

Quantum-Inspired Sublinear Algorithm for Solving Low-Rank Semidefinite Programming

Nai-Hui Chia

Department of Computer Science, University of Texas at Austin, TX, USA
nai@cs.utexas.edu

Tongyang Li

Department of Computer Science, Institute for Advanced Computer Studies, and Joint Center for Quantum Information and Computer Science, University of Maryland, College Park, MD, USA
tongyang@cs.umd.edu

Han-Hsuan Lin

Department of Computer Science, University of Texas at Austin, TX, USA
linhh@cs.utexas.edu

Chunhao Wang

Department of Computer Science, University of Texas at Austin, TX, USA
chunhao@cs.utexas.edu

Abstract

Semidefinite programming (SDP) is a central topic in mathematical optimization with extensive studies on its efficient solvers. In this paper, we present a proof-of-principle *sublinear-time* algorithm for solving SDPs with low-rank constraints; specifically, given an SDP with m constraint matrices, each of dimension n and rank r , our algorithm can compute any entry and efficient descriptions of the spectral decomposition of the solution matrix. The algorithm runs in time $O(m \cdot \text{poly}(\log n, r, 1/\varepsilon))$ given access to a sampling-based low-overhead data structure for the constraint matrices, where ε is the precision of the solution. In addition, we apply our algorithm to a quantum state learning task as an application.

Technically, our approach aligns with 1) SDP solvers based on the matrix multiplicative weight (MMW) framework by Arora and Kale [TOC '12]; 2) sampling-based dequantizing framework pioneered by Tang [STOC '19]. In order to compute the matrix exponential required in the MMW framework, we introduce two new techniques that may be of independent interest:

- Weighted sampling: assuming sampling access to each individual constraint matrix A_1, \dots, A_τ , we propose a procedure that gives a good approximation of $A = A_1 + \dots + A_\tau$.
- Symmetric approximation: we propose a sampling procedure that gives the *spectral decomposition* of a low-rank Hermitian matrix A . To the best of our knowledge, this is the first sampling-based algorithm for spectral decomposition, as previous works only give singular values and vectors.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Spectral decomposition, Semi-definite programming, Quantum-inspired algorithm, Sublinear algorithm

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.23

Related Version The full version of the paper is available at <https://arxiv.org/abs/1901.03254>.

Funding NHC, HHL, and CW were supported by Scott Aaronson's Vannevar Bush Faculty Fellowship.

Tongyang Li: TL was supported by IBM PhD Fellowship, QISE-NET Triplet Award (NSF DMR-1747426), and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Quantum Algorithms Teams program.

Acknowledgements We thank Scott Aaronson, András Gilyén, Ewin Tang, Ronald de Wolf, and anonymous reviewers for their detailed feedback on preliminary versions of this paper.



© Nai-Hui Chia, Tongyang Li, Han-Hsuan Lin, and Chunhao Wang;
licensed under Creative Commons License CC-BY

45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020).

Editors: Javier Esparza and Daniel Král'; Article No. 23; pp. 23:1–23:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Semidefinite programming (SDP) is a central topic in the studies of mathematical optimization and theoretical computer science, with a wide range of applications including algorithm design, machine learning, operations research, etc. The importance of SDP comes from both its generality that contains the better-known linear programming (LP) and the fact that it admits polynomial-time solvers. Mathematically, an SDP is defined as follows:

$$\max \quad \text{Tr}[CX] \tag{1}$$

$$\text{s.t.} \quad \text{Tr}[A_i X] \leq b_i \quad \forall i \in [m]; \tag{2}$$

$$X \succeq 0, \tag{3}$$

where m is the number of constraints, A_1, \dots, A_m, C are $n \times n$ Hermitian matrices, and $b_1, \dots, b_m \in \mathbb{R}$; Eq. (3) restricts the variable matrix X to be *positive semidefinite* (PSD), i.e., X is an $n \times n$ Hermitian matrix with non-negative eigenvalues (more generally, $X \succeq Y$ means that $X - Y$ is a PSD matrix). An ε -approximate solution of this SDP is an X^* that satisfies Eqns. (2) and (3) while $\text{Tr}[CX^*] \geq \text{OPT} - \varepsilon$ (OPT being the optimum of the SDP).

There is rich literature on solving SDPs. Ellipsoid method gave the first polynomial-time SDP solvers [29, 19], and the complexities of the SDP solvers had been subsequently improved by the interior-point method [34] and the cutting-plane method [3, 33]; see also the survey paper [38]. The current state-of-the-art SDP solver [31, 24] runs in time $\tilde{O}(m(m^2 + n^\omega + mn^2)\text{poly}(\log 1/\varepsilon))$, where $\omega < 2.373$ is the exponent of matrix multiplication.¹ On the other hand, if we tolerate polynomial dependence in $1/\varepsilon$, Arora and Kale [7] gave an SDP solver with better complexities in m and n : $\tilde{O}(mn^2(R_p R_d/\varepsilon)^4 + n^2(R_p R_d/\varepsilon)^7)$, where R_p, R_d are given upper bounds on the ℓ_1 -norm of the optimal primal and dual solutions, respectively (see more details in [5]). This is subsequently improved to $\tilde{O}(m/\varepsilon^2 + n^2/\varepsilon^{2.5})$ by Garber and Hazan [16, 17] when $R_p, R_d = 1$ and $b_i = 0$ in Eq. (2) for all $i \in [m]$; as a complement, [16] also established a lower bound $\Omega(m/\varepsilon^2 + n^2/\varepsilon^2)$ under the same assumption.

The SDP solvers mentioned above all use the standard entry-wise access to matrices A_1, \dots, A_m , and C . In contrast, a common methodology in algorithm design is to assume a certain natural *preprocessed data structure* such that the problem can be solved in *sublinear* time, perhaps even in *poly-logarithmic* time, given queries to the preprocessed data structure (e.g., see the examples discussed in Section 1.3). Such methodology is extensively exploited in *quantum algorithms*, where we are given a unitary oracle to access entries of matrices in *superposition*, a fundamental feature in quantum mechanics and the essence of quantum speedups. In particular, quantum SDP solvers in the case that matrices are sparse have been studied in [10, 5, 9, 4] and culminate in a quantum algorithm that runs in time $\tilde{O}((\sqrt{m} + \sqrt{n}R_p R_d/\varepsilon)n(R_p R_d/\varepsilon)^4)$ [4], which achieve *polynomial* speedup comparing to existing classical algorithms in m and n . Based on an oracle that can prepare the quantum state corresponding to the positive semidefinite part of Hermitian matrices,² quantum *exponential* speedup in n has been achieved for matrices with rank $r = \text{poly}(\log n)$ by [9, 4], whose algorithms run in time $\tilde{O}(\sqrt{m}) \cdot \text{poly}(\log m, \log n, r, 1/\varepsilon)$. Considering this, SDP was previously believed to be a strong candidate for exponential quantum speedups in the low-rank setting (see e.g. [35]).

¹ Throughout the paper, $\tilde{O}(f(\cdot))$ denotes $O(f(\cdot)\text{polylog}(f(\cdot)))$.

² Ref. [4] called this the quantum state input model. Their complexity is expressed in terms of a parameter B , which is basically the trace norm of the constraint and cost matrices, which then is basically the rank for matrices with spectral norm 1.

Mutually inspired by both classical and quantum SDP solvers, and the series of “dequantization” results [36, 12] lead by Tang’s breakthrough result [37], in this work we strive to match the exponential speedup of [9, 4] with a classical algorithm, with the same low-rank requirement on constraint and cost matrices.

1.1 Main results

We show that when the constraint and cost matrices are low-rank, with a low-overhead data structure that supports the following sampling access, there exists a classical algorithm whose runtime is logarithmic in the dimension n of the matrices.

► **Definition 1** (Sampling and query access). *Let $M \in \mathbb{C}^{n \times n}$. Denote $\|\cdot\|$ to be the ℓ_2 norm and $\|\cdot\|_F$ to be the Frobenius norm. We say that we have the sampling access to M if we can*

1. *sample a row index $i \in [n]$ of M where the probability of row i being chosen is*

$$\frac{\|M(i, \cdot)\|^2}{\|M\|_F^2};$$

2. *for all $i \in [n]$, sample an index $j \in [n]$ where the probability of j being chosen is*

$$\frac{|M(i, j)|^2}{\|M(i, \cdot)\|^2};$$

3. *query the entry $M(i, j)$ for any $i, j \in [n]$; and*
4. *evaluate norms of $\|M\|_F$ and $\|M(i, \cdot)\|$ for $i \in [n]$,*

with time complexity $O(\text{poly}(\log n))$ for each sampling and norm access.

A low-overhead data structure that allows for this sampling access is shown in Section 2.2. Our main result is as follows.

► **Theorem 2** (informal; see Theorems 6 and 12 and Algorithm 7). *Let $C, A_1, \dots, A_m \in \mathbb{C}^{n \times n}$ be an SDP instance as in Eqns. (1) to (3). Suppose $\text{rank}(C), \max_{i \in [m]} \text{rank}(A_i) \leq r$. Given sampling access to A_1, \dots, A_m, C in Definition 1, there is an algorithm that gives any specific entry of an ε -approximate solution of the SDP with probability at least $2/3$; the algorithm runs in time $O(m \cdot \text{poly}(\log n, r, R_p R_d / \varepsilon))$, where R_p, R_d are given upper bounds on the ℓ_1 -norm of the optimal primal and dual solutions.*

Comparing our results to existing classical randomized algorithms for solving SDP (e.g., [7, 16, 17]), our algorithm outperforms existing classical SDP solvers given sampling access to the constraint matrices (which can be realized with a low-overhead data structure). Specifically, the running time of our algorithm is $O(m \cdot \text{poly}(\log n, r, R_p R_d / \varepsilon))$ according to Theorem 2, which achieves exponential speedup in terms of n with the data structure given in Theorem 5. It is worth noting that there are other ways to implement the sampling and query access. For example, Drineas, Kannan, and Mahoney [14, Lemma 2] showed that the sampling access in Definition 1 can be achieved with poly-logarithmic space if the matrix elements are streamed. Therefore, Theorem 2 also implies that there exists a one-pass poly-logarithmic space algorithm for low-rank SDP in the data-streaming model.

Compared to quantum algorithms, our algorithm has comparable running time. It is because existing quantum SDP solvers that achieve exponential speed up in terms of n all basically have polynomial dependence on the rank r [9, 4], so they also have $\text{poly}(\log n, r)$ complexity. It is worth noting that the quantum SDP solvers require additional assumptions

on the way the matrices are given. Furthermore, we give query access to the solution matrix which was not achieved by existing quantum SDP solvers, as they only give sampling access to the solution matrix. In this regard, it is easy to obtain the sampling access of the solution matrix from our algorithm by extending the rejection sampling techniques of [37] as pointed out by Tang³.

Our result aligns with the studies of sampling-based algorithms for solving linear algebraic problems. In particular, [15] gave low-rank approximations of a matrix in *poly-logarithmic* time with sampling access to the matrix as in Definition 1. Recently, Tang extended the idea of [15] to give a poly-logarithmic time algorithm for solving recommendation systems [37]. Subsequently, still under the same sampling assumption, Tang [36] sketched poly-logarithmic algorithms for principal component analysis and clustering assignments, and two follow-up papers [18, 12] gave poly-logarithmic algorithms for solving low-rank linear systems. All these sampling-based sublinear algorithms directly exploit the sampling approach in [15] (see Section 1.2 for details); to solve SDPs, we derive an augmented sampling toolbox which includes two novel techniques: *weighted sampling* and *symmetric approximation*.

As a corollary, our SDP solver can be applied to learning quantum states⁴ efficiently. A particular task of learning quantum states is *shadow tomography* [1], where we are asked to find a description of an unknown quantum state ρ such that we can approximate $\text{Tr}[\rho E_i]$ up to error ϵ for a specific collection of Hermitian matrices E_1, \dots, E_m where $0 \preceq E_i \preceq I$ and $E_i \in \mathbb{C}^{n \times n}$ for all $i \in [m]$ (such E_i can also be viewed as a measurement operator in a two-outcome POVM in quantum computing). Mathematically, shadow tomography can be formulated as the following SDP feasibility problem:

$$\text{Find } \sigma \text{ such that } |\text{Tr}[\sigma E_i] - \text{Tr}[\rho E_i]| \leq \epsilon \quad \forall i \in [m]; \quad (4)$$

$$\sigma \succeq 0, \quad \text{Tr}[\sigma] = 1. \quad (5)$$

Under a quantum model proposed by [9] where ρ, E_1, \dots, E_m are stored as quantum states, the state-of-the-art quantum algorithm [4] solves shadow tomography with time $O((\sqrt{m} + \min\{\sqrt{n}/\epsilon, r^{2.5}/\epsilon^{3.5}\})r/\epsilon^4)$ where $r = \max_{i \in [m]} \text{rank}(E_i)$; in other words, quantum algorithms achieve poly-logarithmic complexity in n for low-rank shadow tomography. We observe the same phenomenon under our sampling-based model:

► **Corollary 3** (informal; see Corollary 15). *Given sampling access of matrices $E_1, \dots, E_m \in \mathbb{C}^{n \times n}$ as in Definition 1 and real numbers $\text{Tr}[\rho E_1], \dots, \text{Tr}[\rho E_m]$, there is an algorithm that gives a succinct description as in Remark 13 and any entry of an ϵ -approximate solution σ of the shadow tomography problem defined as Eqns. (4) and (5) with probability at least $2/3$; the algorithm runs in time $O(m \cdot \text{poly}(\log n, r, 1/\epsilon))$.*

1.2 Techniques

Matrix multiplicative weight method (MMW). We study a normalized SDP feasibility testing problem defined as follows:

³ Personal communication.

⁴ A quantum state ρ is a PSD matrix with trace one.

► **Definition 4** (Feasibility of SDP). *Given an $\epsilon > 0$, m real numbers $a_1, \dots, a_m \in \mathbb{R}$, and Hermitian $n \times n$ matrices A_1, \dots, A_m where $-I \preceq A_i \preceq I, \forall i \in [m]$, define \mathcal{S}_ϵ as the set of all X such that*

$$\text{Tr}[A_i X] \leq a_i + \epsilon \quad \forall i \in [m]; \quad (6)$$

$$X \succeq 0; \quad (7)$$

$$\text{Tr}[X] = 1. \quad (8)$$

For ϵ -approximate feasibility testing of the SDP, we require that:

■ If $\mathcal{S}_\epsilon = \emptyset$, output “infeasible”;

■ If $\mathcal{S}_0 \neq \emptyset$, output an $X \in \mathcal{S}_\epsilon$.⁵

It is well-known that one can use binary search to reduce ϵ -approximation of the SDP in Eqns. (1) to (3) to $O(\log(R_p R_d / \epsilon))$ calls of the feasibility problem in Definition 4 with $\epsilon = \epsilon / (R_p R_d)$.⁶ Therefore, in this paper we focus on solving feasibility testing of SDPs.

To solve the feasibility testing problem in Definition 4, we follow the *matrix multiplicative weight* (MMW) framework [6]. To be more specific, we follow the approach of regarding MMW as a zero-sum game with two players (see, e.g., [21, 39, 20, 30, 9]), where the first player wants to provide a feasible $X \in \mathcal{S}_\epsilon$, and the second player wants to find any violation $j \in [m]$ of any proposed X , i.e., $\text{Tr}[A_j X] > a_j + \epsilon$. At the t^{th} round of the game, if the second player points out a violation j_t for the current proposed solution X_t , the first player proposes a new solution

$$X_{t+1} \leftarrow \exp[-(A_{j_1} + \dots + A_{j_t})] \quad (9)$$

(up to normalization); such a solution by matrix exponentiation is formally named as a *Gibbs state*. A feasible solution is actually an equilibrium point of the zero-sum game, which can also be approximated by the MMW method [6]; formal discussions are given in Section 2.3.

Improved sampling tools. Before describing our improved sampling tools, let us give a brief review of the techniques introduced by [15]. The basic idea of [15] comes from the fact that a low-rank matrix A can be well-approximated by sampling a small number of rows. More precisely, suppose that A is an $n \times n$ matrix with rank r , where $n \gg r$. Because n is large, it is preferable to obtain a “description” of A without using $\text{poly}(n)$ resources. If we have the sampling access to A in the form of Definition 1, we can sample rows from A according to statement 1 of Definition 1. Suppose S is the $p \times n$ submatrix of A formed by sampling $p = \text{poly}(r)$ rows from A with some normalization. It can be shown that $S^\dagger S \approx A^\dagger A$ in the Frobenius norm. Furthermore, we can apply the similar sampling techniques to sampling p columns of S with some normalization to form a $p \times p$ matrix W such that $WW^\dagger \approx SS^\dagger$. Then the singular values and singular vectors of W , which are easy to compute because p is small, together with the row indices that form S , can be viewed as a succinct description of some matrix $V \in \mathbb{C}^{n \times r}$ satisfying $A \approx AVV^\dagger$, which gives a low-rank projection of A . In [12], this method was extended to approximating the spectral decomposition of AA^\dagger , i.e., calculating a small diagonal matrix D and finding a succinct description of V such that $VD^2V^\dagger \approx AA^\dagger$.

⁵ If $\mathcal{S}_\epsilon \neq \emptyset$ and $\mathcal{S}_0 = \emptyset$, either output is acceptable.

⁶ For the normalized case $R_p R_d = 1$, we first guess a candidate value $c_1 = 0$ for the objective function, and add that as a constraint $\text{Tr}[CX] \geq c_1$ to the optimization problem. If this problem is feasible, the optimum is larger than c_1 and we accordingly take $c_2 = c_1 + \frac{1}{2}$; if this problem is infeasible, the optimum is smaller than c_1 and we accordingly take $c_2 = c_1 - \frac{1}{2}$; we proceed similarly for all c_i . As a result, we could solve the optimization problem with precision ϵ using $\lceil \log_2 \frac{1}{\epsilon} \rceil$ calls to the feasibility problem in Definition 4. For renormalization, it suffices to take $\epsilon = \epsilon / (R_p R_d)$. See also [9].

To implement the MMW framework, we need an approximate description of the matrix exponentiation $X_{t+1} := \exp[-\sum_{\tau=1}^t A_{j_\tau}]$ in Eq. (9). We achieve this in two steps. First, we get an approximate description of the spectral decomposition of $A := \sum_{\tau=1}^t A_{j_\tau}$ as $A \approx \hat{V} \hat{D} \hat{V}^\dagger$, where \hat{V} is an $n \times r$ matrix and \hat{D} is an $r \times r$ real diagonal matrix. Then, we approximate the matrix exponentiation e^{-A} by $\hat{V} e^{-\hat{D}} \hat{V}^\dagger$.

There are two main technical difficulties that we overcome with new tools while following the above strategy. First, since A changes dynamically throughout the MMW method, we cannot assume the sampling access to A ; a more reasonable assumption is to have sampling access to each individual constraint matrix A_{j_τ} , but it is hard to directly sample from A by sampling from each individual A_{j_τ} .⁷ In Section 3.1, we sidestep this difficulty by devising the *weighted sampling* procedure which gives a succinct description of a low-rank approximation of $A = \sum_{\tau} A_{j_\tau}$ by sampling each individual A_{j_τ} . In other words, we cannot sample according to A , but we can still find a succinct description of a low-rank approximation of A .

Second, the original sampling procedure of [15] and the extension by [12] give $VD^2V^\dagger \approx A^\dagger A$ instead of a spectral decomposition $\hat{V} \hat{D} \hat{V}^\dagger \approx A$, even if A is Hermitian. For our purpose of matrix exponentiation, singular value decomposition is problematic because the singular values ignore the signs of the eigenvalues; specifically, we get a large error if we approximate e^{-A} by naively exponentiating the singular value decomposition: $e^{-A} \not\approx V e^{-D} V^\dagger$. Note that this issue of missing negative signs is intrinsic to the tools in [15] because they are built upon the approximation $S^\dagger S \approx A^\dagger A$; Suppose that A has the decomposition $A = UDV^\dagger$, where D is a diagonal matrix, and U and V are isometries. Then $A^\dagger A = VD^\dagger DV^\dagger$, cancelling out any phase on D . We resolve this issue by a novel approximation procedure, *symmetric approximation*. Symmetric approximation is based on the result $A \approx AVV^\dagger$ shown by [15]. It then holds that $A \approx V(V^\dagger AV)V^\dagger$ because the symmetry of A implies that VV^\dagger acts roughly as the identity on the image of A . Since $(V^\dagger AV)$ is a small matrix of size $r \times r$, we can calculate it explicitly and diagonalize it, getting approximate eigenvalues of A . Together with the description of V , we get the desired description of the spectral decomposition of A . See Section 3.2 for more details.

1.3 Related work

As we have mentioned earlier, many SDP solvers use cutting-plane methods or interior-point methods with complexity $\text{poly}(\log(1/\epsilon))$ but larger complexities in m and n . In contrast, our SDP solver follows the MMW framework, and we briefly summarize such SDP solvers in existing literature. They mainly fall into two categories as follows.

First, MMW is adopted in solvers for *positive* SDPs, i.e., $A_1, \dots, A_m, C \succeq 0$. In this case, the power of MMW lies in its efficiency of having only $\tilde{O}(\text{poly}(1/\epsilon))$ iterations (i.e., poly-logarithmic in m, n) and the fact that it admits *width-independent* solvers whose complexities are independent of R_p and R_d . Ref. [32] first gave a width-independent positive LP solver that runs in $O(\log^2(mn)/\epsilon^4)$ iterations, and [22] subsequently generalized this result to give the first width-independent positive SDP solver. The state-of-the-art positive SDP solver was proposed by [2] with only $O(\log^2(mn)/\epsilon^3)$ iterations.

Second, as far as we know, the vast majority of *quantum* SDP solvers use the MMW framework. The first quantum SDP solver was proposed by [10] with worst-case running time $\tilde{O}(\sqrt{mns^2}(R_p R_d/\epsilon)^{32})$, where s is the sparsity of input matrices, i.e., every row or column

⁷ For example, assume we have $A = A_1 + A_2$ such that $A_2 = -A_1 + \epsilon$, where ϵ is a matrix with small entries. In this case, A_1 and A_2 mostly cancel out each other and leave $A = \epsilon$. Since ϵ can be arbitrarily small compared to A_1 and A_2 , it is hard to sample from ϵ by sampling from A_1 and A_2 .

of A_1, \dots, A_m, C has at most s nonzero elements. Subsequently, the quantum complexity of solving SDPs was improved by [5, 9], and the state-of-the-art quantum SDP solver runs in time $\tilde{O}((\sqrt{m} + \sqrt{n}R_p R_d/\epsilon)s(R_p R_d/\epsilon)^4)$ [4]. This is optimal in the dependence of m and n because [10] proved a quantum lower bound of $\Omega(\sqrt{m} + \sqrt{n})$ for constant R_p, R_d, s , and ϵ .

The authors, along with Gilyén and Tang [11], further generalized the techniques to singular value transformation⁸ and proposed a quantum-inspired framework to dequantize almost all known quantum machine learning algorithms with claimed exponential speedups. The low-rank SDP problem also fits in that framework and a better time complexity has been achieved in [11] due to a more efficient sampling method. However, the results in this paper remain its own interest: our techniques are specially crafted for approximating matrices in the form of $e^{-\epsilon A}$ and hence provide additional insights for this line of research.

1.4 Open questions

Our paper raises a few natural open questions for future work. For example:

- Can we give faster sampling-based algorithms for solving LPs? Note that recent breakthroughs by [13, 25] solve LPs with complexity $\tilde{O}(n^\omega)$ where $\omega \approx 2.373$, significantly faster than the state-of-the-art SDP solver [31] with complexity $\tilde{O}(m(m^2 + n^\omega + mn^2))$.
- What is the empirical performance of our sampling-based method? Admittedly, the exponents of our poly-logarithmic factors are large; nevertheless, it is common that numerical experiments perform better than theoretical guarantees, and we wonder if this phenomenon can be observed when applying our method.

2 Preliminaries

2.1 Notations

Throughout the paper, we denote by m and n the number of constraints and the dimension of constraint matrices in SDPs, respectively. We use ϵ to denote the precision of the solution of the SDP feasibility problem in Eq. (6) of Definition 4. We use r to denote an upper bound on the rank of matrices, i.e., $\max_{i \in [n]} \{\text{rank}(A_i), \text{rank}(C)\} \leq r$ (we denote $[n] := \{1, \dots, n\}$).

For a vector $v \in \mathbb{C}^n$, we use \mathcal{D}_v to denote the probability distribution on $[n]$ where the probability of i being chosen is $\mathcal{D}_v(i) = |v(i)|^2 / \|v\|^2$ for all $i \in [n]$. When it is clear from the context, a sample from \mathcal{D}_v is often referred to as a sample from v . For a matrix $A \in \mathbb{C}^{n \times n}$, we use $\|A\|$ and $\|A\|_F$ to denote its spectral norm and Frobenius norm, respectively; we use $A(i, \cdot)$ and $A(\cdot, j)$ to denote the i^{th} row and j^{th} column of A , respectively. We use $\text{rows}(A)$ to denote the n -dimensional vector formed by the norms of its row vectors, i.e., $\text{rows}(A)(i) = \|A(i, \cdot)\|$, for all $i \in [n]$.

2.2 Data structure for sampling and query access

As we develop sublinear-time algorithms for solving SDP in this paper, the whole constraint matrices cannot be loaded into memory since storing them requires at least linear space and time. Instead, we assume the *sampling access* of each constraint matrix as defined in Definition 1. This sampling access relies on a natural probability distribution that arises in many machine learning applications [15, 12, 18, 27, 28, 37, 36] (also see a survey by Kannan and Vempala [26]).

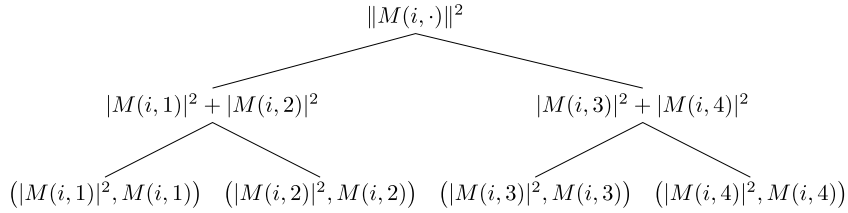
⁸ Similar results were simultaneously obtained by Jethwani, Le Gall, and Singh [23].

Technically, Ref. [15] used this sampling assumption to develop a sublinear algorithm for approximating low-rank projection of matrices. It is well-known (as pointed out by [27] and also used in [12, 18, 28, 37, 36]) that there exist low-overhead preprocessed data structures that allow for the sampling access. More precisely, the existence of the data structures for the sampling access defined in Definition 1 is stated as follows.

► **Theorem 5** ([27]). *Given a matrix $M \in \mathbb{C}^{n \times n}$ with s non-zero entries, there exists a data structure storing M in space $O(s \log n)$, which supports the following:*

1. *Read and write $M(i, j)$ in $O(\log n)$ time.*
2. *Evaluate $\|M(i, \cdot)\|$ in $O(\log n)$ time.*
3. *Evaluate $\|M\|_F^2$ in $O(1)$ time.*
4. *Sample a row index of M according to statement 1 of Definition 1 in $O(\log n)$ time.*
5. *For each row, sample an index according to statement 2 of Definition 1 in $O(\log n)$ time.*

Readers may refer to [27, Theorem A.1] for the proof of Theorem 5. In the following, we give the intuition of the data structure, which is demonstrated in Figure 1. We show how to sample from a row vector: we use a binary tree to store the data of each row. The square of the absolute values of all entries, along with their original values are stored in the leaf nodes. Each internode contains the sum of the values of its two immediate children. It is easy to see that the root node contains the square of the norm of this row vector. To sample an index and to query an entry from this row, logarithmic steps suffice. To fulfill statement 1 of Definition 1, we treat the norms of rows as a vector $(\|M(1, \cdot)\|, \dots, \|M(n, \cdot)\|)$ and organize the data of this vector in a binary tree.



■ **Figure 1** Illustration of a data structure that allows for sampling access to a row of $M \in \mathbb{C}^{4 \times 4}$.

2.3 Feasibility testing of SDPs

We adopt the MMW framework to solve SDPs under the zero-sum approach [21, 39, 20, 30, 9]. This is formulated as the following theorem:

► **Theorem 6** (Master algorithm). *Feasibility of the SDP in Eqns. (6) to (8) can be tested by Algorithm 1.*

This theorem is proved in [9, Theorem 2.3]; note that the weight matrix therein is $W_{t+1} = \exp[\frac{\epsilon}{2} \sum_{i=1}^t M_i]$ where $M_i = \frac{1}{2}(I_n - A_{j_i})$, and this gives the same Gibbs state as in Line 1 since for any Hermitian matrix $W \in \mathbb{C}^{n \times n}$ and real number $c \in \mathbb{R}$,

$$\frac{e^{W+cI}}{\text{Tr}[e^{W+cI}]} = \frac{e^W e^{cI}}{\text{Tr}[e^W e^{cI}]} = \frac{e^W}{\text{Tr}[e^W]}. \quad (10)$$

It should also be understood that this master algorithm is *not* the final algorithm; the step of trace estimation with respect to the Gibbs state (in Line 1 and Line 1) will be fulfilled by our sampling-based approach.

■ **Algorithm 1** MMW for testing feasibility of SDPs.

```

1 Set the initial Gibbs state  $\rho_1 = \frac{I_n}{n}$ , and number of iterations  $T = \frac{16 \ln n}{\epsilon^2}$ ;
2 for  $t = 1, \dots, T$  do
3   Find a  $j_t \in [m]$  such that  $\text{Tr}[A_{j_t} \rho_t] > a_{j_t} + \epsilon$ . If we cannot find such  $j_t$ , claim
   that  $\rho_t \in \mathcal{S}_\epsilon$  (the SDP is feasible) and terminate the algorithm;
4   Define the new weight matrix  $W_{t+1} := \exp[-\frac{\epsilon}{4} \sum_{i=1}^t A_{j_i}]$  and Gibbs state
    $\rho_{t+1} := \frac{W_{t+1}}{\text{Tr}[W_{t+1}]}$ ;
5 end
6 Claim that the SDP is infeasible and terminate the algorithm;
```

3 The main algorithm

Our main algorithm is Algorithm 7, it depends on several building blocks. To begin with, we introduce a useful claim from [18, Lemma 11].

▷ **Claim 7 (Trace inner product estimation [18]).** Let $A \in \mathbb{C}^{n \times n}$ and $B \in \mathbb{C}^{n \times n}$ be two Hermitian matrices. Given sampling and query access to A and query access to B . Then one can estimate $\text{Tr}[AB]$ with the additive error ϵ_s with probability at least $1 - \delta$ by using

$$O\left(\frac{\|A\|_F^2 \|B\|_F^2}{\epsilon_s^2} (Q(A) + Q(B) + S(A) + N(A)) \log \frac{1}{\delta}\right) \quad (11)$$

time and queries, where $Q(B)$ is the cost of query access to B , and $Q(A), S(A), N(A)$ are the cost for query access, sampling access and row norm access to A .

Due to space constraints, the proofs of the theorems and lemmas in this section can be found in the full version of this paper.

3.1 Weighted sampling

Given sample and query access to A_1, \dots, A_τ , the objective of this section is to provide an algorithm to give sample and query access to a matrix V , which approximates the eigenvectors of $A := A_1 + \dots + A_\tau$. Specifically, we will show that $\|VV^\dagger AVV^\dagger - A\|_F$ can be bounded.

Note that trivially invoking the standard FKV sampling method [15] is not capable of this task. In this paper, we propose the *weighted sampling* method. The intuition is to assign each A_ℓ a different weight when computing the probability distribution, and then sampling a row/column index of A according to this probability distribution. The main theorem we prove in this section is as follows.

We first give the method for sampling row indices of A in Procedure 2. The objective of this procedure is to sample a submatrix S such that $S^\dagger S \approx A^\dagger A$.

After applying Procedure 2, we obtain the row indices i_1, \dots, i_p . Let S_1, \dots, S_τ be matrices such that $S_\ell(t, \cdot) = A_\ell(i_t, \cdot) / \sqrt{p P_{i_t}'} for all $t \in [p]$ and $\ell \in [\tau]$. Define S as$

$$S = S_1 + \dots + S_\tau. \quad (12)$$

Next, we give the method for sampling column indices of S as in Procedure 3: we need to sample a submatrix W from S such that $WW^\dagger \approx SS^\dagger$.

Now, we obtained column indices j_1, \dots, j_p . Let W_1, \dots, W_τ be matrices such that $W_\ell(\cdot, t) = S_\ell(\cdot, j_t) / \sqrt{p P_{j_t}'} for all $t \in [p], \ell \in [\tau]$, where $P_j' = \frac{1}{p} \sum_{t=1}^p Q_{j|i_t}$ for $j \in [n]$. Define W as$

$$W = W_1 + \dots + W_\tau. \quad (13)$$

Procedure 2 Weighted sampling of rows.

Input: The sampling and query access to each A_ℓ as in Definition 1 for $A = \sum_{\ell=1}^{\tau} A_\ell$; integer p .

- 1 Sample p indices i_1, \dots, i_p from $[n]$ according to the probability distribution $\{P_1, \dots, P_n\}$ where $P_i = \sum_{j=1}^{\tau} \mathcal{D}_{\text{rows}(A_j)}(i) \|A_j\|_F^2 / \left(\sum_{\ell=1}^{\tau} \|A_\ell\|_F^2 \right)$ by sampling j according to $\|A_j\|_F^2 / \left(\sum_{\ell=1}^{\tau} \|A_\ell\|_F^2 \right)$ then sample according to $\mathcal{D}_{\text{rows}(A_j)}$;

Procedure 3 Weighted sampling of columns.

Input: The sampling and query access to each A_ℓ as in Definition 1 for $A = \sum_{\ell=1}^{\tau} A_\ell$; i_1, \dots, i_p obtained in Procedure 2; integer p .

- 1 Do the following p times independently to obtain samples j_1, \dots, j_p . **begin**
- 2 Sample a row index $t \in [p]$ uniformly at random;
- 3 Sample a column index $j \in [n]$ from the probability distribution $\{Q_{1|i_t}, \dots, Q_{n|i_t}\}$ where $Q_{j|i_t} = \sum_{k=1}^{\tau} \mathcal{D}_{A_k(i_t, \cdot)}(j) \|A_k(i_t, \cdot)\|^2 / \left(\sum_{\ell=1}^{\tau} \|A_\ell(i_t, \cdot)\|^2 \right)$;
- 4 **end**

With the weighted sampling method, we obtained a small submatrix W from A . Now, we use the singular values and singular vectors of W to approximate the ones of A . This is shown in Algorithm 4. The main consequence of Algorithm 4 is summarized in Theorem 8.

► **Theorem 8.** *Let $A = A_1 + \dots + A_\tau \in \mathbb{C}^{n \times n}$ be a Hermitian matrix where $A_\ell \in \mathbb{C}^{n \times n}$ is Hermitian, $\|A_\ell\| \leq 1$, and $\text{rank}(A_\ell) \leq r$ for all $\ell \in [\tau]$. The sampling and query access to each A_ℓ is given as in Definition 1. Take the error parameter ϵ as the input of Algorithm 4 to obtain the singular values $\sigma_1, \dots, \sigma_{\tilde{r}} \in \mathbb{R}$ and singular vectors $u_1, \dots, u_{\tilde{r}} \in \mathbb{C}^p$ for p specified in Line 4 of Algorithm 4. Let $V \in \mathbb{C}^{n \times \tilde{r}}$ be the matrix such that $V(\cdot, j) = \frac{S^\dagger}{\sigma_j} u_j$ for $j \in \{1, \dots, \tilde{r}\}$, where S is defined in Line 4 in Algorithm 4. Then with probability at least $9/10$, it holds that $\|VV^\dagger AVV^\dagger - A\|_F \leq \frac{\epsilon}{300r^2} (2 + \frac{\epsilon}{300r^2(\tau+1)})$.*

3.2 Symmetric approximation of low-rank Hermitian matrices

Now we show that the spectral decomposition of the sum of low-rank Hermitian matrices can be approximated in time logarithmic in the dimension with the given data structure. We call this technique *symmetric approximation*.

Briefly speaking, suppose we are given the approximated left singular vectors V of A outputted by Algorithm 4 such that $\|VV^\dagger AVV^\dagger - A\|_F$ is bounded as in Theorem 8, then we can approximately do spectral decomposition of A as follows. First, we approximate the matrix $V^\dagger AV$ by sampling. Then, since $V^\dagger AV$ is a matrix with low dimension, we can do spectral decomposition of the matrix efficiently as UDU^\dagger (see Lemma 9). Finally, we show that VU is close to an isometry. Therefore, $(VU)D(VU)^\dagger$ is an approximation to the spectral decomposition of A .

► **Lemma 9.** *Let $V \in \mathbb{C}^{n \times \tilde{r}}$ and $A = \sum_{\ell=1}^{\tau} A_\ell \in \mathbb{C}^{n \times n}$ be a Hermitian matrix. Given query access and sampling access to A_ℓ for $\ell \in [\tau]$, where each query and sample take $O(\log n)$ time, and query access to V , where each query takes $O(p)$ time. Let r be some integer such that $r \geq \tilde{r}$, and let $\epsilon_s = \frac{\epsilon}{400r^2}$. Then, one can output a Hermitian matrix $\tilde{B} \in \mathbb{C}^{\tilde{r} \times \tilde{r}}$ such that $\|V^\dagger AV - \tilde{B}\|_F \leq \epsilon_s$ with probability $1 - \delta$ by using $O\left((p + \log n) \frac{r^5 \tau^3}{\epsilon^2} \log \frac{1}{\delta}\right)$ time.*

■ **Algorithm 4** Approximation of singular vectors.

-
- Input:** The sampling and query access to each A_ℓ as in Definition 1 for $A = \sum_{\ell=1}^{\tau} A_\ell$ with $\text{rank}(A_\ell) \leq r$; error parameter ϵ .
- 1 Set $p = 2 \cdot 10^{20} \frac{\tau^{12} r^{19}}{\epsilon^6}$, $\gamma = \frac{\epsilon^2}{3 \times 10^6 \tau^2 r^6}$;
 - 2 Use Procedure 2 to obtain row indices i_1, \dots, i_p ;
 - 3 Let S_1, \dots, S_τ be matrices such that $S_\ell(t, \cdot) = A_\ell(i_t, \cdot) / \sqrt{p P_{i_t}}$ for all $t \in [p]$ and $\ell \in [\tau]$, where P_i is defined in Line 2 in Procedure 2. Let $S = S_1 + \dots + S_\tau$;
 - 4 Use Procedure 3 to obtain column indices j_1, \dots, j_p ;
 - 5 Let W_1, \dots, W_τ be matrices such that $W_\ell(\cdot, t) = S_\ell(\cdot, j_t) / \sqrt{p P'_{j_t}}$ for all $t \in [p]$ and $\ell \in [\tau]$, where $P'_j = \frac{1}{p} \sum_{t=1}^p Q_{j|i_t}$ for $j \in [n]$ and $Q_{j|i}$ is defined in Line 3 in Procedure 3. Let $W = W_1 + \dots + W_\tau$;
 - 6 (Assume the rank of A is \hat{r} .) Compute the top \hat{r} singular values $\sigma_1, \dots, \sigma_{\hat{r}}$ of W and their corresponding left singular vectors $u_1, \dots, u_{\hat{r}}$;
 - 7 Discard the singular values and their corresponding singular vectors satisfying $\sigma_j^2 < \gamma \sum_{\ell=1}^{\tau} \|W_\ell\|_F^2$. Let the remaining number of singular values be \tilde{r} ;
 - 8 Output i_1, \dots, i_p , P_{i_1}, \dots, P_{i_p} , $\sigma_1, \dots, \sigma_{\tilde{r}}$ and $u_1, \dots, u_{\tilde{r}}$;
-

The algorithm for approximating the spectral decomposition of A is described in Algorithm 5, and the effectiveness of Algorithm 5 is summarized in Lemma 10:

■ **Algorithm 5** Approximation of the spectral decomposition of A .

-
- Input:** The sampling and query access to each A_ℓ as in Definition 1 for $A = \sum_{\ell=1}^{\tau} A_\ell$ and query access to $V \in \mathbb{C}^{n \times \tilde{r}}$ (obtained from Algorithm 4, also see Theorem 8); error parameter ϵ .
- 1 Compute a matrix $\tilde{B} \in \mathbb{C}^{\tilde{r} \times \tilde{r}}$ whose spectrum approximates that of A (this is achieved by Lemma 9);
 - 2 Compute the spectral decomposition UDU^\dagger of matrix \tilde{B} ;
 - 3 Output an isometry $U \in \mathbb{C}^{\tilde{r} \times \tilde{r}}$ and a diagonal matrix $D \in \mathbb{C}^{\tilde{r} \times \tilde{r}}$ such that UDU^\dagger is the spectral decomposition of \tilde{B} .
-

► **Lemma 10.** *Algorithm 5 outputs a Hermitian matrix $\tilde{B} \in \mathbb{C}^{\tilde{r} \times \tilde{r}}$ with probability at least $1 - \delta$ with time and query complexity $O((p + \log n) \frac{\tau^5 r^3}{\epsilon^2} \log \frac{1}{\delta})$ such that*

$$\|V \tilde{B} V^\dagger - A\| \leq \left(1 + \frac{\epsilon}{300r^2(\tau+1)}\right)^2 \frac{\epsilon}{400r^2} + \left(2 + \frac{\epsilon}{300r^2(\tau+1)}\right) \frac{\epsilon}{300r^2}. \quad (14)$$

3.3 Approximating Gibbs states

In this subsection, we combine our techniques from Section 3.1 and Section 3.2 to give a sampling-based estimator of the traces of a Gibbs state times a constraint A_ℓ . This is formulated as Algorithm 6.

We show that the output of Algorithm 6 ϵ -approximates $\text{Tr}[A_\ell \rho]$ for $\rho = e^{-\frac{\epsilon}{2} A} / \text{Tr}[e^{-\frac{\epsilon}{2} A}]$. Let $\tilde{A} = VV^\dagger A VV^\dagger$. Let U and D be outputs of Algorithm 5, which will be used in Algorithm 6. We suppose $\|\tilde{A} - A\|_F \leq (2 + \frac{\epsilon}{300r^2(\tau+1)}) \frac{\epsilon}{300r^2}$ as in Theorem 8.

■ **Algorithm 6** Approximation of the trace.

Input: The sampling and query access to each A_ℓ as in Definition 1 for $A = \sum_{\ell=1}^r A_\ell$; query access to $V \in \mathbb{C}^{n \times \tilde{r}}$ (obtained from Algorithm 4, also see Theorem 8); matrices U and D (obtained from Algorithm 5) such that $(VU)D(VU)^\dagger$ is an approximated spectral decomposition of A as in Lemma 10.

- 1 Compute $\eta = \text{Tr}[e^{-\frac{\epsilon}{2}D}]$;
- 2 Approximate $\text{Tr}[A_\ell(VU)(e^{-\frac{\epsilon}{2}D}/\text{Tr}[e^{-\frac{\epsilon}{2}D}]) (VU)^\dagger]$ by ζ according to Claim 7;
- 3 Output ζ, η .

► **Lemma 11.** Let $\rho = \frac{e^{-\frac{\epsilon}{2}A}}{\text{Tr}[e^{-\frac{\epsilon}{2}A}]}$ and $\hat{\rho} = \frac{(VU)e^{-\frac{\epsilon}{2}D}(VU)^\dagger}{\text{Tr}[e^{-\frac{\epsilon}{2}D}]}$. Suppose $\|A - \tilde{A}\|_F \leq (2 + \frac{\epsilon}{300r^2(\tau+1)}) \frac{\epsilon}{300r^2}$. Let A_ℓ be a Hermitian matrix with the promise that $\|A_\ell\| \leq 1$ and $\text{rank}(A_\ell) \leq r$. Then Algorithm 6 outputs ζ such that

$$|\text{Tr}[A_\ell \rho] - \zeta| \leq \epsilon \quad (15)$$

with probability $1 - \delta$ in time $O(\frac{4}{\epsilon^2}(\log n + \tau pr) \log \frac{1}{\delta})$.

3.4 Proof of the main algorithm

We finally state our main result on solving SDPs via sampling.

► **Theorem 12.** Given Hermitian matrices $\{A_1, \dots, A_m\}$ with the promise that each of them has rank at most r , spectral norm at most 1, and the sampling access of each A_i is given by Definition 1. Also given $a_1, \dots, a_m \in \mathbb{R}$ and $\epsilon > 0$. Then Algorithm 7 gives a succinct description and any entry (see Remark 13) of the solution of the SDP feasibility problem

$$\text{Tr}[A_i X] \leq a_i + \epsilon \quad \forall i \in [m]; \quad X \succeq 0; \quad \text{Tr}[X] = 1 \quad (16)$$

with probability at least $2/3$ in $O(\frac{mr^{57} \ln^{37} n}{\epsilon^{92}})$ time.

■ **Algorithm 7** Feasibility testing of SDPs by our sampling-based approach.

- 1 Set the initial Gibbs state $\rho_1 = \frac{I_n}{n}$, and number of iterations $T = \frac{16 \ln n}{\epsilon^2}$;
- 2 **for** $t = 1, \dots, T$ **do**
- 3 Find a $j_t \in [m]$ such that $\text{Tr}[A_{j_t} \rho_t] > a_{j_t} + \epsilon$ using Algorithm 6. If we cannot find such j_t , claim that $\rho_t \in \mathcal{S}_\epsilon$ and terminate the algorithm. Output $\rho_t(\ell, j) = \sum_{k=1}^{\tilde{r}} V(\ell, k) e^{\sigma_k \epsilon / 2} V(j, k)^* / \eta$, where $V(\ell, j) = \sum_{s=1}^p \frac{A^*(i_s, \ell) u_j(t)}{\sqrt{P_{i_s} \sigma_j}}$, $i_1, \dots, i_p, P_{i_1}, \dots, P_{i_p}, \sigma_1, \dots, \sigma_{\tilde{r}}$ and $u_1, \dots, u_{\tilde{r}}$ are obtained from Algorithm 4 and η is obtained from Algorithm 6;
- 4 Define the new weight matrix $W_{t+1} := \exp[-\frac{\epsilon}{4} \sum_{i=1}^t A_{j_i}]$ and Gibbs state $\rho_{t+1} := \frac{W_{t+1}}{\text{Tr}[W_{t+1}]}$;
- 5 **end**
- 6 Claim that the SDP is infeasible and terminate the algorithm;

The algorithm follows the master algorithm in Theorem 6. The main challenge is to estimate $\text{Tr}[A_{j_t} \rho_t]$ where ρ_t is the Gibbs state at iteration t ; this is achieved by Lemma 11 in Section 3.3.

Proof. Correctness: The correctness of Algorithm 7 directly follows from Lemma 11. Specifically, we have shown that one can estimate the quantity $\text{Tr}[A_{j_t}\rho_t]$ with precision ϵ with high probability by applying Algorithms 4 to 6.

Time complexity: First, we show that given the data structure in Theorem 5, Algorithm 4 can be computed in time $O(p^3 + p\tau \log \tau \log n)$. Procedure 2 and Procedure 3 both can be done in time $O(p\tau \log \tau \log n)$. There are many ways to implement Procedure 2. For example, build a binary tree as in Theorem 5 for $\|A_1\|_F, \dots, \|A_\tau\|_F$ to sample $j \in [\tau]$ according to $\|A_j\|_F^2 / (\sum_{\ell=1}^{\tau} \|A_\ell\|_F^2)$, and then use the data structures in Theorem 5 to sample from $\mathcal{D}_{\text{rows}(A_j)}$. The time complexity is $O(\tau \log \tau \log n)$. Similarly, we can implement Procedure 3 in time $O(\tau \log \tau \log n)$. Hence, the time complexity to construct the matrix W and compute its SVD is $O(p\tau \log \tau \log n + p^3)$. Algorithm 4 succeeds with probability $9/10$.

Then, by Lemma 10 and Lemma 11, Algorithms 5 and 6 takes time $O((p + \log n)^{\frac{5}{\epsilon^2}} \log \frac{1}{\delta})$ and $O(\frac{4}{\epsilon^2} (\log n + \tau pr) \log \frac{1}{\delta})$ respectively to succeed with probability at least $1 - \delta$. Recall from the previous paragraph, Algorithm 4 takes time $O(p\tau \log \tau \log n + p^3)$. In total, Algorithms 4 to 6 are each called $Tm = \frac{16m \ln n}{\epsilon^2}$ times. We specify that

$$\blacksquare \quad \tau = T = \frac{16 \ln n}{\epsilon^2}, \text{ and}$$

$$\blacksquare \quad p = 2 \cdot 10^{20} \frac{\tau^{12} r^{19}}{\epsilon^6}$$

(see also Algorithm 4). By setting δ as a small enough constant (say $\delta = 1/6$) and noticing the p dominates other terms, Algorithm 7 succeeds with probability at least $2/3$ in time $O(Tmp^3) = O\left(\frac{mr^{57} \ln^{37} n}{\epsilon^{92}}\right)$. \blacktriangleleft

► **Remark 13.** Theorem 12 solves the SDP feasibility problem, i.e., to decide $\mathcal{S}_0 = \emptyset$ or $\mathcal{S}_\epsilon \neq \emptyset$. For the SDP optimization problem in Eqns. (1) to (3), an approximation to the optimal value can be found by a binary search with feasibility subroutines. (see Footnote 6); however, writing down the approximate solution would take n^2 space, ruining the poly-logarithmic complexity in n . Nevertheless,

- we have its *succinct representation* $i_1, \dots, i_p, P_{i_1}, \dots, P_{i_p}, \sigma_1, \dots, \sigma_{\tilde{r}}, u_1, \dots, u_{\tilde{r}}$, and η ;
- we can compute any entry of the solution matrix according to this succinct description as in Step 3 of Algorithm 7.

With the succinct description given in the first statement, one can perform other operations on the solution matrix using the similar sampling-based methods.

► **Remark 14.** The large polynomial overhead in Theorem 12 may not be necessary in practice and can potentially be reduced by more fine-grained analysis. This is also suggested by numerics in practice (see [8]).

Application to shadow tomography. As a direct corollary of Theorem 12, we have:

► **Corollary 15.** *Given Hermitian matrices $\{E_1, \dots, E_m\}$ with the promise that each of E_1, \dots, E_m has rank at most r , $0 \preceq E_i \preceq I$ and the sampling access to E_i is given as in Definition 1 for all $i \in [m]$. Also given $p_1, \dots, p_m \in \mathbb{R}$. Then for any $\epsilon > 0$, the shadow tomography problem*

$$\text{Find } \sigma \text{ such that } |\text{Tr}[\sigma E_i] - p_i| \leq \epsilon \quad \forall i \in [m] \quad \text{subject to } \sigma \succeq 0, \quad \text{Tr}[\sigma] = 1 \quad (17)$$

can be solved with probability $1 - \delta$ with cost $O(m \cdot \text{poly}(\log n, 1/\epsilon, \log(1/\delta), r))$.

► **Remark 16.** Similar to Remark 13, σ can be stored as a succinct representation because we can have $\sigma = \frac{\exp[\frac{\epsilon}{2} \sum_{\tau=1}^t (-1)^{i_\tau} A_{j_\tau}]}{\text{Tr}[\exp[\frac{\epsilon}{2} \sum_{\tau=1}^t (-1)^{i_\tau} A_{j_\tau}]]}$ in Corollary 15, where $t \leq T$ and $i_\tau \in \{0, 1\}$, $j_\tau \in [m]$ for all $\tau \in [t]$. Storing all i_τ, j_τ takes $t(\log_2 m + 1) = O(\log m \log n / \epsilon^2)$ bits.

References

- 1 Scott Aaronson. Shadow tomography of quantum states. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 325–338. ACM, 2018. [arXiv:1711.01053](#).
- 2 Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive sdp solver. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1824–1831. Society for Industrial and Applied Mathematics, 2016. [arXiv:1507.02259](#).
- 3 Kurt M. Anstreicher. The volumetric barrier for semidefinite programming. *Mathematics of Operations Research*, 25(3):365–380, 2000.
- 4 Joran van Apeldoorn and András Gilyén. Improvements in quantum SDP-solving with applications. In *Proceedings of 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 99:1–99:15, 2019. [arXiv:1804.05058](#).
- 5 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-solvers: Better upper and lower bounds. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2017)*. IEEE, 2017. [arXiv:1705.01843](#).
- 6 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.
- 7 Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 227–236. ACM, 2007.
- 8 Juan Miguel Arrazola, Alain Delgado, Bhaskar Roy Bardhan, and Seth Lloyd. Quantum-inspired algorithms in practice, 2019. [arXiv:1905.10415](#).
- 9 Fernando G. S. L. Brandão, Amir Kalev, Tongyang Li, Cedric Yen-Yu Lin, Krysta M. Svore, and Xiaodi Wu. Quantum SDP solvers: Large speed-ups, optimality, and applications to quantum learning. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, page 27. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. [arXiv:1710.02581](#).
- 10 Fernando G. S. L. Brandão and Krysta Svore. Quantum speed-ups for semidefinite programming. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2017)*. IEEE, 2017. [arXiv:1609.05537](#).
- 11 Nai-Hui Chia, András Gilyén, Tongyang Li, Han-Hsuan Lin, Ewin Tang, and Chunhao Wang. Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, page 387–400. ACM, 2020. [arXiv:1910.06151](#).
- 12 Nai-Hui Chia, Han-Hsuan Lin, and Chunhao Wang. Quantum-inspired sublinear classical algorithms for solving low-rank linear systems, 2018. [arXiv:1811.04852](#).
- 13 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC 2019)*, pages 938–942. ACM, 2019. [arXiv:1810.07896](#).
- 14 P. Drineas, R. Kannan, and M. Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.
- 15 Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004.
- 16 Dan Garber and Elad Hazan. Approximating semidefinite programs in sublinear time. In *Advances in Neural Information Processing Systems (NIPS 2011)*, pages 1080–1088, 2011.
- 17 Dan Garber and Elad Hazan. Almost optimal sublinear time algorithm for semidefinite programming, 2012. [arXiv:1208.5211](#).
- 18 András Gilyén, Seth Lloyd, and Ewin Tang. Quantum-inspired low-rank stochastic regression with logarithmic dependence on the dimension, 2018. [arXiv:1811.04909](#).

- 19 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.
- 20 Gus Gutoski and Xiaodi Wu. Parallel approximation of min-max problems with applications to classical and quantum zero-sum games. In *Proceedings of the 27th Annual IEEE Symposium on Computational Complexity (CCC 2012)*, pages 21–31. IEEE, 2012.
- 21 Elad Hazan. *Efficient algorithms for online convex optimization and their applications*. PhD thesis, Princeton University, 2006.
- 22 Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 463–471. IEEE, 2011. [arXiv:1104.2502](#).
- 23 Dhawal Jethwani, François Le Gall, and Sanjay K. Singh. Quantum-inspired classical algorithms for singular value transformation, 2019. [arXiv:1910.05699](#).
- 24 Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games and its applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, pages 944–953, 2020. [arXiv:2004.04250](#).
- 25 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster LPs, 2020. [arXiv:2004.07470](#).
- 26 Ravindran Kannan and Santosh Vempala. Randomized algorithms in numerical linear algebra. *Acta Numerica*, 26:95–135, 2017.
- 27 Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, pages 49:1–49:21, 2017. [arXiv:1603.08675](#).
- 28 Iordanis Kerenidis and Anupam Prakash. A quantum interior point method for LPs and SDPs, 2018. [arXiv:1808.09266](#).
- 29 Leonid G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):51–68, 1980.
- 30 James R. Lee, Prasad Raghavendra, and David Steurer. Lower bounds on the size of semidefinite programming relaxations. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC 2015)*. ACM, 2015. [arXiv:1411.6317](#).
- 31 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, pages 1049–1065. IEEE, 2015. [arXiv:1508.04874](#).
- 32 Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC 1993)*, pages 448–457. ACM, 1993.
- 33 John E. Mitchell. Polynomial interior point cutting plane methods. *Optimization Methods and Software*, 18(5):507–534, 2003.
- 34 Yurii Nesterov and Arkadi Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.
- 35 John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018. [arXiv:1801.00862](#).
- 36 Ewin Tang. Quantum-inspired classical algorithms for principal component analysis and supervised clustering, 2018. [arXiv:1811.00414](#).
- 37 Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 217–228. ACM, 2019. [arXiv:1807.04271](#).
- 38 Lieven Vandenbergh and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.
- 39 Xiaodi Wu. Parallelized solution to semidefinite programmings in quantum complexity theory, 2010. [arXiv:1009.2211](#).