3) with a trillion real variables.

Storage is one of the main reasons that it has been challenging to solve large instances of the MaxCut SDP reliably. Undeterred, we raise a question:

> *Can we provably find a best rank-one approximation of a solution to the MaxCut SDP (1.3) with storage $O(m+n)$? Can we achieve working storage $O(n)$?*

We are not aware of any correct algorithm that can solve an arbitrary instance of (1.3) with a working storage guarantee better than $\Theta(\min\{m, n^{3/2}\})$; see section 8.

In addition to the limit on storage, a good algorithm should interact with the Laplacian $\boldsymbol{L}$ only through noninvasive, low-cost operations, such as matrix-vector multiplication.

1.1.3. A storage-optimal algorithm for the MaxCut SDP. Surprisingly, it is possible to achieve all the goals announced in the last subsection.

Theorem 1.1 (MaxCut via SketchyCGAL). *For any $\varepsilon, \zeta > 0$ and any Laplacian $\boldsymbol{L} \in \mathbb{R}^{n\times n}$, the SketchyCGAL algorithm computes a $(1+\zeta)$-optimal rank-one approximation of an ε-optimal point of (1.3); see subsection 2.2. The working storage is $O(n/\zeta)$. The algorithm performs at most $\tilde{O}(\varepsilon^{-2.5})$ matrix-vector multiplies with the Laplacian $\boldsymbol{L}$, plus lower-order arithmetic. The algorithm is randomized; it succeeds with high probability over its random choices.*

Theorem 1.1 follows from Theorem 6.3. As usual, $\tilde{O}$ suppresses constants and logarithmic factors. In contrast to Burer–Monteiro methods [97] and to the approximate complementarity paradigm [36], SketchyCGAL provably succeeds for every instance of MaxCut.

In our experience, SketchyCGAL works *better* than the theorem says. Figure 1.1 compares its scalability with four standard SDP solvers on a laptop equivalent (details are in subsection 7.2).

1.2. A model problem. SketchyCGAL can solve all standard-form SDPs that satisfy strong duality. To simplify parts of the presentation, we focus on a model problem that includes an extra trace constraint. Section SM4 extends SketchyCGAL to a more expressive problem template that includes standard-form SDPs with additional (conic) inequality constraints.

1.2.1. The trace-constrained SDP. We work over the field $\mathbb{F} = \mathbb{R}$ or $\mathbb{F} = \mathbb{C}$. For each $n \in \mathbb{N}$, define the set $\mathbb{S}_n := \mathbb{S}_n(\mathbb{F})$ of (conjugate) symmetric $n \times n$ matrices with entries in $\mathbb{F}$.

Introduce the set of $n \times n$ psd matrices with trace one:

$$\boldsymbol{\Delta}_n := \{\boldsymbol{X} \in \mathbb{S}_n : \operatorname{tr} \boldsymbol{X} = 1 \text{ and } \boldsymbol{X} \text{ is psd}\}. \tag{1.4}$$

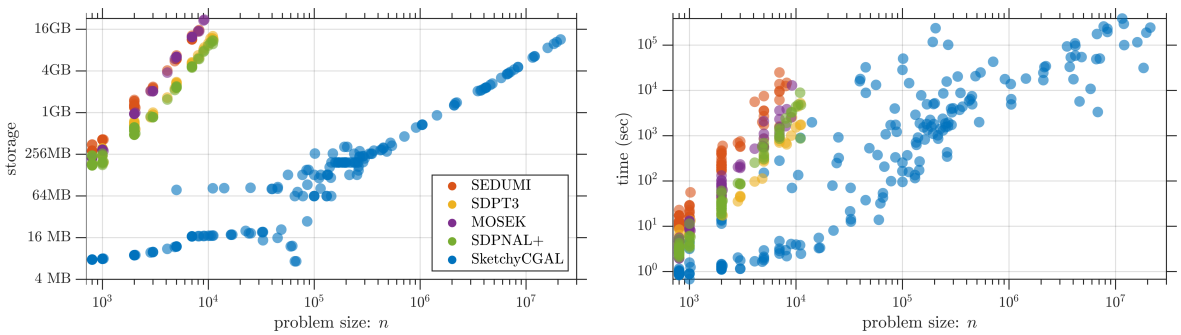


Figure 1.1. *MaxCut SDP: Scalability. Storage cost (left) and runtime (right) of SketchyCGAL with sketch size $R = 10$ as compared with four standard SDP solvers. The relative error tolerance for each solver is 10^{-1}. Each marker represents one dataset. See subsection 7.2.3 for details.*

Our model problem is the following trace-constrained SDP:

$$\begin{aligned} &\text{minimize} && \operatorname{tr}(\boldsymbol{C}\boldsymbol{X}) \\ &\text{subject to} && \operatorname{tr}(\boldsymbol{A}_i\boldsymbol{X}) = b_i \quad \text{for } i = 1, \dots, d \quad \text{and} \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \end{aligned} \tag{1.5}$$

The trace parameter $\alpha > 0$, each matrix $\boldsymbol{C}, \boldsymbol{A}_1, \dots, \boldsymbol{A}_d \in \mathbb{S}_n$, and $b_1, \dots, b_d \in \mathbb{R}$. We always assume that (1.5) satisfies strong duality with its standard-form dual problem.

To solve a general standard-form SDP, we replace the inclusion $\boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n$ with the constraints that $\boldsymbol{X}$ is psd and $\operatorname{tr}(\boldsymbol{X}) \le \alpha$ for a large enough parameter α.

1.2.2. Applications. The model problem (1.5) has diverse applications in statistics, signal processing, quantum information theory, combinatorics, and beyond. Evidently, the MaxCut SDP (1.3) is a special case. The template (1.5) includes problems in computer vision [49], in microscopy [48], and in robotics [84]. It also supports contemporary machine learning tasks, such as certifying robustness of neural networks [82]. There is a galaxy of other examples.

1.3. Complexity of SDP formulations and solutions. This section describes some special features that commonly appear in large, real-world SDPs. Our algorithm will take advantage of these features, even as it provides guarantees for every instance of (1.5).

1.3.1. Structure of the problem data. The matrices $\boldsymbol{C}$ and $\boldsymbol{A}_i$ that appear in (1.5) are often highly structured, or sparse, or have low rank. As such, we can specify the SDP using a small amount of information. In our work, we exploit this property by treating the problem data for the SDP (1.5) as a collection of black boxes that support specific linear algebraic operations. The algorithm for solving the SDP only needs to access the data via these black boxes, and we can insist that these subroutines are implemented efficiently.

1.3.2. Low-rank solutions of SDPs. We will also capitalize on the fact that SDPs frequently have low-rank solutions, or the solutions are approximated well by low-rank matrices. There are several reasons why we can make this surmise.

Weakly constrained SDPs. First, many SDPs have low-rank solutions just because they are weakly constrained. That is, the number d of linear equalities in (1.5) is much smaller than the number n^2 of components in the matrix variable. This situation often occurs in signal processing and statistics problems, where d reflects the amount of measured data. SketchyCGAL is designed for weakly constrained SDPs, but it does not require this property.

A weakly constrained SDP has at least one low-rank solution because of the geometry of the set of psd matrices; see [16, Prop. II(13.4) and Prob. II.14.5] and [79].

Fact 1.2 (Barvinok–Pataki). *When $\mathbb{F} = \mathbb{R}$, the SDP (1.5) has a solution with rank $r \le \sqrt{2(d+1)}$. When $\mathbb{F} = \mathbb{C}$, there is a solution with rank $r \le \sqrt{d+1}$.*

For example, the MaxCut SDP (1.3) admits a solution with rank $\sqrt{2(n+1)}$.

Although a weakly constrained SDP can have solutions with high rank, a *generic* weakly constrained SDP has a unique solution, which must be low rank [5].

Fact 1.3 (Alizadeh, Haeberly, and Overton [5]). *Let $\mathbb{F} = \mathbb{R}$. Except for a set of matrices $\{\boldsymbol{C}, \boldsymbol{A}_1, \dots, \boldsymbol{A}_d\}$ with measure zero, the solution set of the SDP (1.5) is a unique matrix with rank $r \le \sqrt{2(d+1)}$.*

Matrix rank minimization. Second, some SDPs are *designed* to produce a low-rank matrix that satisfies a system of linear matrix equations. This idea can be traced to the control theory literature [68, 78], and it was explored thoroughly in Fazel's thesis [37]. Early applications include Euclidean distance matrix completion [2] and collaborative filtering [89]. Extensive empirical work indicates that these SDPs often produce low-rank solutions.

Structural properties. There are other reasons that an SDP must have a low-rank solution. For instance, consider the optimal power flow SDPs developed by Lavaei and Low [60], where the rank of the solution is controlled by the geometry of the power grid.

1.3.3. Algorithms? To summarize, many realistic SDPs have structured data, and they admit solutions that are (close to) low rank. Are there algorithms that can exploit these features? Although there are a number of methods that attempt to do so, none can provably solve every SDP with limited arithmetic and minimal storage. See section 8 for related work.

Why has it been so difficult to develop provably correct algorithms for finding low-rank solutions to structured SDPs? Most approaches that try to control the rank run headlong into a computational complexity barrier: for any fixed rank parameter r, it is NP-hard to solve the model problem (1.5) if the variable $\boldsymbol{X}$ is also constrained to be a rank-r matrix [37, p. 7].

To escape this sticky fact, we revise the computational goal, following [106]. The key insight is to seek a rank-r matrix that *approximates a solution* to (1.5). See subsection 2.2 for a detailed explanation. This shift in perspective opens up new algorithmic prospects.

1.4. Contributions. Inspired by [106], we derive an algorithm that harnesses the favorable properties common in large SDPs. The SketchyCGAL algorithm solves the SDP (1.5) using a primal-dual optimization method [102] developed by a subset of the authors. Each iteration requires one coarse eigenvector computation [57] and leads to a rank-one update of the psd matrix variable. Instead of storing this matrix, we maintain a compressed representation by means of a matrix sketching technique [93]. After the optimization algorithm terminates, we extract from the sketch a low-rank approximation of a solution of the SDP. This idea leads to a practical, provably correct SDP solver that economizes on storage and arithmetic.

Theorem 1.4 (the model problem via SketchyCGAL). *Assume that the model problem (1.5) satisfies strong duality. For any $\varepsilon, \zeta > 0$ and any rank parameter r, the SketchyCGAL algorithm computes a $(1+\zeta)$-optimal rank-r approximation of an ε-optimal point of (1.5); see subsection 2.2. The storage cost is $\boldsymbol{O}(d + rn/\zeta)$. Most of the arithmetic consists in $\tilde{O}(\varepsilon^{-2.5})$ matrix-vector products with each matrix $\boldsymbol{C}, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_d$ from the problem data. The algorithm succeeds with high probability.*

Theorem 6.3 contains full theoretical details. Note that the storage $\Theta(d+rn)$ is the minimum possible for any primal-dual algorithm that returns a rank-r solution to (1.5). Section 7 contains numerical evidence that the algorithm is effective for a range of examples.

1.5. Roadmap. Section 2 presents an abstract framework for studying SDPs that exposes the challenges associated with large problems. Section 3 outlines a primal-dual algorithm, called CGAL, for the model problem (1.5). Sections 4 and 5 introduce methods from randomized linear algebra that we use to control storage and arithmetic costs. Section 6 develops the SketchyCGAL algorithm, its convergence theory, and its resource usage guarantees. Section 7 contains a numerical study of SketchyCGAL, and section 8 covers related work.

1.6. Notation. The symbol $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm, while $\|\cdot\|_*$ is the nuclear norm (i.e., Schatten-1). The unadorned norm $\|\cdot\|$ refers to the ℓ_2 norm of a vector, the spectral norm of a matrix, or the operator norm of a linear map from $(\mathbb{S}_n, \|\cdot\|_{\mathrm{F}})$ to $(\mathbb{R}^d, \|\cdot\|)$. We write $\langle\cdot,\cdot\rangle$ for both the ℓ_2 inner product on vectors and the trace inner product on matrices.

The map $[\![\boldsymbol{M}]\!]_r$ returns an r-truncated singular-value decomposition of the matrix $\boldsymbol{M}$, which is a best rank-r approximation with respect to every unitarily invariant norm [69].

We use the standard computer science interpretation of the asymptotic notation $O, \tilde{O}, \Theta$.

2. Scalable semidefinite programming.

To solve the model problem (1.5) efficiently, we need to exploit structure inherent in the problem data. This section outlines an abstract approach that directs our attention to the core computational difficulties.

2.1. Abstract form of the model problem. Let us instate compact notation for the linear constraints in the model problem (1.5). Define a linear map $\mathcal{A}$ and its adjoint $\mathcal{A}^*$ via

$$\begin{aligned} \mathcal{A} &: \mathbb{S}_n \to \mathbb{R}^d, \quad \text{where} \quad \mathcal{A}\boldsymbol{X} = \big[\langle \boldsymbol{A}_1, \boldsymbol{X}\rangle \quad \dots \quad \langle \boldsymbol{A}_d, \boldsymbol{X}\rangle\big]; \\ \mathcal{A}^* &: \mathbb{R}^d \to \mathbb{S}_n, \quad \text{where} \quad \mathcal{A}^*\boldsymbol{z} = \sum\nolimits_{i=1}^{d} z_i \boldsymbol{A}_i. \end{aligned} \tag{2.1}$$

We bound the linear map with the operator norm $\|\mathcal{A}\| := \|\mathcal{A}\|_{\mathrm{F}\to\ell_2}$. Form the vector $\boldsymbol{b} := (b_1, \dots, b_d) \in \mathbb{R}^d$ of constraint values. In this notation, (1.5) becomes

$$\text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{X}\rangle \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \tag{2.2}$$

Problem instances are parameterized by the tuple $(\boldsymbol{C}, \mathcal{A}, \boldsymbol{b}, \alpha)$.

2.2. Approximate solutions. Let $\boldsymbol{X}_\star$ be a solution to the model problem (2.2). For $\varepsilon \geq 0$, we say that a matrix $\boldsymbol{X}_\varepsilon$ is *ε-optimal* for (2.2) when

$$\|\mathcal{A}\boldsymbol{X}_\varepsilon - \boldsymbol{b}\| \leq \varepsilon \quad \text{and} \quad \langle \boldsymbol{C}, \boldsymbol{X}_\varepsilon\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle \leq \varepsilon.$$

Many optimization algorithms aim to produce such ε-optimal points; cf. section 8.

As we saw in subsection 1.3.2, there are many situations where the solutions to (2.2) have low rank or they admit accurate low-rank approximations. The ε-optimal points inherit these properties for sufficiently small ε. This insight suggests a new computational goal.

For a rank parameter r, we will seek a rank-r matrix $\widehat{\boldsymbol{X}}$ that approximates an ε-optimal point $\boldsymbol{X}_\varepsilon$. More precisely, for a fixed suboptimality parameter $\zeta > 0$, we want

$$\|\boldsymbol{X}_\varepsilon - \widehat{\boldsymbol{X}}\|_* \leq (1+\zeta)\cdot\|\boldsymbol{X}_\varepsilon - [\![\boldsymbol{X}_\varepsilon]\!]_r\|_*, \quad \text{where } \operatorname{rank}\widehat{\boldsymbol{X}} \leq r \text{ and } \boldsymbol{X}_\varepsilon \text{ is } \varepsilon\text{-optimal}. \tag{2.3}$$

Given (2.3), if the ε-optimal point $\boldsymbol{X}_\varepsilon$ is close to *any* rank-r matrix, then the rank-r approximate solution $\widehat{\boldsymbol{X}}$ is also close to the ε-optimal point $\boldsymbol{X}_\varepsilon$. This formulation is advantageous because it is easier to compute and to store the low-rank matrix $\widehat{\boldsymbol{X}}$.

2.3. Black-box presentation of problem data. To develop scalable algorithms for (2.2), it is productive to hide the internal complexity of the problem instance from the algorithm [106]. To do so, we treat $\boldsymbol{C}$ and $\mathcal{A}$ as black boxes that support three primitive computations:

(2.4)

❶	❷	❸
$\boldsymbol{u} \mapsto \boldsymbol{C}\boldsymbol{u}$	$(\boldsymbol{u}, \boldsymbol{z}) \mapsto (\mathcal{A}^*\boldsymbol{z})\boldsymbol{u}$	$\boldsymbol{u} \mapsto \mathcal{A}(\boldsymbol{u}\boldsymbol{u}^*)$
$\mathbb{R}^n \to \mathbb{R}^n$	$\mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^n$	$\mathbb{R}^n \to \mathbb{R}^d$

The vectors $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^d$ are arbitrary. Although these functions may seem abstract, they are often quite natural and easy to implement well. For example, see subsection 2.4.

We will formulate algorithms for (2.2) that interact with the problem data only through the operations (2.4). We tacitly assume that the primitives require minimal storage and arithmetic; otherwise, it may be impossible to develop a truly efficient algorithm.

2.4. Example: The MaxCut SDP. To provide a concrete example, let us summarize the meaning of the primitives and the desired storage costs for solving the MaxCut SDP (1.3).

- The primitive ❶ : $\boldsymbol{u} \mapsto -\boldsymbol{L}\boldsymbol{u}$. In the typical case that the Laplacian $\boldsymbol{L}$ is sparse, this amounts to a sparse matrix-vector multiply.
- The primitive ❷ : $(\boldsymbol{u}, \boldsymbol{z}) \mapsto (\operatorname{diag}^* \boldsymbol{z})\boldsymbol{u} = (z_1 u_1, \dots, z_n u_n)$.
- The primitive ❸ : $\boldsymbol{u} \mapsto \operatorname{diag}(\boldsymbol{u}\boldsymbol{u}^*) = (|u_1|^2, \dots, |u_n|^2)$.
- The MaxCut SDP has n linear constraints, and we seek a rank-one approximation of the solution. Thus, we desire an algorithm that operates with $\Theta(n)$ working storage.

Note that we still need $\Theta(m)$ numbers to store the Laplacian $\boldsymbol{L}$ of a generic graph with m edges. But we do not charge the optimization algorithm for this storage because the algorithm only interacts with $\boldsymbol{L}$ through the primitives (2.4).

3. An algorithm for the model problem. We will develop a scalable method for the model problem (2.2) by enhancing an existing algorithm, called CGAL [102], developed by a subset of the authors. This method works well, but it is not as scalable as we would like because it lacks storage and arithmetic guarantees. This section summarizes the CGAL method and its convergence properties. Subsequent sections introduce additional ideas that we need to control resource usage, culminating with the SketchyCGAL algorithm in section 6.

3.1. The augmented problem. For a parameter $\beta > 0$, we revise the problem (2.2) by introducing an extra term in the objective:

$$\text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \frac{\beta}{2} \|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|^2 \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n. \tag{3.1}$$

The original problem (2.2) and the augmented problem (3.1) share the same optimal value and optimal set. But the new formulation has several benefits: the augmented objective cooperates with the affine constraint to penalize infeasible points, and the dual of the augmented problem is smooth, whereas the dual of the original problem is not. See [18, Chap. 2] for background.

3.2. The augmented Lagrangian. We construct the Lagrangian L_β of the augmented problem (3.1) by introducing a dual variable $\boldsymbol{y} \in \mathbb{R}^d$ and promoting the affine constraint:

$$L_\beta(\boldsymbol{X}; \boldsymbol{y}) := \langle \boldsymbol{C}, \boldsymbol{X} \rangle + \langle \boldsymbol{y}, \mathcal{A}\boldsymbol{X} - \boldsymbol{b} \rangle + \frac{\beta}{2} \|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|^2 \quad \text{for } \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n \text{ and } \boldsymbol{y} \in \mathbb{R}^d. \tag{3.2}$$

For reference, note that the partial derivatives of the augmented Lagrangian L_β satisfy

$$\partial_{\boldsymbol{X}} L_\beta(\boldsymbol{X}; \boldsymbol{y}) = \boldsymbol{C} + \mathcal{A}^*\boldsymbol{y} + \beta\mathcal{A}^*(\mathcal{A}\boldsymbol{X} - \boldsymbol{b}) \quad \text{and} \quad \partial_{\boldsymbol{y}} L_\beta(\boldsymbol{X}; \boldsymbol{y}) = \mathcal{A}\boldsymbol{X} - \boldsymbol{b}. \tag{3.3}$$

We attempt to minimize the augmented Lagrangian L_β with respect to the primal variable $\boldsymbol{X}$, while we attempt to maximize with respect to the dual variable $\boldsymbol{y}$.

Algorithm 3.1 CGAL for the model problem (2.2)

Input: Problem data for (2.2) implemented via the primitives (2.4); number T of iterations
Output: Approximate solution $\boldsymbol{X}_T$ to (2.2)

function CGAL(T)
  Scale problem data (subsection 7.1.1) ▷ **[opt]** Recommended!
  $\beta_0 \leftarrow 1$ and $K \leftarrow +\infty$ ▷ Default parameters
  $\boldsymbol{X} \leftarrow \mathbf{0}_{n\times n}$ and $\boldsymbol{y} \leftarrow \mathbf{0}_d$
  for $t \leftarrow 1, 2, 3, \ldots, T$ **do**
    $\beta \leftarrow \beta_0\sqrt{t+1}$ and $\eta \leftarrow 2/(t+1)$
    $(\xi, \boldsymbol{v}) \leftarrow \mathsf{ApproxMinEvec}(\boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y} + \beta(\mathcal{A}\boldsymbol{X} - \boldsymbol{b})); q_t)$ ▷ Algorithm 4.1 with $q_t = t^{1/4}\log n$
    ▷ Implement with primitives (2.4)❶❷!
    $\boldsymbol{X} \leftarrow (1-\eta)\,\boldsymbol{X} + \eta\,(\alpha\,\boldsymbol{v}\boldsymbol{v}^*)$
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \gamma(\mathcal{A}\boldsymbol{X} - \boldsymbol{b})$ ▷ Step size γ satisfies (3.8)

3.3. The augmented Lagrangian strategy. The form of the augmented Lagrangian suggests an algorithm. We generate a sequence $\{(\boldsymbol{X}_t; \boldsymbol{y}_t)\}$ of primal-dual pairs by alternately minimizing over the primal variable $\boldsymbol{X}$ and taking a gradient step in the dual variable $\boldsymbol{y}$.

1. **Initialize:** Choose $\boldsymbol{X}_1 \in \alpha\boldsymbol{\Delta}_n$ and $\boldsymbol{y}_1 \in \mathbb{R}^d$.
2. **Primal step:** $\boldsymbol{X}_{t+1} \in \arg\min\{L_\beta(\boldsymbol{X}; \boldsymbol{y}_t) : \boldsymbol{X} \in \alpha\boldsymbol{\Delta}_n\}$.
3. **Dual step:** $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \beta(\mathcal{A}\boldsymbol{X} - \boldsymbol{b})$.

As we proceed, we can also increase the smoothing parameter β to make violations of the affine constraint in (3.1) more and more intolerable.

The augmented Lagrangian strategy is powerful, but it is hard to apply in this setting because of the cost of implementing the primal step, even approximately.

3.4. The CGAL iteration. The CGAL iteration [102] is related to the augmented Lagrangian paradigm, but the primal steps are inspired by the conditional gradient method (CGM) [39]. CGAL identifies an update direction for the primal variable by minimizing a linear proxy for the augmented Lagrangian. We take a small primal step in the update direction, and we improve the dual variable by taking a small gradient step. At each iteration, the smoothing parameter increases according to a fixed schedule.

This subsection outlines the steps in the CGAL iteration. Afterward, we give a priori guarantees on the convergence rate. Pseudocode for CGAL appears as Algorithm 3.1.

3.4.1. Initialization. Let $\beta_0 > 0$ be an initial smoothing parameter, and fix a (large) bound $K > 0$ on the maximum allowable size of the dual variable. We also assume that we have access to the norm $\|\mathcal{A}\|$ of the constraint matrix, or—failing that—a *lower* bound. Begin with an arbitrary choice $\boldsymbol{X}_1 \in \mathbb{S}_n$ for the primal variable; set the initial dual variable $\boldsymbol{y}_1 = \mathbf{0}$.

3.4.2. Primal updates via linear minimization. At iteration $t = 1, 2, 3, \ldots$, we increase the smoothing parameter to $\beta_t := \beta_0\sqrt{t+1}$, and we form the partial derivative of the augmented Lagrangian with respect to the primal variable at the current pair of iterates:

$$\boldsymbol{D}_t := \partial_{\boldsymbol{X}} L_{\beta_t}(\boldsymbol{X}_t; \boldsymbol{y}_t) = \boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b})). \tag{3.4}$$

Then we compute an update $\boldsymbol{H}_t \in \alpha\boldsymbol{\Delta}_t$ by finding the feasible point that is most correlated with the negative gradient $-\boldsymbol{D}_t$. This amounts to a linear minimization problem:

$$\begin{aligned}\boldsymbol{H}_t &\in \arg\min\{\langle \boldsymbol{D}_t, \boldsymbol{H}\rangle : \boldsymbol{H} \in \alpha\boldsymbol{\Delta}_n\} \\ &= \operatorname{convex\,hull}\{\alpha\boldsymbol{v}\boldsymbol{v}^* : \boldsymbol{v} \text{ is a unit-norm minimum eigenvector of } \boldsymbol{D}_t\}.\end{aligned} \tag{3.5}$$

In particular, we may take $\boldsymbol{H}_t = \alpha\boldsymbol{v}_t\boldsymbol{v}_t^*$ for a minimum eigenvector $\boldsymbol{v}_t$ of the gradient $\boldsymbol{D}_t$. Next, update the matrix primal variable by taking a small step in the direction $\boldsymbol{H}_t$:

$$\boldsymbol{X}_{t+1} := (1-\eta_t)\boldsymbol{X}_t + \eta_t\boldsymbol{H}_t \in \alpha\boldsymbol{\Delta}_n, \quad \text{where} \quad \eta_t := \frac{2}{t+1}. \tag{3.6}$$

The appeal of this approach is that it only requires a single eigenvector computation (3.5). Moreover, we need not compute the eigenvector accurately; see subsection 3.4.5 for details.

3.4.3. Dual updates via gradient ascent. Next, we update the dual variable by taking a small gradient step on the dual variable in the augmented Lagrangian:

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \gamma_t\,\partial_{\boldsymbol{y}} L_{\beta_t}(\boldsymbol{X}_{t+1}; \boldsymbol{y}_t) = \boldsymbol{y}_t + \gamma_t(\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}). \tag{3.7}$$

The dual step size γ_t is the largest number that satisfies the conditions

$$\gamma_t\|\mathcal{A}\boldsymbol{X}_{t+1} - \boldsymbol{b}\|^2 \le \frac{4\alpha^2\beta_0}{(t+1)^{3/2}}\|\mathcal{A}\|^2 \quad \text{and} \quad 0 \le \gamma_t \le \beta_0. \tag{3.8}$$

We omit the dual step if it makes the dual variable too large. More precisely, if (3.7) and (3.8) result in $\|\boldsymbol{y}_{t+1}\| > K$, then we set $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t$ instead. This is the CGAL iteration.

3.4.4. Assessing solution quality. Given a primal-dual pair $(\boldsymbol{X}_t; \boldsymbol{y}_t)$, we can assess the quality of the primal solution to the model problem (2.2). First, note that we can simply compute the infeasibility: $\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}$. Second, we can bound the suboptimality of the primal objective value. To do so, define the *surrogate duality gap*

$$g_t(\boldsymbol{X}) := \max_{\boldsymbol{H}\in\alpha\boldsymbol{\Delta}_n} \langle \boldsymbol{D}_t, \boldsymbol{X} - \boldsymbol{H}\rangle = \langle \boldsymbol{C}, \boldsymbol{X}_t\rangle + \langle \boldsymbol{y}_t + \beta_t(\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}), \mathcal{A}\boldsymbol{X}_t\rangle - \lambda_{\min}(\boldsymbol{D}_t). \tag{3.9}$$

The latter expression follows from (3.4) and (3.5). As usual, $\lambda_{\min}$ is the minimum eigenvalue. The suboptimality of $\boldsymbol{X}_t$ is bounded as

$$\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle \le g_t(\boldsymbol{X}_t) - \langle \boldsymbol{y}_t, \mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\rangle - \frac{1}{2}\beta_t\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\|^2, \tag{3.10}$$

where $\boldsymbol{X}_\star$ is a primal optimal point. See subsection SM1.7 for the derivation of (3.10).

For CGAL, the surrogate duality gap (3.9) is analogous to the Frank–Wolfe duality gap [50]. It offers a practical tool for detecting early convergence. Unfortunately, we have not been able to prove that the CGAL error bound (3.10) converges to zero as the algorithm converges.

3.4.5. Approximate eigenvector computations. A crucial fact is that the CGAL strategy provably works if we replace (3.5) with an approximate minimum eigenvector computation. At step t, suppose that we find an update direction

$$\boldsymbol{H}_t := \alpha\boldsymbol{v}_t\boldsymbol{v}_t^*, \quad \text{where} \quad \boldsymbol{v}_t^*\boldsymbol{D}_t\boldsymbol{v}_t \le \lambda_{\min}(\boldsymbol{D}_t) + \frac{1}{\sqrt{t+1}}\|\boldsymbol{D}_t\|, \tag{3.11}$$

for a unit vector $\boldsymbol{v}_t \in \mathbb{F}^n$. The matrix $\boldsymbol{H}_t$ defined in (3.11) serves in place of a solution to (3.5) throughout the algorithm. We discuss solvers for the subproblem (3.11) in section 4.

3.5. Convergence guarantees for CGAL. The following result describes the convergence of the CGAL iteration. The analysis is adapted from [102, Thm. 3.1]; see section SM1 for details and a recapitulation of the proof.

Fact 3.1 (CGAL: Convergence). *Assume problem (2.2) satisfies strong duality. The CGAL iteration (subsection 3.4) with approximate eigenvector computations (3.11) yields a sequence* $\{\boldsymbol{X}_t : t = 1, 2, 3, \dots\} \subset \alpha\boldsymbol{\Delta}_n$ *that satisfies*

$$\|\mathcal{A}\boldsymbol{X}_t - \boldsymbol{b}\| \leq \frac{\text{Const}}{\sqrt{t}} \quad \text{and} \quad |\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle| \leq \frac{\text{Const}}{\sqrt{t}}. \tag{3.12}$$

The matrix $\boldsymbol{X}_\star$ *solves (2.2). The constant depends on the problem data* $(\boldsymbol{C}, \mathcal{A}, \boldsymbol{b}, \alpha)$*, the minimum Euclidean norm of a dual solution, and the algorithm parameters* β_0 *and* K*.*

In view of (3.12), CGAL produces a primal iterate $\boldsymbol{X}_T$ that is ε-optimal after $T = O(\varepsilon^{-2})$ iterations. The numerical work in [102, sect. 5] shows that CGAL finds an ε-optimal point after $T = O(\varepsilon^{-1})$ iterations for realistic problem instances. See also subsection SM5.3.

The analysis behind Fact 3.1 indicates that CGAL converges more quickly if we precondition the problem data by rescaling; see subsection 7.1.1. This step is critical in practice.

4. Approximate eigenvectors. Most of the computation in CGAL occurs in the approximate eigenvector step (3.11). This section describes our approach to this subproblem.

4.1. Krylov methods. The approximate minimum eigenvector computation (3.11) involves a matrix of the form $\boldsymbol{D}_t = \boldsymbol{C} + \mathcal{A}^*\boldsymbol{w}_t$. We can multiply the matrix $\boldsymbol{D}_t$ by a vector using the primitives (2.4)❶❷. Krylov methods compute eigenvectors of $\boldsymbol{D}_t$ by repeated matrix-vector multiplication with $\boldsymbol{D}_t$, so they are obvious tools for the subproblem (3.11).

Unfortunately, CGAL tends to generate matrices $\boldsymbol{D}_t$ that have clustered eigenvalues. These matrices challenge standard eigenvector software, such as ARPACK [62], the engine behind the MATLAB command `eigs`. Instead, we retreat to a more classical technique.

4.2. A storage-optimal randomized Lanczos method. We use a nonstandard implementation of the randomized Lanczos method [57] to find an approximate eigenvector with minimal storage. This hinges on two ideas. First, we run the Lanczos iteration with a random initial vector, which supports convergence guarantees. Second, we do not store the Lanczos vectors but rather regenerate them to construct the approximate eigenvector. See Algorithm 4.1 for pseudocode. Kuczyński & Woźniakowski [57, Thm. 4.2(a)] have obtained error bounds.

Fact 4.1 (randomized Lanczos method). *Let* $\boldsymbol{M} \in \mathbb{S}_n$*. For* $\varepsilon \in (0, 1]$ *and* $\delta \in (0, 0.5]$*, the randomized Lanczos method, Algorithm 4.1, computes a unit vector* $\boldsymbol{u} \in \mathbb{F}^n$ *that satisfies*

$$\boldsymbol{u}^*\boldsymbol{M}\boldsymbol{u} \leq \lambda_{\min}(\boldsymbol{M}) + \frac{\varepsilon}{8}\|\boldsymbol{M}\| \quad \textit{with probability at least } 1 - 2\delta$$

after $q \geq \frac{1}{2} + \varepsilon^{-1/2}\log(n/\delta^2)$ *iterations. The arithmetic cost is at most* q *matrix-vector multiplies with* $\boldsymbol{M}$ *and* $O(qn)$ *extra operations. The working storage is* $O(n + q)$*.*

With constant probability, we solve the eigenvector problem (3.11) successfully in every iteration t of CGAL if we use $q_t = O(t^{1/4}\log(tn))$ iterations of Algorithm 4.1. In practice, we implement CGAL with $q_t = t^{1/4}\log n$. Although the Lanczos method has complicated numerical behavior in finite-precision arithmetic, it serves well as a subroutine within CGAL.

Algorithm 4.1 ApproxMinEvec via storage-optimal randomized Lanczos (subsection 4.2)

Input: Input matrix $\boldsymbol{M} \in \mathbb{S}_n$ and maximum number q of iterations
Output: Approximate minimum eigenpair $(\xi, \boldsymbol{v}) \in \mathbb{R} \times \mathbb{F}^n$ of the matrix $\boldsymbol{M}$

1 **function** ApproxMinEvec($\boldsymbol{M}$; q)
2 $\quad \boldsymbol{v}_1 \leftarrow \texttt{randn}(n, 1)$
3 $\quad \boldsymbol{v}_1 \leftarrow \boldsymbol{v}_1 / \|\boldsymbol{v}_1\|$ ▷ Store $\boldsymbol{v}_1$ for reuse
4 $\quad$ **for** $i \leftarrow 1, 2, 3, \ldots, \min\{q, n-1\}$ **do** ▷ During loop, store only $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i+1}$
5 $\quad\quad \omega_i \leftarrow \mathrm{Re}(\boldsymbol{v}_i^*(\boldsymbol{M}\boldsymbol{v}_i))$
6 $\quad\quad \boldsymbol{v}_{i+1} \leftarrow \boldsymbol{M}\boldsymbol{v}_i - \omega_i \boldsymbol{v}_i - \rho_{i-1}\boldsymbol{v}_{i-1}$ ▷ Three-term Lanczos recurrence; $\rho_0 \boldsymbol{v}_0 = \boldsymbol{0}$
7 $\quad\quad \rho_i \leftarrow \|\boldsymbol{v}_i\|$
8 $\quad\quad$ **if** $\rho_i = 0$ **then break** ▷ Found an invariant subspace!
9 $\quad\quad \boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1} / \rho_i$
10 $\quad \boldsymbol{T} \leftarrow \texttt{tridiag}(\boldsymbol{\rho}_{1:(i-1)}, \boldsymbol{\omega}_{1:i}, \boldsymbol{\rho}_{1:(i-1)})$ ▷ Form tridiagonal matrix
11 $\quad [\xi, \boldsymbol{u}] \leftarrow \texttt{MinEvec}(\boldsymbol{T})$ ▷ Exploit tridiagonal form [77, Chap. 7], or just use `eig`
12 $\quad \boldsymbol{v} \leftarrow \sum_{j=1}^{i} u_j \boldsymbol{v}_j$ ▷ Modify lines 4–9 to regenerate $\boldsymbol{v}_2, \ldots, \boldsymbol{v}_i$ and form sum

Remark 4.2 (time-storage trade-off). We can retain the vectors $\boldsymbol{v}_i$ in Algorithm 4.1 to reduce the flop count by approximately a factor of two, but the storage increases to $O(qn)$.

5. Sketching and reconstruction of a psd matrix.

The CGAL iteration generates a psd matrix that solves the model problem (2.2) via a sequence (3.6) of rank-one linear updates. To control storage costs, SketchyCGAL retains only a compressed version of the psd matrix variable $\boldsymbol{X}_t$. This section outlines the *Nyström sketch*, an established method [45, 43, 64, 93] that can track the evolving psd matrix and then report a provably accurate low-rank approximation.

For more information about the implementation and behavior of low-rank matrix approximation from streaming data, see section SM2 and our papers [93, 94, 95].

5.1. Sketching and updates.

Consider a psd input matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let R be a parameter that modulates the storage cost of the sketch and the quality of the matrix approximation.

To construct the Nyström sketch, we draw and fix a standard normal[1] test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n \times R}$. Our summary, or *sketch*, of the matrix $\boldsymbol{X}$ takes the form

$$\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega} \in \mathbb{F}^{n \times R}. \tag{5.1}$$

The sketch $\boldsymbol{S}$ supports linear rank-one updates to $\boldsymbol{X}$. Indeed, we can track the evolution

$$\begin{aligned} &\boldsymbol{X} \leftarrow (1-\eta)\,\boldsymbol{X} + \eta\,\boldsymbol{v}\boldsymbol{v}^* \qquad \text{for } \eta \in [0,1] \text{ and } \boldsymbol{v} \in \mathbb{F}^n \\ &\text{via} \quad \boldsymbol{S} \leftarrow (1-\eta)\,\boldsymbol{S} + \eta\,\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{\Omega}). \end{aligned} \tag{5.2}$$

The test matrix $\boldsymbol{\Omega}$ and the sketch $\boldsymbol{S}$ require storage of $2Rn$ numbers in $\mathbb{F}$. The arithmetic cost of the linear update (5.2) to the sketch is $\Theta(Rn)$ numerical operations.

Remark 5.1 (structured random matrices). We can reduce storage costs by a factor of two by using a structured random matrix in place of $\boldsymbol{\Omega}$. For example, see [95, sect. 3] or [91].

[1] Each entry of the matrix is an independent Gaussian random variable with mean zero and variance one. In the complex setting, the real and imaginary parts of each entry are independent standard normal variables.

Algorithm 5.1 NystromSketch implementation (see section 5)

Input: Dimension n of input matrix, size R of sketch
Output: Rank-R approximation $\widehat{\boldsymbol{X}}$ of sketched matrix in factored form $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U} \in \mathbb{F}^{n\times R}$ has orthonormal columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{R\times R}$ is nonnegative diagonal

function NystromSketch.Init(n, R)
  $\boldsymbol{\Omega} \leftarrow \texttt{randn}(n, R)$ ▷ Draw and fix random test matrix
  $\boldsymbol{S} \leftarrow \texttt{zeros}(n, R)$ ▷ Form sketch of zero matrix

function NystromSketch.RankOneUpdate($\boldsymbol{v}$, η) ▷ Implements (5.2)
  $\boldsymbol{S} \leftarrow (1-\eta)\,\boldsymbol{S} + \eta\,\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{\Omega})$ ▷ Update sketch of matrix

function NystromSketch.Reconstruct()
  $\sigma \leftarrow \sqrt{n}\,\texttt{eps}(\texttt{norm}(\boldsymbol{S}))$ ▷ Compute a shift parameter
  $\boldsymbol{S}_\sigma \leftarrow \boldsymbol{S} + \sigma\,\boldsymbol{\Omega}$ ▷ Implicitly form sketch of $\boldsymbol{X} + \sigma\,\mathbf{I}$
  $\boldsymbol{L} \leftarrow \texttt{chol}(\boldsymbol{\Omega}^*\boldsymbol{S}_\sigma)$
  $[\boldsymbol{U}, \boldsymbol{\Sigma}, \sim] \leftarrow \texttt{svd}(\boldsymbol{S}_\sigma/\boldsymbol{L})$ ▷ Dense SVD
  $\boldsymbol{\Lambda} \leftarrow \max\{0, \boldsymbol{\Sigma}^2 - \sigma\,\mathbf{I}\}$ ▷ Remove shift

5.2. The reconstruction process. Given the test matrix $\boldsymbol{\Omega}$ and the sketch $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$, we form a rank-$R$ approximation $\widehat{\boldsymbol{X}}$ of the sketched matrix $\boldsymbol{X}$. The approximation is defined by

$$\widehat{\boldsymbol{X}} := \boldsymbol{S}(\boldsymbol{\Omega}^*\boldsymbol{S})^\dagger\boldsymbol{S}^* = (\boldsymbol{X}\boldsymbol{\Omega})(\boldsymbol{\Omega}^*\boldsymbol{X}\boldsymbol{\Omega})^\dagger(\boldsymbol{X}\boldsymbol{\Omega})^*, \tag{5.3}$$

where † is the pseudoinverse. This reconstruction is called a *Nyström approximation*. We often truncate $\widehat{\boldsymbol{X}}$ by replacing it with its best rank-r approximation $[\![\widehat{\boldsymbol{X}}]\!]_r$ for some $r \le R$.

See Algorithm 5.1 for a numerically stable implementation of the Nyström approximation (5.3). The algorithm takes $\Theta(R^2 n)$ numerical operations and $\Theta(Rn)$ storage.

5.3. A priori error bounds. The Nyström approximation $\widehat{\boldsymbol{X}}$ yields a provably good estimate for the matrix $\boldsymbol{X}$ contained in the sketch [93, Thm. 4.1].

Fact 5.2 (Nyström sketch: Error bound). *Fix a psd matrix $\boldsymbol{X} \in \mathbb{S}_n$. Let $\boldsymbol{S} = \boldsymbol{X}\boldsymbol{\Omega}$ where $\boldsymbol{\Omega} \in \mathbb{F}^{n\times R}$ is standard normal. For each $r < R-1$, the Nyström approximation (5.3) satisfies*

$$\mathbb{E}_{\boldsymbol{\Omega}}\,\|\boldsymbol{X} - \widehat{\boldsymbol{X}}\|_* \le \left(1 + \frac{r}{R-r-1}\right)\cdot\|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*, \tag{5.4}$$

where $\mathbb{E}_{\boldsymbol{\Omega}}$ is the expectation with respect to $\boldsymbol{\Omega}$. If we replace $\widehat{\boldsymbol{X}}$ with its rank-r truncation $[\![\widehat{\boldsymbol{X}}]\!]_r$, the error bound (5.4) remains valid. Similar results hold with high probability.

6. Scalable semidefinite programming via SketchyCGAL. We may now present an extension of CGAL that solves the model problem (2.2) with controlled storage and computation. Our new algorithm, SketchyCGAL, enhances the CGAL iteration from subsection 3.4. Instead of storing the matrix $\boldsymbol{X}_t$ in the CGAL iteration,

1. we drive the iteration with the d-dimensional primal state variable $\boldsymbol{z}_t := \mathcal{A}\boldsymbol{X}_t$;
2. we maintain a Nyström sketch of the primal iterate $\boldsymbol{X}_t$ using storage $\Theta(Rn)$;
3. when the iteration is halted, say, at time T, we use the sketch to construct a rank-R approximation $\widehat{\boldsymbol{X}}_T$ of the implicitly computed solution $\boldsymbol{X}_T$ of the model problem.

As we will see, the resulting method exhibits almost the same convergence behavior as the CGAL algorithm, but it also enjoys a strong storage guarantee (when $d \ll n^2$ and $R \ll n$).

6.1. The SketchyCGAL iteration. To develop the SketchyCGAL iteration, we start with the CGAL iteration. Then we make the substitutions $\boldsymbol{z}_t = \mathcal{A}\boldsymbol{X}_t$ and $\boldsymbol{h}_t = \mathcal{A}\boldsymbol{H}_t$ to eliminate the matrix variables. Let us summarize what happens; see subsection 6.2 for additional explanation. Algorithm 6.1 contains pseudocode with implementation recommendations.

6.1.1. Initialization. First, we choose the size R of the Nyström sketch. Then we draw and fix the random test matrix $\boldsymbol{\Omega} \in \mathbb{F}^{n\times R}$. Select an initial smoothing parameter β_0 and a (finite) bound K on the dual variable. Define the smoothing parameters $\beta_t := \beta_0\sqrt{t+1}$ and the step size parameters $\eta_t := 2/(t+1)$. Initialize the primal state variable $\boldsymbol{z}_1 := \mathbf{0} \in \mathbb{R}^d$, the sketch $\boldsymbol{S}_1 := \mathbf{0} \in \mathbb{F}^{n\times R}$, and the dual variable $\boldsymbol{y}_1 := \mathbf{0} \in \mathbb{R}^d$. These choices correspond to the simplest initial iterate $\boldsymbol{X}_1 := \mathbf{0}$.

6.1.2. Primal updates. At iteration $t = 1, 2, 3, \dots$, we compute a unit-norm vector $\boldsymbol{v}_t$ that is an approximate minimum eigenvector of the gradient $\boldsymbol{D}_t$ of the smoothed objective:

$$\xi_t := \boldsymbol{v}_t^*\boldsymbol{D}_t\boldsymbol{v}_t \le \lambda_{\min}(\boldsymbol{D}_t) + \frac{\|\boldsymbol{D}_t\|}{\sqrt{t+1}}, \quad \text{where} \quad \boldsymbol{D}_t := \boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y}_t + \beta_t(\boldsymbol{z}_t - \boldsymbol{b})). \tag{6.1}$$

This calculation corresponds with (3.11). Form the primal update direction $\boldsymbol{h}_t = \mathcal{A}(\alpha\boldsymbol{v}_t\boldsymbol{v}_t^*)$, and then update the primal state variable $\boldsymbol{z}_t$ and the sketch $\boldsymbol{S}_t$:

$$\boldsymbol{z}_{t+1} := (1-\eta_t)\boldsymbol{z}_t + \eta_t\boldsymbol{h}_t \quad \text{and} \quad \boldsymbol{S}_{t+1} := (1-\eta_t)\boldsymbol{S}_t + \eta_t\alpha\boldsymbol{v}_t(\boldsymbol{v}_t^*\boldsymbol{\Omega}). \tag{6.2}$$

We obtain the update rule for the primal state variable $\boldsymbol{z}_t$ by applying the linear map $\mathcal{A}$ to the primal update rule (3.6). The update rule for the sketch $\boldsymbol{S}_t$ follows from (5.2).

6.1.3. Dual updates. The update to the dual variable takes the form

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \gamma_t(\boldsymbol{z}_{t+1} - \boldsymbol{b}), \tag{6.3}$$

where we choose the largest γ_t that satisfies the conditions

$$\gamma_t\|\boldsymbol{z}_{t+1} - \boldsymbol{b}\|^2 \le \frac{4\alpha^2\beta_0}{(t+1)^{3/2}}\|\mathcal{A}\|^2 \quad \text{and} \quad 0 \le \gamma_t \le \beta_0. \tag{6.4}$$

If needed, we set $\gamma_t = 0$ to prevent $\|\boldsymbol{y}_{t+1}\| > K$. This is the SketchyCGAL iteration.

6.2. Connection with CGAL. There is a tight connection between the iterates of SketchyCGAL and CGAL. Let $\boldsymbol{X}_1 := \mathbf{0}$. Using the vectors $\boldsymbol{v}_t$ computed in (6.1), define matrices

$$\boldsymbol{H}_t := \alpha\boldsymbol{v}_t\boldsymbol{v}_t^* \quad \text{and} \quad \boldsymbol{X}_{t+1} := (1-\eta_t)\boldsymbol{X}_t + \eta_t\boldsymbol{H}_t. \tag{6.5}$$

With these definitions, the following loop invariants are in force:

$$\boldsymbol{h}_t = \mathcal{A}\boldsymbol{H}_t \quad \text{and} \quad \boldsymbol{z}_t = \mathcal{A}\boldsymbol{X}_t \quad \text{and} \quad \boldsymbol{S}_t = \boldsymbol{X}_t\boldsymbol{\Omega}. \tag{6.6}$$

By comparing subsections 3.4 and 6.1, we see that the trajectory $\{(\boldsymbol{X}_t, \boldsymbol{H}_t, \boldsymbol{y}_t) : t = 1, 2, 3, \dots\}$ could also have been generated by running the CGAL iteration. In other words, the variables in SketchyCGAL track the variables of some invocation of CGAL and inherit their behavior. We refer to the matrices $\boldsymbol{X}_t$ as the *implicit* CGAL iterates.

Algorithm 6.1 SketchyCGAL for the model problem (2.2)

Input: Problem data for (2.2) implemented via the primitives (2.4), sketch size R, number T of iterations
Output: Rank-R approximate solution to (2.2) in factored form $\widehat{\boldsymbol{X}}_T = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U} \in \mathbb{F}^{n\times R}$ has orthonormal columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{R\times R}$ is nonnegative diagonal

1 **function** SketchyCGAL(R; T)
2 Scale problem data (subsection 7.1.1) ▷ **[opt]** Recommended!
3 $\beta_0 \leftarrow 1$ and $K \leftarrow +\infty$ ▷ Default parameters
4 NystromSketch.Init(n, R)
5 $\boldsymbol{z} \leftarrow \mathbf{0}_d$ and $\boldsymbol{y} \leftarrow \mathbf{0}_d$
6 **for** $t \leftarrow 1, 2, 3, \ldots, T$ **do**
7 $\beta \leftarrow \beta_0\sqrt{t+1}$ and $\eta \leftarrow 2/(t+1)$
8 $[\xi, \boldsymbol{v}] \leftarrow$ ApproxMinEvec($\boldsymbol{C} + \mathcal{A}^*(\boldsymbol{y} + \beta(\boldsymbol{z} - \boldsymbol{b}))$; q_t) ▷ Algorithm 4.1 with $q_t = t^{1/4}\log n$
▷ Implement with primitives (2.4)❶❷!
9 $\boldsymbol{z} \leftarrow (1-\eta)\,\boldsymbol{z} + \eta\,\mathcal{A}(\alpha\boldsymbol{v}\boldsymbol{v}^*)$ ▷ Use primitive (2.4)❸
10 $\boldsymbol{y} \leftarrow \boldsymbol{y} + \gamma(\boldsymbol{z} - \boldsymbol{b})$ ▷ γ is the largest solution to (6.4)
11 NystromSketch.RankOneUpdate($\sqrt{\alpha}\boldsymbol{v}$, η)
12 $[\boldsymbol{U}, \boldsymbol{\Lambda}] \leftarrow$ NystromSketch.Reconstruct()
13 $\boldsymbol{\Lambda} \leftarrow \boldsymbol{\Lambda} + (\alpha - \mathrm{tr}(\boldsymbol{\Lambda}))\mathbf{I}_R/R$ ▷ **[opt]** Enforce trace constraint in (2.2)

6.2.1. Approximating the CGAL iterates. We do not have access to the implicit CGAL iterates described above. Nevertheless, the sketch permits us to approximate them! After iteration t of SketchyCGAL, we can form a rank-R approximation $\widehat{\boldsymbol{X}}_t$ of the implicit iterate $\boldsymbol{X}_t$ by invoking the formula (5.3) with $\boldsymbol{S} = \boldsymbol{S}_t$. According to Fact 5.2, for each $r < R - 1$,

$$\mathbb{E}_{\boldsymbol{\Omega}}\,\|\boldsymbol{X}_t - \widehat{\boldsymbol{X}}_t\|_* \leq \left(1 + \frac{r}{R-r-1}\right) \cdot \|\boldsymbol{X}_t - [\![\boldsymbol{X}_t]\!]_r\|_*. \tag{6.7}$$

In other words, the computed approximation $\widehat{\boldsymbol{X}}_t$ is a good proxy for the implicit iterate $\boldsymbol{X}_t$ whenever the latter matrix is well-approximated by a low-rank matrix. The same bound (6.7) holds if we replace $\widehat{\boldsymbol{X}}_t$ by the truncated rank-r matrix $[\![\widehat{\boldsymbol{X}}_t]\!]_r$.

Remark 6.1 (trace correction). The model problem (2.2) requires the matrix variable to have trace α, but the computed solution $\widehat{\boldsymbol{X}}_t$ rarely satisfies this constraint. Algorithm 6.1 includes an optional projection step (line 13) that corrects the trace. This step never increases the error in the Nyström approximation by more than a factor of two (subsection SM2.4). Our analysis of SketchyCGAL *does not* include the projection, but it is valuable in practice.

6.2.2. Assessing solution quality. Given quantities computed by SketchyCGAL, we can assess how well the implicit iterate $\boldsymbol{X}_t$ solves the model problem (2.2). The infeasibility is just $\boldsymbol{z}_t - \boldsymbol{b}$. To bound suboptimality, we track the objective $p_t := \langle \boldsymbol{C}, \boldsymbol{X}_t\rangle$ by setting $p_1 = 0$ and using the update $p_{t+1} = (1-\eta_t)p_t + \eta_t\alpha\boldsymbol{v}_t^*(\boldsymbol{C}\boldsymbol{v}_t)$. The surrogate duality gap (3.9) takes the form

$$g_t(\boldsymbol{X}_t) = p_t + \langle \boldsymbol{y}_t + \beta_t(\boldsymbol{z}_t - \boldsymbol{b}),\, \boldsymbol{z}_t\rangle - \lambda_{\min}(\boldsymbol{D}_t).$$

In practice, we use (6.1) to approximate the minimum eigenvalue: $\lambda_{\min}(\boldsymbol{D}_t) \approx \xi_t$. In light of (3.10), we can bound the suboptimality of $\boldsymbol{X}_t$ via the expression

$$\langle \boldsymbol{C}, \boldsymbol{X}_t\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle \leq g_t(\boldsymbol{X}_t) - \langle \boldsymbol{y}_t, \boldsymbol{z}_t - \boldsymbol{b}\rangle - \frac{1}{2}\beta_t\|\boldsymbol{z}_t - \boldsymbol{b}\|^2, \tag{6.8}$$

where $\boldsymbol{X}_\star$ is a primal optimal point. See subsection SM3.1 for more details.

6.3. Convergence of SketchyCGAL. The implicit iterates $\boldsymbol{X}_t$ converge to a solution of the model problem (2.2) at the same rate as the iterates of CGAL would. On average, the rank-R iterates $\widehat{\boldsymbol{X}}_t$ track the implicit iterates. The discrepancy between them depends on how well the implicit iterates are approximated by low-rank matrices. Here is a simple convergence result that reflects this intuition; see subsection SM3.2 for the easy proof and further results.

Theorem 6.2 (SketchyCGAL: Convergence). *Assume problem (2.2) satisfies strong duality, and let $\Psi_\star$ be its solution set. For each $r < R - 1$, the iterates $\widehat{\boldsymbol{X}}_t$ computed by SketchyCGAL (subsections 6.1 and 6.2) satisfy*

$$\limsup_{t\to\infty} \mathbb{E}_{\boldsymbol{\Omega}} \operatorname{dist}_*(\widehat{\boldsymbol{X}}_t, \Psi_\star) \leq \left(1 + \frac{r}{R - r - 1}\right) \cdot \max_{\boldsymbol{X}\in\Psi_\star} \|\boldsymbol{X} - [\![\boldsymbol{X}]\!]_r\|_*.$$

The same bound holds if we replace $\widehat{\boldsymbol{X}}_t$ with the truncated approximation $[\![\widehat{\boldsymbol{X}}_t]\!]_r$. Here, dist_* is the nuclear-norm distance between a matrix and a set of matrices.

6.4. Resource usage. The main working variables in the SketchyCGAL iteration are the primal state $\boldsymbol{z}_t \in \mathbb{R}^d$, the computed eigenvector $\boldsymbol{v}_t \in \mathbb{R}^n$, the update direction $\boldsymbol{h}_t \in \mathbb{R}^d$, the test matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n\times R}$, the sketch $\boldsymbol{S}_t \in \mathbb{R}^{n\times R}$, and the dual variable $\boldsymbol{y}_t \in \mathbb{R}^d$. The total cost for storing these variables is $\Theta(d + Rn)$.

The arithmetic bottleneck in SketchyCGAL comes from the approximate eigenvector computation (6.1); see subsection 4.2 for resource requirements. The remaining computation takes place in the variable updates. To form the primal update direction $\boldsymbol{h}_t$, we invoke the primitive (2.4)❸. To update the primal state variable $\boldsymbol{z}_t$ and the sketch $\boldsymbol{S}_t$ in (6.2), we need $O(d + Rn)$ arithmetic operations. No further storage is required at this stage.

Table 1 documents the cost of performing T iterations of the SketchyCGAL method. The first column summarizes the resources consumed in the outer iteration. The second column tabulates the total resources spent to solve the eigenvalue problem (6.1) via the randomized Lanczos method (Algorithm 4.1).

In light of Fact 3.1 and subsection 6.2, we can be confident that the implicit iterate $\boldsymbol{X}_T$ is ε-optimal within $T = O(\varepsilon^{-2})$ iterations. The formula (6.7) gives a priori guarantees on the quality of the approximation $\widehat{\boldsymbol{X}}_T$.

Table 1

Resource usage for T iterations of SketchyCGAL with sketch size R. The algorithm returns a rank-R approximation to an ε-optimal solution (subsection 2.2) to the model problem (2.2). In theory, $T \sim \varepsilon^{-2}$. In practice, $T \sim \varepsilon^{-1}$. Constants and logarithms are omitted. See subsection 6.5.

	SketchyCGAL Algorithm 6.1	With Lanczos Algorithm 4.1
Storage (floats)	$d + Rn$	n
Arithmetic (flops)	$T(d + Rn)$	$T^{5/4}n$
Primitives (calls)		
(2.4)❶❷	—	$T^{5/4}$
(2.4)❸	T	—

6.5. Theoretical performance of SketchyCGAL. We can package up this discussion in a theorem that describes the theoretical performance of the SketchyCGAL method.

Theorem 6.3 (SketchyCGAL). *Assume the model problem (2.2) satisfies strong duality. Fix a rank r, and set the sketch size $R \geq \zeta^{-1}r + 1$. After $T = O(\varepsilon^{-2})$ iterations, SketchyCGAL (subsections 6.1 and 6.2) returns a rank-r approximation $\widehat{\boldsymbol{X}} = [\![\widehat{\boldsymbol{X}}_T]\!]_r$ to an ε-optimal point of (2.2) that satisfies (2.3) with constant probability.*

Suppose we use the storage-optimal Lanczos method (Algorithm 4.1) to solve (6.1). Then the total storage cost for SketchyCGAL is $O(d + Rn)$ floats. The arithmetic requirements are $O(\varepsilon^{-2}(d + Rn) + \varepsilon^{-5/2}n\log(n/\varepsilon))$ flops, $O(\varepsilon^{-5/2}\log(n/\varepsilon))$ invocations of the primitives (2.4)❶❷, and $O(\varepsilon^{-2})$ applications of the primitive (2.4)❸.

The constants depend on the problem data $(\boldsymbol{C}, \mathcal{A}, \boldsymbol{b}, \alpha)$ and algorithm parameters (β_0, K).

In practice, SketchyCGAL performs better than Theorem 6.3 suggests: empirically, we find that $T = O(\varepsilon^{-1})$ for real-world problem instances.

7. Numerical examples. This section showcases computational experiments that establish that SketchyCGAL is a practical method for solving large SDPs. We show that the algorithm is flexible by applying it to several classes of SDPs, and we show it is reliable by solving a large number of instances of each type. We give empirical evidence that the (implicit) iterates converge to optimality much faster than Theorem 6.3 suggests. Comparisons with other general-purpose SDP solvers demonstrate that SketchyCGAL is competitive for small SDPs, while it scales to problems that standard methods cannot handle.

7.1. Setup. All experiments are performed in MATLAB_R2018a with double-precision arithmetic. Source code is included with the supplementary material, linked from the main article webpage. To simulate the processing power of a personal laptop computer, we use a single Intel Xeon CPU E5-2630 v3, clocked at 2.40 GHz, with RAM usage capped at 16 GB.

Arithmetic costs are measured in terms of actual runtime. MATLAB does not currently offer a memory profiler, so we externally monitor the total memory allocated. We approximate the storage cost by reporting the peak value minus the storage at a MATLAB idle state.

7.1.1. Problem scaling. The bounds in the convergence theorem Fact 3.1 for the implicit iterates of SketchyCGAL depend on problem scaling. Our analysis motivates us to set

$$\|\boldsymbol{C}\|_{\mathrm{F}} = \|\mathcal{A}\| = \alpha = 1 \quad \text{and} \quad \|\boldsymbol{A}_1\|_{\mathrm{F}} = \|\boldsymbol{A}_2\|_{\mathrm{F}} = \cdots = \|\boldsymbol{A}_d\|_{\mathrm{F}}. \tag{7.1}$$

In our experiments, we enforce the scalings (7.1), except where noted.

7.1.2. Implementation. Our experiments require a variant of SketchyCGAL that can handle inequality constraints; see section SM4. We implement the algorithm with the default parameters ($\beta_0 = 1$ and $K = +\infty$). The sketch uses a Gaussian test matrix. The eigenvalue subproblem is solved via randomized Lanczos (Algorithm 4.1). We include the optional trace normalization (line 13 in Algorithm 6.1) whenever appropriate. *No other tuning is done.*

7.2. The MaxCut SDP. We begin with MaxCut SDPs (1.3). Our goal is to assess the storage and arithmetic costs of SketchyCGAL for a standard testbed. We compare with provable solvers for general SDPs: SEDUMI [90], SDPT3 [92], MOSEK [70], and SDPNAL+ [100].

7.2.1. Rounding. To extract a cut from an approximate solution to (1.3), we apply a simple rounding procedure. SketchyCGAL returns a matrix $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$, where $\boldsymbol{U} \in \mathbb{R}^{n\times R}$ has orthonormal columns. The columns of the entrywise signum, $\operatorname{sgn}(\boldsymbol{U})$, are the signed indicators of R cuts. We compute the weights of all R cuts and select the largest. The other solvers return a full-dimensional solution $\widehat{\boldsymbol{X}}$; we compute the top R eigenvectors of $\widehat{\boldsymbol{X}}$ using the MATLAB command `eigs` and invoke the same rounding procedure.

7.2.2. Datasets. We consider datasets from two different benchmark groups:

1. GSET: 67 binary-valued matrices generated by an autonomous random graph generator and published online [101]. The dimension n varies from 800 to 10,000.
2. DIMACS10: This benchmark consists of 150 symmetric matrices (with n varying from 39 to 50,912,018) chosen for the 10th DIMACS Implementation Challenge [12]. We consider 148 datasets with dimension $n \leq 24,000,000$. Two problems (`rgg_n_2_24_s0` and `Europe_osm`) exceeded the 16 GB storage limit.

7.2.3. Storage and arithmetic comparisons. We run each solver for each dataset and measure the storage cost and the runtime. We invoke SketchyCGAL with rank parameter $R = 10$, and we stop the algorithm when the error bound (6.8) guarantees that the implicit iterates have both relative suboptimality and infeasibility below 10^{-1}. For other solvers, we set the relative error tolerance to 10^{-1}. See section SM5 for implementation details.

Figure 1.1 displays the results of this experiment. The conventional convex solvers do not scale beyond $n = 10^4$ due to their high storage costs. In contrast, SketchyCGAL handles problems with $n \approx 2.2 \cdot 10^7$. For nine datasets, SketchyCGAL did not reach the error tolerance within five days. Further details and results appear in subsection SM5.2.

7.2.4. Empirical convergence rates. Next, we investigate the empirical convergence of SketchyCGAL and the effect of the sketch size parameter R. We consider the MaxCut SDP (1.3) for the G67 dataset ($n = 10\,000$), the largest instance in GSET. We run 10^6 iterations of SketchyCGAL for each $R \in \{10, 25, 100\}$.

We use a high-accuracy solution from SDPT3 to approximate an optimal point $\boldsymbol{X}_\star$ of (2.2). Given a prospective solution $\boldsymbol{X}$, we compute its relative suboptimality and feasibility as

$$\texttt{objective residual} = \frac{|\langle \boldsymbol{C}, \boldsymbol{X}\rangle - \langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle|}{1 + |\langle \boldsymbol{C}, \boldsymbol{X}_\star\rangle|} \quad \text{and} \quad \texttt{infeasibility} = \frac{\|\mathcal{A}\boldsymbol{X} - \boldsymbol{b}\|}{1 + \|\boldsymbol{b}\|}.$$

It is standard to increment the denominator by one to handle small values gracefully. These quantities are evaluated with respect to the original (not rescaled) problem data.

Figure 7.1 illustrates the convergence of SketchyCGAL for this problem. After t iterations, the residual and infeasibility decay like $O(t^{-1})$, which is far better than the $O(t^{-1/2})$ bound in Theorem 6.3. Similar behavior is manifest for other datasets and other problems.

Figure 7.1 also displays the weight of the cut obtained after rounding, compared with the weight of the cut obtained from the SDPT3 solution. Observe that sketch size $R = 10$ is sufficient, and the SketchyCGAL solutions yield excellent cuts after a few hundred iterations.

7.2.5. Primal-dual convergence. We have observed empirically that convergence occurs for the implicit primal sequence $(\boldsymbol{X}_t)$, the dual sequence $(\boldsymbol{y}_t)$, and the posterior error bound (6.8) generated by SketchyCGAL. See subsection SM5.3 for numerical evidence.

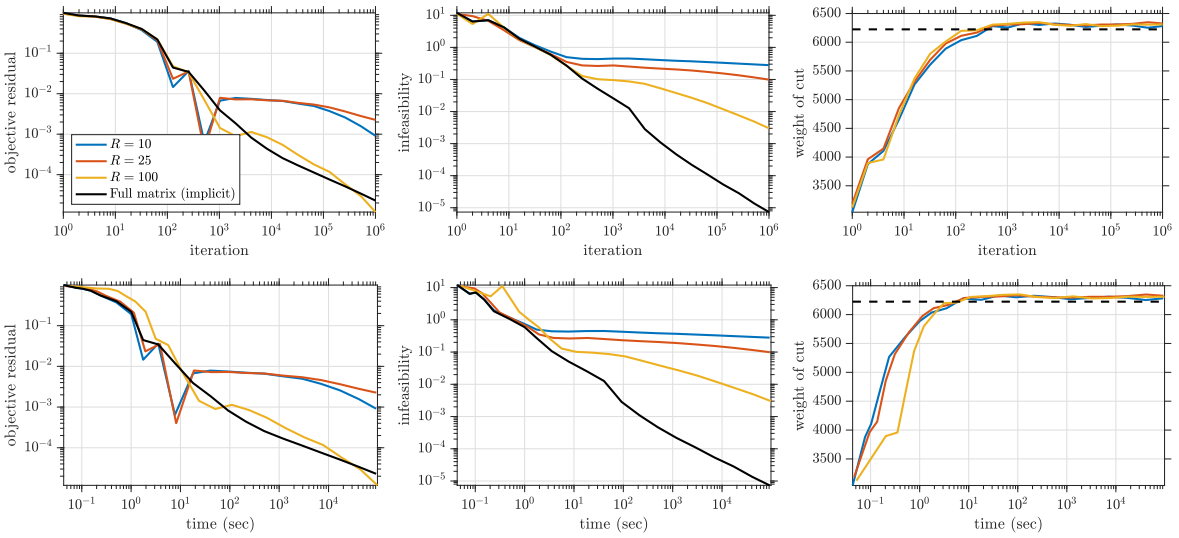


Figure 7.1. *MaxCut SDP: Convergence. We solve the MaxCut SDP for the G67 dataset ($n = 10\,000$) with SketchyCGAL. The subplots show the suboptimality (left), infeasibility (center), and cut value (right) of the implicit iterates and low-rank approximations for sketch size $R \in \{10, 25, 100\}$ as a function of iteration (top) and runtime (bottom). The dashed line is the cut value of a high-accuracy SDPT3 solution. See subsection 7.2.4.*

7.2.6. Hard MaxCut instances. Waldspurger and Waters [97] construct instances of the MaxCut SDP (1.3) that are challenging for algorithms based on the Burer–Monteiro [24] factorization heuristic (subsection 8.4). Each instance has a unique solution and the solution has rank 1, but Burer–Monteiro methods require factorization rank $R = \Theta(\sqrt{n})$, resulting in storage cost $\Theta(n^{3/2})$. In subsection SM5.4, we give numerical evidence that SketchyCGAL can solve these instances with sketch size $R = 2$, achieving the optimal storage $\Theta(n)$. We confirm that Burer–Monteiro usually fails in the optimal-storage regime.

7.3. Abstract phase retrieval. Phase retrieval is the problem of reconstructing a complex-valued signal from intensity-only measurements. It arises in interferometry [38], speech processing [15], array imaging [30], microscopy [48], and many other applications. We will outline a standard method [30, 48] for performing phase retrieval by means of an SDP.

This section uses synthetic instances of a phase retrieval SDP to compare the scaling behavior of SketchyCGAL and CGAL. We also consider a third algorithm ThinCGAL, inspired by [104], that maintains a thin SVD of the matrix variable via rank-one updates [22].

7.3.1. Phase retrieval SDPs. Let $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ be an unknown (discrete) signal. For known vectors $\boldsymbol{a}_i \in \mathbb{C}^n$, we acquire measurements of the form

$$b_i = |\langle \boldsymbol{a}_i, \boldsymbol{\chi}_\natural \rangle|^2 \quad \text{for} \quad i = 1, 2, 3, \ldots, d. \tag{7.2}$$

Abstract phase retrieval is the challenging problem of recovering $\boldsymbol{\chi}_\natural$ from $\boldsymbol{b}$.

Let us summarize a lifting approach introduced by Balan et al. [14]. The key idea is to replace the signal vector $\boldsymbol{x}_\natural$ by the matrix $\boldsymbol{X}_\natural = \boldsymbol{\chi}_\natural \boldsymbol{\chi}_\natural^*$. Then rewrite (7.2) as

$$b_i = \boldsymbol{a}_i^* \boldsymbol{X}_\natural \boldsymbol{a}_i = \langle \boldsymbol{A}_i, \boldsymbol{X}_\natural \rangle, \quad \text{where} \quad \boldsymbol{A}_i = \boldsymbol{a}_i \boldsymbol{a}_i^* \quad \text{for } i = 1, 2, 3, \ldots, d. \tag{7.3}$$

Promoting the implicit constraints on $\boldsymbol{X}_\natural$ and forming the $\boldsymbol{A}_i$ into a linear map $\mathcal{A}$, we can express the problem of finding $\boldsymbol{X}_\natural$ as a feasibility problem with a matrix variable:

$$\text{find} \quad \boldsymbol{X} \in \mathbb{S}_n \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \text{ is psd}, \quad \operatorname{rank}(\boldsymbol{X}) = 1.$$

To reach a tractable convex formulation, we pass to a trace minimization SDP [37]:

$$\text{minimize} \quad \operatorname{tr} \boldsymbol{X} \quad \text{subject to} \quad \mathcal{A}\boldsymbol{X} = \boldsymbol{b}, \quad \boldsymbol{X} \text{ is psd}, \quad \operatorname{tr} \boldsymbol{X} \leq \alpha. \tag{7.4}$$

The parameter α is an upper bound on the signal energy $\|\boldsymbol{\chi}_\natural\|^2$, which can be estimated from the observed data $\boldsymbol{b}$; see [104] for the details. We can solve the SDP (7.4) via SketchyCGAL.

7.3.2. Rounding. Suppose that we have obtained an approximate solution $\boldsymbol{X}$ to (7.4). To estimate the signal $\boldsymbol{\chi}_\natural$, we form the vector $\boldsymbol{\chi} = \sqrt{\lambda}\boldsymbol{u}$ where $(\lambda, \boldsymbol{u})$ is a maximum eigenpair of $\boldsymbol{X}$. Both SketchyCGAL and ThinCGAL return eigenvalue decompositions, so this step is trivial. For CGAL, we use the MATLAB function `eigs` to perform this computation.

7.3.3. Dataset and evaluation. We consider synthetic phase retrieval instances. For each $n \in \{10^2, 10^3, \ldots, 10^6\}$, we generate 20 independent datasets as follows. First, draw $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ from the complex standard normal distribution. Then acquire $d = 12n$ phaseless measurements (7.2) using the coded diffraction pattern model [28]; see subsection SM6.1. The induced linear maps $\mathcal{A}$ and $\mathcal{A}^*$ can be applied via the fast Fourier transform (FFT).

The relative error in a signal reconstruction $\boldsymbol{\chi}$ is given by $\min_{\phi \in \mathbb{R}} \|\mathrm{e}^{\mathrm{i}\phi}\boldsymbol{\chi} - \boldsymbol{\chi}_\natural\| / \|\boldsymbol{\chi}_\natural\|$. In the SDP (7.4), we set $\alpha = 3n$ to demonstrate insensitivity of the algorithm to the choice of α.

7.3.4. Storage and arithmetic comparisons. For each algorithm, we report the storage cost and runtime required to produce a signal estimate $\boldsymbol{\chi}$ with (exact) relative error below 10^{-2}. We invoke SketchyCGAL with sketch size parameter $R = 5$.

Figure 7.2 displays the outcome. We witness the benefit of sketching for both storage and arithmetic. CGAL fails for all large-scale instances ($n = 10^5$ and 10^6) due to storage allocation. For the same reason, ThinCGAL fails for $n = 10^6$. SketchyCGAL successfully solves all problem instances to the target accuracy.

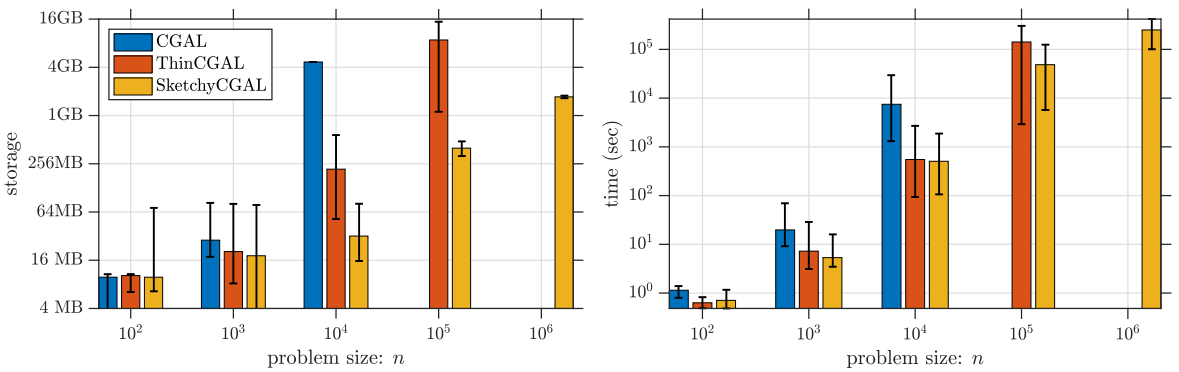


Figure 7.2. *Phase retrieval SDP: Scalability. Storage cost (left) and runtime (right) to solve random instances with algorithms CGAL, ThinCGAL, and SketchyCGAL. The height of each bar is the mean; the interval marks the minimum and maximum over* 20 *trials. Missing bars indicate total failure. See subsection* 7.3.4.

7.4. Phase retrieval in microscopy. Next, we study a more realistic phase retrieval problem that arises from a type of microscopy system [110] called Fourier ptychography (FP). Phase retrieval SDPs offer a potential approach to FP imaging [48]. This section shows that SketchyCGAL can successfully solve the difficult phase retrieval SDPs that arise from FP.

7.4.1. Fourier ptychography. FP microscopes circumvent the physical limits of a simple lens to achieve high-resolution and wide field-of-view simultaneously [110]. To do so, an FP microscope illuminates a sample from many angles and uses a simple lens to collect low-resolution intensity-only images. The measurements are low-pass filters, whose transfer functions depend on the lens and the angle of illumination [48]. From the data, we form a high-resolution image by solving a phase retrieval problem, e.g., via the SDP (7.4).

The high-resolution image of the sample is represented by a Fourier-domain vector $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$. We acquire d intensity-only measurements of the form (7.2), where d is the total number of pixels in the low-resolution illuminations. The low-pass measurements are encoded in vectors $\boldsymbol{a}_i$. The operators $\mathcal{A}$ and $\mathcal{A}^*$, built from the matrices $\boldsymbol{A}_i = \boldsymbol{a}_i \boldsymbol{a}_i^*$, can be applied via the FFT.

7.4.2. Dataset and evaluation. The authors of [48] provided transmission matrices $\boldsymbol{A}_i$ of a working FP system. We simulate this system in the computer environment to acquire noiseless intensity-only measurements of a high-resolution target image [33]. In this setup, $\boldsymbol{\chi}_\natural \in \mathbb{C}^n$ corresponds to the Fourier transform of an $n = 320^2 = 102,400$ pixel grayscale image. We normalize $\boldsymbol{\chi}_\natural$ so that $\|\boldsymbol{\chi}_\natural\| = 1$. We acquire 225 low-resolution illuminations of the original image, each with $64^2 = 4,096$ pixels. The total number of measurements is $d = 921,600$.

We evaluate the error of an estimate $\boldsymbol{\chi}$ as in subsection 7.3.3. Although we know that the signal energy equals 1, it is more realistic to approximate the signal energy by setting $\alpha = 1.5$ in the SDP (7.4).

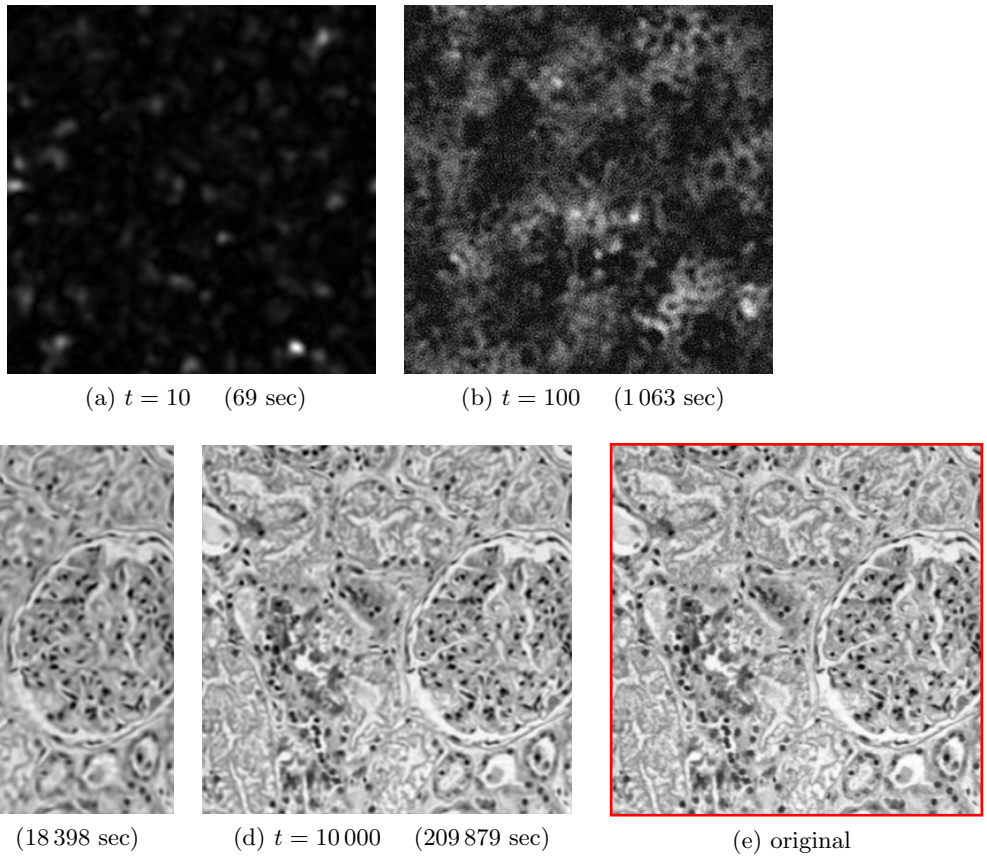
7.4.3. FP imaging. We solve the phase retrieval SDP (7.4) by performing $10,000$ iterations of SketchyCGAL with rank parameter $R = 5$. The top eigenvector of the output gives an approximation $\boldsymbol{\chi} \in \mathbb{C}^n$ of the signal. The inverse Fourier transform of $\boldsymbol{\chi}$ is the desired image.

Figure 7.3 displays the image reconstruction after $t \in \{10, 10^2, 10^3, 10^4\}$ iterations. We obtain a good-quality result in 5 hours after $1,000$ iterations; a sharper image emerges in 59 hours after $10,000$ iterations are complete. We believe the computational time can be reduced substantially with a parallel or GPU implementation. Nevertheless, it is gratifying that we have solved a difficult SDP whose matrix variable has over 10^{10} entries. See subsection SM6.2 for a larger FP instance with $n = 640^2$ pixels.

7.5. The quadratic assignment problem. The QAP is a very difficult combinatorial optimization problem that includes the traveling salesman, max-clique, and bandwidth problems and many others as special cases [66]. SDP relaxations offer a powerful approach for obtaining good solutions to large QAP problems [108]. In this section, we demonstrate that SketchyCGAL can solve these challenging SDPs.

7.5.1. QAP. We begin with the simplest form of the QAP. Fix symmetric $\mathtt{n} \times \mathtt{n}$ matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{S}_{\mathtt{n}}$ where $\mathtt{n}$ is a natural number. We wish to "align" the matrices by solving

$$\text{minimize} \quad \operatorname{tr}(\boldsymbol{A}\boldsymbol{\Pi}\boldsymbol{B}\boldsymbol{\Pi}^*) \quad \text{subject to} \quad \boldsymbol{\Pi} \text{ is an } \mathtt{n} \times \mathtt{n} \text{ permutation matrix.} \tag{7.5}$$



(a) $t = 10$ (69 sec) (b) $t = 100$ (1 063 sec)

(c) $t = 1\,000$ (18 398 sec) (d) $t = 10\,000$ (209 879 sec) (e) original

Figure 7.3. *Phase retrieval SDP: Imaging. Reconstruction of an $n = 320^2$ pixel image from FP data. We solve an $n \times n$ phase retrieval SDP via SketchyCGAL with rank parameter $R = 5$ and show the images obtained at iterations $t = 10^3, 10^4$. The last subfigure is the original. See subsection 7.4.3.*

Recall that a permutation matrix $\mathbf{\Pi}$ has precisely one nonzero entry in each row and column and that nonzero entry equals one. A brute force search over the $\mathtt{n}!$ permutation matrices of size $\mathtt{n}$ quickly becomes intractable as $\mathtt{n}$ grows. Unsurprisingly, the QAP problem (7.5) is NP-hard [86]. Instances with $\mathtt{n} > 30$ usually cannot be solved in reasonable time.

7.5.2. Relaxations. There is an extensive literature on SDP relaxations for QAP, beginning with the work [108] of Zhao et al. We consider a weaker relaxation inspired by [49, 23]:

$$
\begin{aligned}
\text{minimize} \quad & \operatorname{tr}[(\boldsymbol{B} \otimes \boldsymbol{A})\mathsf{Y}] \\
\text{subject to} \quad & \operatorname{tr}_1(\mathsf{Y}) = \mathbf{I}, \quad \operatorname{tr}_2(\mathsf{Y}) = \mathbf{I}, \quad \mathcal{G}(\mathsf{Y}) \geq 0, \\
& \operatorname{vec}(\boldsymbol{P}) = \operatorname{diag}(\mathsf{Y}), \quad \boldsymbol{P}\mathbf{1} = \mathbf{1}, \quad \mathbf{1}^*\boldsymbol{P} = \mathbf{1}^*, \quad \boldsymbol{P} \geq \mathbf{0}, \\
& \begin{bmatrix} 1 & \operatorname{vec}(\boldsymbol{P})^* \\ \operatorname{vec}(\boldsymbol{P}) & \mathsf{Y} \end{bmatrix} \succcurlyeq \mathbf{0}, \quad \operatorname{tr} \mathsf{Y} = n.
\end{aligned}
\tag{7.6}
$$

We have written $\otimes$ for the Kronecker product.

The constraint $\mathcal{G}(\mathsf{Y}) \geq 0$ enforces nonnegativity of a subset of the entries in Y. In the Zhao et al. relaxation, $\mathcal{G}$ is the identity map, so it yields $O(\mathtt{n}^4)$ constraints. We reduce the complexity by choosing $\mathcal{G}$ more carefully. In our formulation, $\mathcal{G}$ extracts precisely the nonzero entries of the matrix $\boldsymbol{B} \otimes \mathbf{1}\mathbf{1}^*$. This is beneficial because $\boldsymbol{B}$ is sparse in many applications. For example, in traveling salesman and bandwidth problems, $\boldsymbol{B}$ has $O(\mathtt{n})$ nonzero entries, so the map $\mathcal{G}$ produces only $O(\mathtt{n}^3)$ constraints.

The main variable Y in (7.6) has dimension $\mathtt{n}^2 \times \mathtt{n}^2$. As a consequence, the problem has $O(\mathtt{n}^4)$ degrees of freedom, together with $O(\mathtt{n}^3)$ to $O(\mathtt{n}^4)$ constraints (depending on $\mathcal{G}$). The explosive growth of this relaxation scuttles most algorithms by the time $\mathtt{n} > 50$. To solve larger instances, many researchers resort to even weaker relaxations.

In contrast, we can solve the relaxation (7.6) directly using SketchyCGAL, up to $\mathtt{n} = 150$. By limiting the number of inequality constraints, via the operator $\mathcal{G}$, we achieve substantial reductions in resource usage. We validate our algorithm on QAPs where the exact solution is known, and we compare the performance with algorithms [107, 23] for other relaxations.

7.5.3. Rounding. Given an approximate solution to (7.6), we use a rounding method to construct a permutation. SketchyCGAL returns a matrix $\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$ where $\boldsymbol{U}$ has dimension $(\mathtt{n}^2 + 1) \times R$. We extract the first column of $\boldsymbol{U}$, discard its first entry, and reshape the remaining part into an $\mathtt{n} \times \mathtt{n}$ matrix. Then we project this matrix onto the set of permutation matrices via the Hungarian method [58, 71, 52]. This yields a feasible point $\boldsymbol{\Pi}$ for the problem (7.5). We repeat this procedure for all R columns of $\boldsymbol{U}$, and we pick the one that minimizes $\operatorname{tr}(\boldsymbol{A}\boldsymbol{\Pi}\boldsymbol{B}\boldsymbol{\Pi}^*)$. This permutation gives an upper bound on the optimal value of (7.5).

7.5.4. Datasets and evaluation. We consider instances from QAPLIB [27] and TSPLIB [83] that are used in [23]. The optimal values are known, and the permutation size $\mathtt{n}$ varies between 12 and 150. (Recall that the SDP matrix dimension $n = \mathtt{n}^2 + 1$.) We report

$$\texttt{relative gap } \% = \frac{\texttt{upper bound obtained - optimum}}{\texttt{optimum}} \times 100. \tag{7.7}$$

7.5.5. Solving QAPs. To solve (7.6), we cannot use the scaling (7.1) because $\|\mathcal{A}\|$ is not available; see the source code for our approach. We apply SketchyCGAL with sketch size $R = \mathtt{n}$, so the sketch uses storage $\Theta(\mathtt{n}^3)$. After rounding, a low- or medium-accuracy solution of (7.6) often provides a better permutation than a high-accuracy solution. Therefore, we applied the rounding step at iterations $2, 4, 8, 16, \ldots$ and tracked the quality of the best permutation attained on the solution path. We stopped after (the first of) 10^6 iterations or 72 hours.

The results of this experiment appear in Figure 7.4. We compare against the best value reported by Bravo Ferreira, Khoo, and Singer in [23, Tables 4 and 6] for their CSDP method with clique size $k \in \{2, 3, 4\}$. We also include the results that [23] lists for the PATH method [107].

SketchyCGAL allows us to solve a tighter SDP relaxation of QAP than the other methods (CSDP, PATH). As a consequence, we obtain significantly smaller gaps for most instances.

8. Related work.

There is a large body of literature on numerical methods for solving SDPs; see [67] for a recent survey. Here we describe the SDP literature just enough to contextualize our contributions.

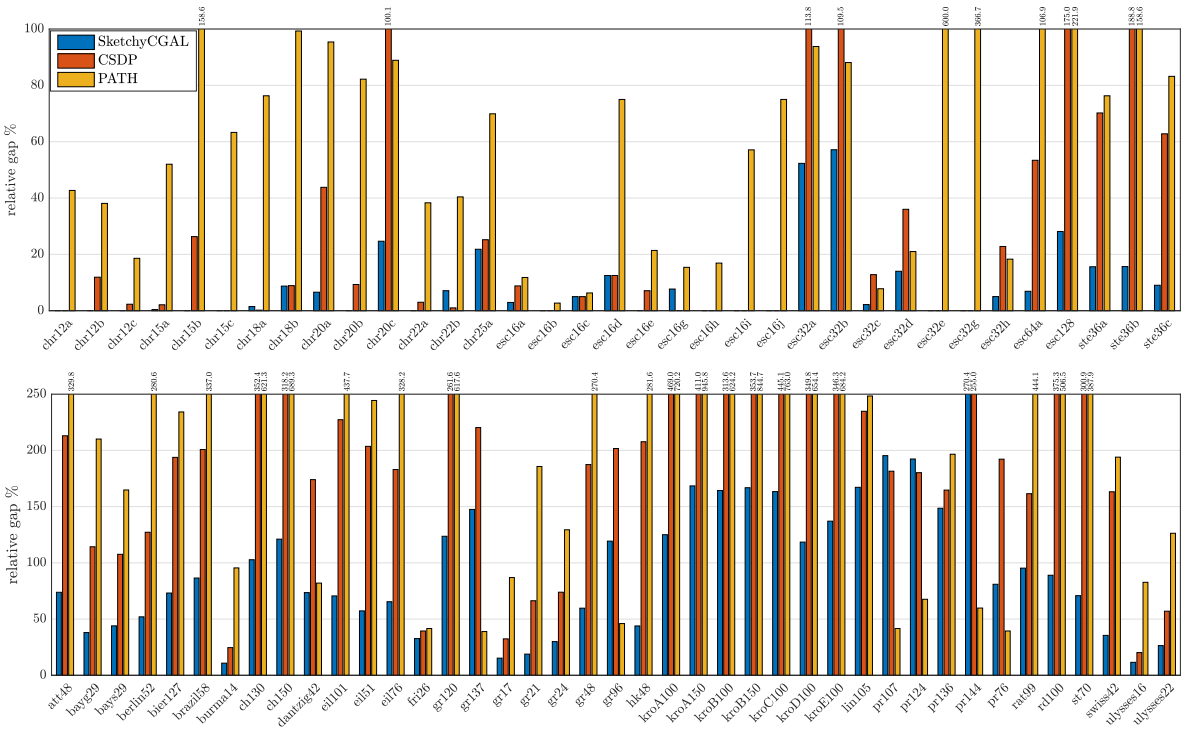


Figure 7.4. *QAP relaxation: Solution quality. Using SketchyCGAL, the CSDP method [23], and the PATH method [107], we solve SDP relaxations of QAP instances from QAPLIB (top) and TSPLIB (bottom). The bars compare the cost of the computed solution against the optimal value; shorter is better. See subsection 7.5.5.*

8.1. Standard methodologies. First, we outline the approaches that drive most of the reliable, general-purpose SDP software packages that are currently available.

Interior-point methods (IPMs) [75, 76, 3, 4, 53, 54] reformulate the SDP as an unconstrained problem and take an (approximate) Newton step at each iteration. In exchange, they deliver quadratic convergence. Hence IPMs are widely used to solve SDPs to high precision [56, 9, 73, 17]. Software packages include SeDuMi [90], MoSeK [70], and SDPT3 [92]. Alas, IPMs do not scale to large problems: to solve the Newton system we must store and factor large, dense matrices. A typical IPM for the SDP (2.2) requires $\Theta(n^3 + d^2n^2 + d^3)$ arithmetic operations per iteration and $\Theta(n^2 + dn + d^2)$ memory [6].

Several effective SDP solvers are based on the augmented Lagrangian paradigm [26, 98, 109, 99]. In particular, Zhao, Sun, and Toh [109] employ a semismooth Newton method and the conjugate gradient method to solve the subproblems. Their method is enhanced and implemented in the software package SDPNAL+ [100]. Augmented Lagrangian methods for SDPs typically require storage $\Omega(n^2)$.

8.2. First-order methods. First-order methods use only gradient information to solve the SDP to reduce runtime and storage requirements. We focus on *projection-free* algorithms, suitable large SDPs, that do not require full SVD computations. Major first-order methods for SDP include approaches based on the CGM [39, 63, 51] and extensions that handle more complex constraints [32, 46, 50], primal-dual subgradient algorithms [74, 105], and the matrix

multiplicative weight (MMW) method [96, 10] or, equivalently, the mirror-prox algorithm with the quantum entropy mirror map [72].

The standard CGM algorithm does not apply to the model problem (2.2) because of the affine constraint $\mathcal{A}\boldsymbol{X} = \boldsymbol{b}$. Several variants [42, 65, 87, 103] of CGM can handle affine constraints. In particular, CGAL [102] does so by applying CGM to an augmented Lagrangian formulation. We have chosen to extend CGAL because of its strong empirical performance and its robustness to inexact eigenvector computations; see [102, sect. 4-5].

Primal-dual subgradient methods perform subgradient ascent on the dual problem; the cost of each iteration is dominated by an eigenvector computation. Nesterov [74] constructs primal iterates by proximal mapping. The algorithm in Yurtsever, Tran Dinh, and Cevher [105] constructs primal iterates by an averaging technique, similar to the updates in CGM; this method involves a line search, so it requires very accurate eigenvector calculations.

The MMW method is derived by reducing the SDP to a sequence of feasibility problems. These are reformulated as a primal-dual game whose dual is an eigenvalue optimization problem. The resulting MMW algorithm can be interpreted as performing gradient descent in a dual space and using the matrix exponential map to transfer information back to the primal space. To scale this approach to larger problems, researchers have proposed linearization, random projection, sparsification techniques, and stochastic Lanczos quadrature to approximate the matrix exponential [11, 7, 80, 40, 41, 8, 13, 61, 29]. Even so, the reduction to a sequence of feasibility problems makes this technique impractical for general SDPs. We are aware of only one computational evaluation of the MMW idea [13].

8.3. Storage considerations. Almost all provably correct SDP algorithms store and operate on a full-dimensional matrix variable, so they are not suitable for very large SDPs.

Some primal-dual subgradient methods and CGM variants build an approximate solution as a convex combination of rank-one updates, so the rank of the solution does not exceed the number of iterations. This fact has led researchers to call these methods "storage-efficient," but this claim is misleading because the algorithms require many iterations to converge.

In the conference paper [106], we observed that certain types of optimization algorithms can be combined with sketching to control storage costs. As a first example, we augmented CGM with sketching to obtain a new algorithm called SketchyCGM. This method solves a special class of low-rank matrix optimization problems that arise in statistics and machine learning. We believe that SketchyCGM is the first algorithm for this class of problems that provably succeeds with optimal storage.

To develop SketchyCGAL, we changed the base optimization algorithm (to CGAL) so that we can solve standard-form SDPs. We switched to a simpler sketching technique (the Nyström sketch) that has better empirical performance. We also analyzed how accurately to solve the eigenvalue problems to ensure that SketchyCGAL succeeds (section SM1), and we deployed an approximate eigenvalue computation method (randomized Lanczos) that meets our needs. Altogether, this effort leads to a storage-optimal algorithm that works for all standard-form SDPs with a minimum of tuning. Reducing the storage has the ancillary benefit of reducing arithmetic and communication costs, which also improves scalability.

In concurrent work with Ding [36], a subset of the authors developed a new *approximate complementarity principle* that also yields a storage-optimal algorithm for standard-form

SDPs. This approach uses a suboptimal dual point to approximate the range of the primal solution to the SDP. By compressing the primal problem to this subspace, we can solve the primal SDP with limited storage. This method, however, has more limited guarantees than SketchyCGAL. A numerical evaluation is in progress.

8.4. Nonconvex Burer–Monteiro methods. The most famous approach to low-storage semidefinite programming is the factorization heuristic proposed by Homer and Peinado [47] and refined by Burer and Monteiro (BM) [24]. The main idea is to reformulate the model problem (2.2) by expressing the psd matrix variable $\boldsymbol{X} = \boldsymbol{F}\boldsymbol{F}^*$ in terms of a factor $\boldsymbol{F} \in \mathbb{R}^{n\times R}$, where the rank parameter $R \ll n$. That is,

$$\text{minimize} \quad \langle \boldsymbol{C}, \boldsymbol{F}\boldsymbol{F}^* \rangle \quad \text{subject to} \quad \mathcal{A}(\boldsymbol{F}\boldsymbol{F}^*) = \boldsymbol{b}, \quad \boldsymbol{F}\boldsymbol{F}^* \in \alpha\boldsymbol{\Delta}_n. \tag{8.1}$$

This approach controls storage by sacrificing convexity and the associated guarantees.

Many nonlinear programming methods have been applied to optimize (8.1). Augmented Lagrangian methods are commonly used [24, 25, 59, 85, 88]. The most popular research software based on BM factorization is Manopt [20], which implements manifold optimization algorithms including Riemannian gradient and Riemannian trust region methods [1]. Consequently, Manopt is limited to problems where the factorized formulation (8.1) defines a smooth manifold.

There has been an intense effort to establish theoretical results for the BM factorization approach. It is clear that every solution to the SDP (2.2) of rank R or less is also a solution to the factorized problem (8.1). On the other hand, (8.1) may admit local minima that are not global minima of (2.2). Some guarantees are available. For example, if $\boldsymbol{C}$ is generic and the constraint set of (8.1) is a smooth manifold and $R \geq \sqrt{2(d+1)}$, then each second-order critical point of (8.1) is a global optimum [21]. A second-order critical point can be located using a Riemannian trust region method. See [19, 81, 31] for additional theoretical analysis.

The storage and arithmetic costs of solving (8.1) depend on the factorization rank R. Unfortunately, the BM method may fail when $R = o(\sqrt{d})$. Below this threshold, the BM formulation (8.1) can have spurious solutions (second-order critical points that are not globally optimal), and the bad problem instances can form a set of positive measure [97]. Hence the BM approach cannot support provably correct algorithms with storage costs better than $\Omega(n\sqrt{d})$. See subsection SM5.4 for numerical evidence.

9. Conclusion. We have presented a practical, new approach for solving SDPs at scale. Our algorithm, SketchyCGAL, combines a primal-dual optimization method with randomized linear algebra techniques to achieve unprecedented guarantees when the problem is weakly constrained and the solution is approximately low rank. We hope that our ideas lead to further algorithmic advances and support new applications of semidefinite programming.

SketchyCGAL is currently limited by the arithmetic cost of solving large eigenvalue problems to increasing accuracy. It also falters for SDPs with a large number of constraints because it depends on a primal-dual approach. Moreover, our analysis does not fully explain the observed behavior of the algorithm, including the rate of convergence of the primal variable or the convergence of the dual variable and the surrogate duality gap. These topics merit further research.

Acknowledgments. Nicolas Boumal encouraged us to reconsider the storage-optimal randomized Lanczos method (Algorithm 4.1); this algorithm simplifies the presentation while improving our theoretical and numerical results. Richard Kueng allowed us to include his results on trace normalization (step 13 in Algorithm 6.1). Irène Waldspurger provided code that generates instances of the MaxCut problem that are difficult for the Burer–Monteiro heuristic. We would also like to thank the editor and reviewers for their feedback.

REFERENCES

[1] P.-A. Absil, C. Baker, and K. Gallivan, *Trust-region methods on Riemannian manifolds*, Found. Comput. Math., 7 (2007), pp. 303–330.
[2] A. Y. Alfakih, A. Khandani, and H. Wolkowicz, *Solving Euclidean distance matrix completion problems via semidefinite programming*, Comput. Optim. Appl., 12 (1999), pp. 13–30.
[3] F. Alizadeh, *Combinatorial Optimization with Interior Point Methods and Semi-Definite Matrices*, Ph.D. thesis, University of Minnesota, 1991.
[4] F. Alizadeh, *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM J. Optim., 5 (1995), pp. 13–51.
[5] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, *Complementarity and nondegeneracy in semidefinite programming*, Math. Program., 77 (1997), pp. 111–128.
[6] F. Alizadeh, J.-P. A. Haeberly, and M. L. Overton, *Primal-dual interior point methods for semidefinite programming: Convergence rates, stability and numerical results*, SIAM J. Optim., 8 (1998), pp. 746–768.
[7] Z. Allen-Zhu, Y. T. Lee, and L. Orecchia, *Using Optimization to Obtain a Width-Independent, Parallel, Simpler, and Faster Positive SDP Solver*, https://arxiv.org/abs/1507.02259, 2015.
[8] Z. Allen-Zhu and Y. Li, *Follow the Compressed Leader: Faster Online Learning of Eigenvectors and Faster MMWU*, https://arxiv.org/abs/1701.01722, 2017.
[9] M. F. Anjos and J. B. Lasserre, *Handbook on Semidefinite, Conic and Polynomial Optimization*, Springer, New York, 2012.
[10] S. Arora, E. Hazan, and S. Kale, *Fast algorithms for approximate semidefinite programming using the multiplicative weights update method*, in Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, Washington, DC, 2005, pp. 339–348.
[11] S. Arora and S. Kale, *A combinatorial, primal-dual approach to semidefinite programs*, J. ACM, 63 (2016), pp. 12:1–12:35.
[12] D. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, *Graph partitioning and graph clustering*, in 10th DIMACS Implementation Challenge Workshop, 2012, https://www.cise.ufl.edu/research/sparse/matrices/DIMACS10/index.html.
[13] M. Baes, M. Bürgisser, and A. Nemirovski, *A randomized mirror-prox method for solving structured large-scale matrix saddle-point problems*, SIAM J. Optim., 23 (2013), pp. 934–962.
[14] R. Balan, B. G. Bodmann, P. G. Casazza, and D. Edidin, *Painless reconstruction from magnitudes of frame coefficients*, J. Fourier Anal. Appl., 15 (2009), pp. 488–501.
[15] R. Balan, P. Casazza, and D. Edidin, *On signal reconstruction without phase*, Appl. Comput. Harmon. Anal., 20 (2006), pp. 345–356.
[16] A. Barvinok, *A Course in Convexity*, AMS, Providence, RI, 2002.
[17] A. Ben-Tal and A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, MOS-SIAM Ser. Optim. Z, SIAM, Philadelphia, 2001.
[18] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Comput. Sci. Appl. Math., Academic Press, New York, 1982.
[19] S. Bhojanapalli, N. Boumal, P. Jain, and P. Netrapalli, *Smoothed analysis for low-rank solutions to semidefinite programs in quadratic penalty form*, in Proceedings of the 31st Conference on Learning Theory, PMLR 75, 2018, pp. 3243–3270.
[20] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, *Manopt, a MATLAB toolbox for optimization on manifolds*, J. Mach. Learn. Res., 15 (2014), pp. 1455–1459.

[21] 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 29, 2016, pp. 2757–2765.
[22] M. Brand, *Fast low-rank modifications of the thin singular value decomposition*, Linear Algebra Appl., 415 (2006), pp. 20 – 30.
[23] J. F. S. Bravo Ferreira, Y. Khoo, and A. Singer, *Semidefinite programming approach for the quadratic assignment problem with a sparse graph*, Comput. Optim. Appl., 69 (2018), pp. 677–712.
[24] S. Burer and R. D. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Math. Program., 95 (2003), pp. 329–357.
[25] S. Burer and R. D. Monteiro, *Local minima and convergence in low-rank semidefinite programming*, Math. Program., 103 (2005), pp. 427–444.
[26] S. Burer and D. Vandenbussche, *Solving lift-and-project relaxations of binary integer programs*, SIAM J. Optim., 16 (2006), pp. 726–750.
[27] R. E. Burkard, S. E. Karisch, and F. Rendl, *QAPLIB—a quadratic assignment problem library*, J. Global Optim., 10 (1997), pp. 391–403.
[28] E. J. Candès, X. Li, and M. Soltanolkotabi, *Phase retrieval from coded diffraction patterns*, Appl. Comput. Harmon. Anal., 39 (2015), pp. 277 – 299.
[29] Y. Carmon, J. C. Duchi, S. Aaron, and T. Kevin, *A rank-1 sketch for matrix multiplicative weights*, in Proceedings of the 32nd Conference on Learning Theory, PMLR 99, 2019, pp. 589–623.
[30] A. Chai, M. Moscoso, and G. Papanicolaou, *Array imaging using intensity-only measurements*, Inverse Problems, 27 (2010), 015005.
[31] D. Cifuentes, *Burer–Monteiro Guarantees for General Semidefinite Programs*, https://arxiv.org/abs/1904.07147, 2019.
[32] K. L. Clarkson, *Coresets, sparse greedy approximation, and the Frank–Wolfe algorithm*, ACM Trans. Algorithms, 6 (2010), pp. 63:1–63:30.
[33] J. L. Conrad, *Marburg Virus Hemorrhagic Fever: Cytoarchitecture and Histopathology*, https://pixnio.com/de/wissenschaft/mikroskopische-aufnahmen/hamorrhagisches-fieber-marburg-virus/cytoarchitectural-histopathologischen-erkannt-niere-probe-marburg-patient.
[34] C. Delorme and S. Poljak, *Laplacian eigenvalues and the maximum cut problem*, Math. Programming, 62 (1993), pp. 557–574.
[35] C. Delorme and S. Poljak, *The performance of an eigenvalue bound on the max-cut problem in some classes of graphs*, Discrete Math., 111 (1993), pp. 145–156.
[36] L. Ding, A. Yurtsever, V. Cevher, J. A. Tropp, and M. Udell, *An Optimal-Storage Approach to Semidefinite Programming Using Approximate Complementarity*, https://arxiv.org/abs/1902.03373, 2019.
[37] M. Fazel, *Matrix Rank Minimization with Applications*, Ph.D. thesis, Stanford University, Palo Alto, CA, 2002.
[38] J. R. Fienup, *Phase retrieval algorithms: A comparison*, Appl. Optics, 21 (1982), pp. 2758–2769.
[39] M. Frank and P. Wolfe, *An algorithm for quadratic programming*, Naval Res. Logist. Quart., 3 (1956), pp. 95–110.
[40] D. Garber and E. Hazan, *Approximating semidefinite programs in sublinear time*, in Advances in Neural Information Processing Systems 25, 2011.
[41] D. Garber and E. Hazan, *Sublinear time algorithms for approximate semidefinite programming*, Math. Programming, 158 (2016), pp. 329–361.
[42] G. Gidel, F. Pedregosa, and S. Lacoste-Julien, *Frank–Wolfe splitting via augmented Lagrangian method*, in Proceedings of the 21st International Conference on Artificial Intelligence and Statistics, PMLR 84, 2018, pp. 1456–1465.
[43] A. Gittens, *Topics in Randomized Numerical Linear Algebra*, Ph.D. thesis, California Institute of Technology, Pasadena, 2013.
[44] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.
[45] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[46] E. Hazan, *Sparse approximate solutions to semidefinite programs*, in LATIN 2008: Theoretical Informatics, 2008, pp. 306–316.
[47] S. Homer and M. Peinado, *Design and performance of parallel and distributed approximation algorithms for maxcut*, J. Parallel Distrib. Comput., 46 (1997), pp. 48–61.
[48] R. Horstmeyer, R. Y. Chen, X. Ou, B. Ames, J. A. Tropp, and C. Yang, *Solving ptychography with a convex relaxation*, New J. Phys., 17 (2015), 053044.
[49] Q. Huang, Y. Chen, and L. Guibas, *Scalable semidefinite relaxation for maximum a posterior estimation*, in Proceedings of the 31st International Conference on Machine Learning, PMLR 32, 2014, pp. 64–72.
[50] M. Jaggi, *Revisiting Frank–Wolfe: Projection-free sparse convex optimization*, in Proceedings of the 30th International Conference on Machine Learning, PMLR 28, 2013, pp. 427–435.
[51] L. K. Jones, *A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training*, Ann. Statist., 20 (1992), pp. 608–613.
[52] R. Jonker and A. Volgenant, *A shortest augmenting path algorithm for dense and sparse linear assignment problems*, Computing, 38 (1987), pp. 325–340.
[53] A. P. Kamath and N. K. Karmarkar, *A continuous method for computing bounds in integer quadratic optimization problems*, J. Global Optim., 2 (1991), pp. 229–241.
[54] A. P. Kamath and N. K. Karmarkar, *An $O(nL)$ iteration algorithm for computing bounds in quadratic optimization problems*, in Complexity in Numerical Optimization, World Scientific, River Edge, NJ, 1993, pp. 254–268.
[55] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Springer, New York, 1972, pp. 85–103.
[56] K. Krishnan and T. Terlaky, *Interior point and semidefinite approaches in combinatorial optimization*, in Graph Theory and Combinatorial Optimization, Springer, New York, 2005.
[57] J. Kuczyński and H. Woźniakowski, *Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1094–1122.
[58] H. Kuhn, *The Hungarian Method for the assignment problem*, Naval Res. Logist. Quart., 2 (1955), pp. 83–97.
[59] B. Kulis, A. C. Surendran, and J. C. Platt, *Fast low-rank semidefinite programming for embedding and clustering*, in Proceedings of the 11th International Conference on Artificial Intelligence and Statistics, Vol. 2, San Juan, Puerto Rico, 2007, pp. 235–242.
[60] J. Lavaei and S. H. Low, *Zero duality gap in optimal power flow problem*, IEEE Trans. Power Syst., 27 (2012), pp. 92–107.
[61] Y. T. Lee and S. Padmanabhan, *An $\tilde{O}(m/\varepsilon^3.5)$-cost algorithm for semidefinite programs with diagonal constraints*, in Proceedings of the Conference on Learning Theory, 2020, pp. 3069–3119.
[62] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, Software Environ. Tools 6, SIAM, Philadelphia, 1998.
[63] E. S. Levitin and B. T. Poljak, *Minimization methods in the presence of constraints*, Ž. Vyčisl. Mat. Mat. Fiz., 6 (1966), pp. 787–823.
[64] H. Li, G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert, *Algorithm 971: An implementation of a randomized algorithm for principal component analysis*, ACM Trans. Math. Software, 43 (2017), 28.
[65] Y.-F. Liu, X. Liu, and S. Ma, *On the nonergodic convergence rate of an inexact augmented lagrangian framework for composite convex programming*, Math. Oper. Res., 44 (2019), pp. 632–650.
[66] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, *A survey for the quadratic assignment problem*, European J. Oper. Res., 176 (2007), pp. 657–690.
[67] A. Majumdar, G. Hall, and A. A. Ahmadi, *A Survey of Recent Scalability Improvements for Semidefinite Programming with Applications to Machine Learning, Control, and Robotics*, http://arXiv.org/abs/1908.05209, 2019.
[68] M. Mesbahi and G. P. Papavassilopoulos, *On the rank minimization problem over a positive semidefinite linear matrix inequality*, IEEE Trans. Automat. Control, 42 (1997), pp. 239–243.
[69] L. Mirsky, *Symmetric gauge functions and unitarily invariant norms*, Quart. J. Math. Oxford Ser. (2), 11 (1960), pp. 50–59.

[70] *The MOSEK Optimization Toolbox for MATLAB Manual. Version* 9.0, Mosek ApS, 2019.
[71] J. Munkres, *Algorithms for the assignment and transportation problems*, J. SIAM, 5 (1957), pp. 32–38.
[72] A. Nemirovski, *Prox-method with rate of convergence $O(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems*, SIAM J. Optim., 15 (2004), pp. 229–251.
[73] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Springer, New York, 2004.
[74] Y. Nesterov, *Primal-dual subgradient methods for convex problems*, Math. Program., 120 (2009), pp. 221–259.
[75] Y. Nesterov and A. Nemirovski, *Self-Concordant Functions and Polynomial Time Methods in Convex Programming*, Central Economic & Mathematic Institute report, USSR Academy of Sciences, 1990.
[76] Y. Nesterov and A. Nemirovski, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia, 1994.
[77] B. N. Parlett, *The symmetric eigenvalue problem*, Classics in Appl. Math. 20, SIAM, Philadelphia, 1998, https://doi.org/10.1137/1.9781611971163.
[78] P. Parrilo, *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*, Ph.D. thesis, California Institute of Technology, Pasadena, 2000.
[79] G. Pataki, *On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues*, Math. Oper. Res., 23 (1998), pp. 339–358.
[80] R. Peng and K. Tangwongsan, *Faster and simpler width-independent parallel algorithms for positive semidefinite programming*, in Proceedings of the 24th Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2012, pp. 101–108.
[81] T. Pumir, S. Jelassi, and N. Boumal, *Smoothed analysis of the low-rank approach for smooth semidefinite programs*, in Advances in Neural Information Processing Systems 31, 2018, pp. 2281–2290.
[82] A. Raghunathan, J. Steinhardt, and P. Liang, *Semidefinite relaxations for certifying robustness to adversarial examples*, in Advances in Neural Information Processing Systems 31, 2018, pp. 10900–10910.
[83] G. Reinelt, *TSPLIB*, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.
[84] 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*, Int. J. Robot. Res., 38 (2019), pp. 95–125.
[85] M. F. Sahin, A. Eftekhari, A. Alacaoglu, F. Latorre, and V. Cevher, *An inexact augmented Lagrangian framework for nonconvex optimization with nonlinear constraints*, in Advances in Neural Information Processing Systems 32, 2019, pp. 13965–13977.
[86] S. Sahni and T. Gonzalez, *P-complete approximation problems*, J. ACM, 23 (1976), pp. 555–565.
[87] A. Silveti-Falls, C. Molinari, and J. Fadili, *Generalized Conditional Gradient with Augmented Lagrangian for Composite Minimization*, https://arxiv.org/abs/1901.01287, 2019.
[88] M. Souto, J. D. Garcia, and A. Veiga, *Exploiting Low-Rank Structure in Semidefinite Programming by Approximate Operator Splitting*, https://arxiv.org/abs/1810.05231, 2018.
[89] N. Srebro, *Learning with Matrix Factorizations*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, 2004.
[90] J. Sturm, *Using SeDuMi* 1.02*, a MATLAB toolbox for optimization over symmetric cones*, Optim. Methods Softw., 11–12 (1999), pp. 625–653.
[91] Y. Sun, Y. Guo, J. A. Tropp, and M. Udell, *Tensor random projection for low memory dimension reduction*, in NeurIPS Workshop on Relational Representation Learning, 2018.
[92] K. A. Toh, M. J. Todd, and R. H. Tütüncü, *SDPT*3*—a Matlab software package for semidefinite programming, optimization methods and software*, Optim. Methods Softw., 11 (1999), pp. 545–581.
[93] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Fixed-rank approximation of a positive-semidefinite matrix from streaming data*, in Advances in Neural Information Processing Systems 30, 2017, pp. 1225–1234.
[94] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1454–1485.
[95] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Streaming low-rank matrix approximation with an application to scientific simulation*, SIAM J. Sci. Comput., 41 (2019), pp. A2430–A2463.

[96] K. Tsuda, G. Rätsch, and M. K. Warmuth, *Matrix exponentiated gradient updates for on-line learning and Bregman projections*, J. Mach. Learn. Res., 6 (2005), pp. 995–1018.
[97] I. Waldspurger and A. Waters, *Rank optimality for the Burer–Monteiro factorization*, https://arxiv.org/abs/1812.03046, 2018.
[98] Z. Wen, D. Goldfarb, S. Ma, and K. Scheinberg, *Row by Row Methods for Semidefinite Programming*, IEOR report, Columbia University, 2009.
[99] Z. Wen, D. Goldfarb, and W. Yin, *Alternating direction augmented Lagrangian methods for semidefinite programming*, Math. Program. Comput., 2 (2010), pp. 203–230.
[100] L. Yang, D. Sun, and K.-C. Toh, *SDPNAL+: A majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints*, Math. Progam. Comput., 7 (2015), pp. 331–366.
[101] Y. Ye, *GSet Random Graphs*, https://www.cise.ufl.edu/research/sparse/matrices/Gset/.
[102] A. Yurtsever, O. Fercoq, and V. Cevher, *A conditional-gradient-based augmented Lagrangian framework*, in Proceedings of the 36th International Conference on Machine Learning, PMLR 97, 2019, pp. 7272–7281.
[103] A. Yurtsever, O. Fercoq, F. Locatello, and V. Cevher, *A conditional gradient framework for composite convex minimization with applications to semidefinite programming*, in Proceedings of the 35th International Conference on Machine Learning, PMLR 80, 2018, pp. 5727–5736.
[104] A. Yurtsever, Y.-P. Hsieh, and V. Cevher, *Scalable convex methods for phase retrieval*, in Proceedings of the 6th International IEEE Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2015, pp. 381–384.
[105] A. Yurtsever, Q. Tran Dinh, and V. Cevher, *A universal primal-dual convex optimization framework*, in Advances in Neural Information Processing Systems 28, 2015, pp. 3150–3158.
[106] A. Yurtsever, M. Udell, J. Tropp, and V. Cevher, *Sketchy decisions: Convex low-rank matrix optimization with optimal storage*, in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, PMLR, 54, 2017, pp. 1188–1196.
[107] M. Zaslavskiy, F. Bach, and J. Vert, *A path following algorithm for the graph matching problem*, IEEE Trans. Pattern Anal. Mach. Intel., 31 (2009), pp. 2227–2242.
[108] Q. Zhao, S. E. Karisch, F. Rendl, and H. Wolkowicz, *Semidefinite programming relaxations for the quadratic assignment problem*, J. Combin. Optim., 2 (1998), pp. 71–109.
[109] X.-Y. Zhao, D. Sun, and K.-C. Toh, *A Newton-CG augmented Lagrangian method for semidefinite programming*, SIAM J. Optim., 20 (2010), pp. 1737–1765.
[110] G. Zheng, R. Horstmeyer, and C. Yang, *Wide-field, high-resolution Fourier ptychography microscopy*, Nat. Photonics, 7 (2013), pp. 739–745.