Solving SDP Faster: A Robust IPM Framework and Efficient Implementation

Baihe Huang  
Peking University  
baiehuang@pku.edu.cn

Shunhua Jiang  
Columbia University  
sj30050@columbia.edu

Zhao Song  
Adobe Research  
zsong@adobe.com

Runzhou Tao  
Columbia University  
runzhou.tao@columbia.edu

Ruizhe Zhang  
UT-Austin  
ruizhe@utexas.edu

Abstract—
This paper introduces a new robust interior point method analysis for semidefinite programming (SDP). This new robust analysis can be combined with either logarithmic barrier or hybrid barrier.

Under this new framework, we can improve the running time of semidefinite programming (SDP) with variable size \( n \times n \) and \( m \) constraints up to \( \epsilon \) accuracy.

We show that for the case \( m = \Omega(n^2) \), we can solve SDPs in \( m^\omega \) time. This suggests solving SDP is nearly as fast as solving the linear system with equal number of variables and constraints. This is the first result that tall dense SDP can be solved in the nearly-optimal running time, and it also improves the state-of-the-art SDP solver [Jiang, Kathuria, Lee, Padmanabhan and Song, FOCS 2020].

In addition to our new IPM analysis, we also propose a number of techniques that might be of further interest, such as, maintaining the inverse of a Kronecker product using lazy updates, a general amortization scheme for positive semi-definite matrices.

I. INTRODUCTION

Semidefinite programming (SDP) optimizes a linear objective function over the intersection of the positive semidefinite (PSD) cone with an affine space. SDP is of great interest both in theory and in practice. Many problems in operations research, machine learning, and theoretical computer science can be modeled or approximated as semidefinite programming problems. In machine learning, SDP has applications in adversarial machine learning [RLS18], learning structured distribution [CLM20], sparse PCA [AW08], [dEGIL07], robust learning [DKK+16], [DHL19], [JLT20]. In theoretical computer science, SDP has been used in approximation algorithms for max-cut [GW94], coloring 3-colorable graphs [KMS94], and sparsest cut [ARV09], quantum complexity theory [IUW11], robust learning and estimation [CG18], [CDG19], [CDGW19], graph sparsification [LS17], algorithmic discrepancy and rounding [BDG16], [BG17], [Ban19], sum-of-squares optimization [BS16], [FKP19], [JNW22], terminal embeddings [CN21], and discrepancy theory [BG17], [BDG19], [DGLN19], [HRS21].

SDP is formally defined as follows:

Definition I.1 (Semidefinite programming). Given symmetric\(^1\) matrices \( C,A_1,\cdots,A_m \in \mathbb{R}^{n \times n} \) and a vector \( b \in \mathbb{R}^m \), the goal is to solve the following optimization problem:

\[
\max_{X \in \mathbb{R}^{n \times n}} \langle C,X \rangle \text{ subject to } \langle A_i,X \rangle = b_i, \ \forall i \in [m], \ X \succeq 0,
\]

where \( \langle A,B \rangle := \sum_{i,j} A_{i,j} B_{i,j} \) is the matrix inner product.

The input size of an SDP instance is \( mn^2 \), since there are \( m \) constraint matrices each of size \( n \times n \). The well-known linear programming (LP) is a simpler case than SDP, where \( X \succeq 0 \) and \( A_1,\cdots,A_m \) are restricted to be \( n \times n \) diagonal matrices. The input size of an LP instance is thus \( mn \).

Over the last many decades, there are three different lines of high accuracy SDP solvers (running time depending logarithmically on the accuracy). The first line of work is using the cutting plane method, such as [Sho77], [YN76], [Kha80], [KTE88], [NN89], [Vai89a], [KM03], [BV02], [LSW00], [JLSW20]. This line of work uses \( m \) iterations, and each iteration uses some SDP-based oracle call. The second line of work is using interior point method (IPM) and log barrier function such as [NN92], [JKL+20]. The third line of work is using interior point method and hybrid barrier function such as [NN94], [Ans00].

Recently, a line of work uses robust analysis and dynamic maintenance to speedup the running time of linear programming [CLS19], [Bra20], [BLSS20], [SY21], [JSWZ21], [Bra21]. One major reason made solving SDP much more harder than solving linear programming is: in LP the slack variable is a vector(can be viewed as a diagonal matrix), and in SDP the slack variable is a positive definite matrix. Due to that reason, the gradient/Hessian computation requires some complicated and heavy calculations based on the Kronecker product of matrices, while LP only needs the basic matrix-matrix product [Vai89b], [CLS19], [JSWZ21]. Therefore, handling the errors in each iteration and maintaining the slack matrices are way more harder in SDP. Thus, we want to ask the following question:

\(^1\) We can without loss of generality assume that \( C,A_1,\cdots,A_m \) are symmetric. Given any \( A \in \mathbb{R}^{n \times n} \), we have \( \sum_{i,j} A_{ij} X_{ij} = \sum_{j} A_{j} X_{jj} = \sum_{i} (A^T)_{ij} X_{ij} \) since \( X \) is symmetric, so we can replace \( A \) with \( (A + A^T)/2 \).
Can we efficiently solve SDP without computing exact gradient, Hessian, and Newton steps?

In this work, we will answer the above question by introducing new framework for both IPM analysis and variable maintenance. For IPM analysis, we build a robust IPM framework for arbitrary barrier functions that supports errors in computing gradient, Hessian, and Newton steps. For variable maintenance, we provide a general amortization method that gives improved guarantees on reducing the computational complexity by lazily updating the Hessian matrices.

For solving SDP using IPM with log barrier, the current best algorithm (due to Jiang, Kathuria, Lee, Padmanabhan and Song [JKL+20]) runs in \(O(\sqrt{n}(mn^2 + m\omega + n\omega))\) time. Since the input size of SDP is \(mn^2\), ideally we would want an SDP algorithm that runs in \(O(mn^2 + m\omega + n\omega)\) time, which is roughly the running time to solve linear systems. The current best algorithms are still at least a \(\sqrt{n}\) factor away from the optimal.

Inspired by the result [CLS19] which solves LP in the current matrix multiplication time, a natural and fundamental question for SDP is

**Can we solve SDP in the current matrix multiplication time?**

More formally, for the above formulation of SDP (Definition 1.1), is that possible to solve it in \(mn^2 + m\omega + n\omega\) time? In this work, we give a positive answer to this question by using our new techniques. For the tall dense SDP where \(m = \Omega(n^2)\), our algorithm runs in \(m\omega + m^{2+1/4}\) time, which matches the current matrix multiplication time. The tall dense SDP finds many applications and is one of the two predominant cases in [JKL+20]. This is the first result that shows SDP can be solved as fast as solving linear systems.

Finally, we also show that our techniques and framework are quite versatile and can be used to directly speedup the SDP solver via the hybrid barrier [NN89], [Ans00].

**a) Our results:** We present the simplified version of our main result in the following theorem. The formal version can be found in the full version.

**Theorem 1.2 (Main result).** For \(\epsilon\)-accuracy, there is a classical algorithm that solves a general SDP instance with variable size \(n \times n\) and \(m\) constraints in time \(O((\sqrt{n}(mn^2 + n^4) + m\omega + n^{3\omega})\log(1/\epsilon))\), where \(\omega\) is the exponent of matrix multiplication.

In particular, when \(m = \Omega(n^2)\), our algorithm takes matrix multiplication time \(m\omega\) for current \(\omega \approx 2.373\).

2We note that a recent breakthrough result by Peng and Vempala [PV21] showed that a sparse linear system can be solved faster than matrix multiplication. However, their algorithm essentially rely on the sparsity of the problems. And it is still widely believed that general linear system requires matrix multiplication time.

3See Table 2.1 in [JKL+20] and Section VI.

4We use \(O^*(\cdot)\) to hide \(n^{\omega(1)}\) and \(\log(O^*(mn))\) factors, and \(\tilde{O}(\cdot)\) to hide \(\log(O^*(mn))\) factors.

**Remark 1.3.** For any \(m \geq n^{2-0.5/\omega} \approx n^{1.79}\) with current \(\omega \approx 2.37286\) [Wii12], [LG14], [AW21], our algorithm runs faster than [JKL+20].

Theorem 1.2 and [JKL+20] are focusing on the log barrier method for solving SDP. However, the area of speeding up the hybrid barrier-based SDP solver is quite blank. We also improve the state-of-the-art implementation of the hybrid barrier-based SDP solver [NN89], [Ans00] in all parameter regimes. See Section V and Theorem V.1 for more details.

b) **Roadmap:** In Section II, we review the previous approaches for solving SDP and discuss their bottlenecks. In Section III, we introduce our robust framework for IPM. In Section IV, we show our main techniques and sketch the proof of our main result (Theorem 1.2). In Section V, we overview the approach of applying our robust framework to speedup the hybrid barrier-based SDP solver. Related works are provided in Section VI. We define our notations and include several useful tools in Section VII. In the full version [HJS+22], we give the formal version of our algorithm and the main theorem, where the proof is given in the full version. Our general robust IPM framework is displayed in the full version. Our fast implementation of the hybrid barrier-based SDP solver can be found in the full version.

II. AN OVERVIEW OF PREVIOUS TECHNIQUES

Under strong duality, the primal formulation of the SDP in Eq. (1) is equivalent to the following dual formulation:

**Definition II.1 (Dual problem).** Given symmetric matrices \(C,A_1,\ldots,A_m \in \mathbb{R}^{n \times n}\) and \(b_i \in \mathbb{R}\) for all \(i \in [m]\), the goal is to solve the following convex optimization problem:

\[
\min_{y \in \mathbb{R}^n} \quad b^T y \quad \text{subject to} \quad S = \sum_{i=1}^m y_i A_i - C, \quad S \succeq 0. \tag{2}
\]

Interior point methods (IPM) solve the above problem by (approximately) following a central path in the feasible region \(\{ y \in \mathbb{R}^n : S = \sum_{i=1}^m y_i A_i - C \succeq 0 \}\). As a rich subclass of IPM, barrier methods [NN92], [Ans00] define a point on the central path as the solution to the following optimization problem parametrized by \(\eta > 0\):

\[
\eta f_\eta(y) := \eta \cdot \langle b, y \rangle + \phi(y) \tag{3}
\]

is the augmented objective function and \(\phi : \mathbb{R}^m \to \mathbb{R}\) is a barrier function that restricts \(y\) to the feasible region since \(\phi(y)\) increases to infinity when \(y\) approaches the boundary of the feasible region. Barrier methods usually start with an initial feasible \(y\) for a small \(\eta\), and increase \(\eta\) in each iteration until \(y\) is close to the optimal solution of the SDP. In short-step barrier methods with log barrier, \(\eta' = (1 + 1/\sqrt{\eta})\eta\).

5The choice of the barrier function leads to different numbers of iterations. Nesterov and Nemirovski [NN92] utilize the log barrier function \(\phi_{\log}(y) = -\log \det(S)\) which guarantees convergence in \(O(\sqrt{\eta})\) iterations. Anstreicher [Ans00] uses the Hybrid barrier \(\phi_{\text{hybrid}}(y) = 225(n/m)^{1/2}\). \(\phi_{\text{vol}}(y) + \phi_{\log}(y) \cdot (m - 1)/(n - 1)\) where \(\phi_{\text{vol}}(y)\) is the volumetric barrier \(\phi_{\text{vol}}(y) = \frac{1}{2} \log \det(\nabla^2 \phi_{\log}(y))\). Hybrid barrier guarantees convergence in \(O((mn)^{1/4})\) iterations.
It takes a Newton step \(-H(y)^{-1}g(y, \eta)\) in each iteration to keep \(y\) in the proximity of the central path. Here \(g(y, \eta)\) and \(H(y)\) are the gradient and the Hessian of \(f_\eta(y)\).

**Techniques and bottlenecks of existing algorithms**

Fast solvers of SDP include the cutting plane method and interior point method. The fastest known algorithms for SDP based on the cutting plane method [LSW15], [JLSW20] have \(m\) iterations and run in \(O(n(mn^2 + n^2 + m^2))\) time. The fastest known algorithm for SDP based on the interior point method [JKL⁺20] has \(\sqrt{n}\) iterations and runs in \(O^*(\sqrt{n}(mn^2 + n^2 + m^2))\) time. In most applications of SDP, where \(m \geq n\), interior point method of [JKL⁺20] runs faster. In the following we briefly discuss the techniques and bottlenecks of interior point methods.

a) Central path: Interior point method updates the dual variable \(y\) by Newton step \(-H(y)^{-1}g(y, \eta)\) to keep it in the proximity of central path. This proximity is measured by the potential function \(\|H(y)^{-1}g(y, \eta)\|_2\). In classical interior point literature, this potential function is well controlled by taking exact Newton step (see e.g. [Ren01]). [JKL⁺20] relaxes this guarantee and allows PSD approximation to the Hessian matrix \(H(y)\). However, their convergence also relies on exact computation of slack matrix \(S\) and gradient \(g\). This leads to a \(mn^{2.5}\) term in their running time.

b) Amortization techniques: [JKL⁺20] keeps a PSD approximation \(\tilde{H}\) of the Hessian \(H\) and updates \(\tilde{H}\) by a low rank matrix in each iteration. The running time of this low rank update is then controlled by a delicate amortization technique. This technique also appears in linear programming [CLS19] and empirical risk minimization [LSZ19]. [JKL⁺20] brings this technique to SDP, and costs \(n^{1.5}m^2\) time in computing the inverse of Hessian matrix. When \(m\) becomes larger, this term dominates the complexity and becomes undesirable.

**III. THE ROBUST SDP FRAMEWORK**

In this section, we introduce our robust SDP framework. This framework works for general barrier functions and finds applications in several Algorithms in the full version [HJS⁺22]. We consider self-concordant barrier function \(\Phi\) with complexity \(\theta\). For the regularized optimized \(f_\eta\) in Eq. (3), we define the gradient \(g: \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^m\) as \(g(y, \eta) = \eta \cdot b - \nabla \phi(y)\).

Interior point method takes Newton step \((\nabla^2 \phi(y))^{-1}g(y, \eta)\) and guarantees the variables in the proximity of the central path by bounding the potential function \(\Phi(z, y, \eta) = \|g(y, \eta)/(\nabla^2 \phi(z))^{-1}\|\). In practical implementations, there are perturbations in the Newton step due to slack matrix \(S\), gradient \(g(y, \eta)\), Hessian matrix \(\nabla^2 \phi(y)\) and Newton step \((\nabla^2 \phi(y))^{-1}g(y, \eta)\). Many fast algorithms maintain approximations to these quantities to reduce the running time.

We propose a more general robust framework (compared with [Ren01], [JKL⁺20]) which captures all these errors. We show that as long as these errors are bounded by constants in the local norm\(^6\), the potential function stays bounded, which guarantees the closeness to central path. Therefore this analysis is currently the most robust possible. The main component of our robust analysis is the following one step error control.

**Lemma III.1** (One step error control of the robust framework, informal version). Let the potential function of IPM defined by \(\Psi(z, y, \eta) := \|g(y, \eta)/(\nabla^2 \phi(z))^{-1}\|.\) Given any parameters \(\alpha_S \in [1, 1 + 10^{-4}], c_H \in [10^{-1}, 1], \epsilon_\theta, \epsilon_\delta \in [0, 10^{-4}],\) and \(\epsilon_N \in (0, 10^{-1}), \eta > 0\). Suppose that there is

- **Condition 0.** a feasible dual solution \(y \in \mathbb{R}^m\) satisfies \(\Phi(y, y, \eta) \leq \epsilon_N\).
- **Condition 1.** a symmetric matrix \(\tilde{H} \in \mathbb{S}^{n \times n}\) satisfies \(c_H \cdot \nabla^2 \phi(y) \geq \tilde{H} \geq \nabla^2 \phi(y)\).
- **Condition 2.** a vector \(\tilde{y} \in \mathbb{R}^m\) satisfies \(\|\tilde{g} - g(y, \eta_{\text{new}})/(\nabla^2 \phi(y))^{-1}\| \leq \epsilon_\theta \cdot \|g(y, \eta_{\text{new}})/(\nabla^2 \phi(y))^{-1}\|\).
- **Condition 3.** a vector \(\delta_{\tilde{y}} \in \mathbb{R}^m\) satisfies \(\|\tilde{g} / \nabla^2 \phi(y) - \tilde{H}^{-1} \delta_{\tilde{y}}\| \leq \epsilon_\delta \cdot \|\tilde{H}^{-1}\| \cdot \tilde{g} / \nabla^2 \phi(y)\).

Then \(y_{\text{new}} = \eta(1 + \frac{\epsilon_\theta}{20\sqrt{\theta}})\) and \(y_{\text{new}} = y - \delta_{\tilde{y}}\) satisfy \(\Psi(y_{\text{new}}, y_{\text{new}}, \eta_{\text{new}}) \leq \epsilon_N\).

This result suggests that as long as we find an initial dual variable \(y\) in the proximity of central path, i.e. \(\Phi(y, y, \eta) \leq \epsilon_N\). Lemma III.1 will guarantee that the invariant \(\Phi(y, y, \eta) \leq \epsilon_N\) holds throughout Algorithm 1, even when there exist errors in the slack matrices, Hessian, gradient and Newton steps. As shown in the full version [HJS⁺22], the duality gap is upper bounded by \(\theta \cdot \Phi(y, y, \eta)/\eta\). In at most \(O(\sqrt{\theta} \cdot \log(\theta/\epsilon))\) iterations, \(\eta\) will become greater than \(\theta \cdot \Phi(y, y, \eta)/\epsilon\). Therefore Algorithm 1 finds \(\epsilon\)-optimal solution within \(O(\sqrt{\theta} \cdot \log(\theta/\epsilon))\) iterations.

We note that [Ans00] and [Ren01] only consider Condition 0 and requires the \(c_H = 1, \epsilon_\theta = \epsilon_\delta = 0\) in Condition 1, 2, and 3. [JKL⁺20] considered Condition 0 and Condition 1 in Lemma III.1 and requires the \(\epsilon_\theta = \epsilon_\delta = 0\) in Condition 2 and 3. Moreover, the Condition 1 in [JKL⁺20] requires \(c_H\) to be very close to 1, and we relax this condition to support any constant in \([10^{-1}, 1]\). In addition, our framework also relaxes the computation of gradient and Newton direction to allow some approximations, which makes it possible to apply more algorithmic techniques in the interior-point method. More details are provided in the full version [HJS⁺22].

**IV. OUR TECHNIQUES**

In this section, we introduce our main techniques, and provide a self-contained proof sketch of our main result Theorem I.2. We tackle the two bottlenecks of \(\eta^{1.5}\) cost per iteration in [JKL⁺20] by proposing two different techniques:

**Bottleneck 1:** Instead of inverting the Hessian matrix from scratch in each iteration, we make use of the already-computed Hessian inverse of the previous iteration. We prove that

\[8\]See condition 0-3 in Lemma III.1 for details.
using low-rank updates, the change to the inverse of Hessian matrices (computed using Kronecker product) is low-rank, and thus we can use Woodbury identity to efficiently update the Hessian inverse. In Section IV-A we introduce the low-rank update to the Hessian, and in Section IV-B we describe how to compute the Hessian inverse efficiently using Woodbury identity and fast matrix rectangular multiplication.

**Bottleneck 2:** We propose a better amortization scheme for PSD matrices that improves upon the previous \( m^2 \) amortized cost. We give a proof sketch of our amortized analysis in Section IV-C.

**A. Low rank update of Hessian**

Low-rank approximation of Kronecker product itself is an interesting problem and has been studied in [SWZ19]. In this section, we describe how the low-rank update of the slack matrix leads to a low-rank update of the Hessian matrix that involves Kronecker product.

The Hessian matrix is defined as \( H = A \cdot (S^{-1} \otimes S^{-1}) \cdot A^\top \). We take the following three steps to construct the low-rank update of \( H \).

a) **Step 1: low-rank update of the slack matrix:** We use an approximate slack matrix that yields a low-rank update. In the \( t \)-th iteration of Algorithm 2, we use \( \tilde{S} \) to denote the current approximate slack matrix, and \( S_{\text{new}} \) to denote the new exact slack matrix. We will use \( \tilde{S} \) and \( S_{\text{new}} \) to find the new approximate slack matrix \( S_{\text{new}} \).

Define \( Z = (S_{\text{new}})^{-1/2} \tilde{S} (S_{\text{new}})^{-1/2} − I \) which captures the changes of the slack matrix. We compute the spectral decomposition: \( Z = U \cdot \text{diag}(\lambda) \cdot U^\top \). We show that

\[
\sum_{i=1}^{n} \lambda_i^2 = \| S_{\text{new}}^{-1/2} \tilde{S} (S_{\text{new}})^{-1/2} - I \|_F = O(1),
\]

which implies that only a few eigenvalues of \( Z \) are significant, say e.g. \( \lambda_1, \ldots, \lambda_{r_1} \). We only keep these eigenvalues and set the rest to be zero. In this way we get a low-rank approximation of \( Z \): \( \tilde{Z} = U \cdot \text{diag}(\tilde{\lambda}) \cdot U^\top \) where \( \tilde{\lambda} = [\lambda_1, \ldots, \lambda_{r_1}, 0, \ldots, 0]^\top \). Now we can use \( \tilde{Z} \) to update the approximate slack matrix by a low-rank matrix:

\[
\tilde{S}_{\text{new}} = \tilde{S} + (S_{\text{new}})^{1/2} \cdot \tilde{Z} \cdot (S_{\text{new}})^{1/2} = \tilde{S} + V_1 \cdot V_2^\top,
\]

where \( V_1 \) and \( V_2 \) both have size \( n \times r_1 \). Since \( \tilde{Z} \) is a good approximation of \( Z \), \( \tilde{S}_{\text{new}} \) is a PSD approximation of \( S_{\text{new}} \), which guarantees that \( y \) still lies in the proximity of the central path.

b) **Step 2: low-rank update of inverse of slack:** Using Woodbury identity, we can show that

\[
(S_{\text{new}})^{-1} = (S + V_1 \cdot V_2^\top)^{-1} = \tilde{S}^{-1} + V_3 V_4^\top,
\]

where \( V_3 = -\tilde{S}^{-1} V_1 (I + V_2^\top \tilde{S}^{-1} V_1)^{-1} \) and \( V_4 = \tilde{S}^{-1} V_2 \) both have size \( n \times r_1 \). Thus, this means \((S_{\text{new}})^{-1} - \tilde{S}^{-1}\) has a rank \( r_1 \) decomposition.

c) **Step 3: low-rank update of Hessian:** Using the linearity and the mixed product property (Part 2 of Fact VII.7) of Kronecker product, we can find a low-rank update to \((S_{\text{new}})^{-1} \otimes (S_{\text{new}})^{-1}\). More precisely, we can rewrite \((S_{\text{new}})^{-1} \otimes (S_{\text{new}})^{-1}\) as follows:

\[
(S_{\text{new}})^{-1} \otimes (S_{\text{new}})^{-1} = (S_{\text{new}})^{-1} \otimes (S_{\text{new}})^{-1} \sim (\tilde{S}^{-1} + V_3 V_4^\top)
\]

The term \( S_{\text{diff}} \) is the difference that we want to compute, we can show

\[
S_{\text{diff}} = \tilde{S}^{-1} \otimes (V_3 V_4^\top) + (V_3 V_4^\top) \otimes \tilde{S}^{-1} + (V_3 V_4^\top) \otimes (V_3 V_4^\top)
\]

\[
= (\tilde{S}^{-1/2} \otimes V_3) \cdot (\tilde{S}^{-1/2} \otimes V_4^\top) + (V_3 \otimes \tilde{S}^{-1/2}) \cdot (V_4^\top \otimes \tilde{S}^{-1/2})
\]

\[
+ (V_3 \otimes V_2) \cdot (V_4^\top \otimes V_4^\top)
\]

\[
= Y_1 \cdot Y_2^\top
\]
Algorithm 2 Informal version. An implementation of \textsc{GeneralRobustSdp}

1: \textbf{procedure} \textsc{SolvesDP}( \( A \in \mathbb{R}^{m \times n^2}, b \in \mathbb{R}^m, C \in \mathbb{R}^{n \times n} \))

2: \textbf{for} \( t = 1 \rightarrow T \) \textbf{do}

3: \quad \eta^{\text{new}} \leftarrow \eta \cdot (1 + 1/\sqrt{\eta})

4: \quad g_{\eta^{\text{new}}}((y)_j) \leftarrow \eta^{\text{new}} \cdot b_j - \text{tr}[S^{-1} \cdot A_j], \; \forall j \in m

5: \quad \delta_y \leftarrow -\tilde{H}^{-1} \cdot g_{\eta^{\text{new}}} (y)

6: \quad y^{\text{new}} \leftarrow y + \delta_y

7: \quad S^{\text{new}} \leftarrow \sum_{\eta^{\text{new}}(y^{\text{new}})} A_k = C

8: \quad \text{Compute } V_1, V_2 \in \mathbb{R}^{n \times n_r} \text{ such that } \tilde{S}^{\text{new}} = \tilde{S} + V_1 \cdot V_2^T

9: \quad \text{Compute } V_3, V_4 \in \mathbb{R}^{n \times n_r} \text{ such that } (\tilde{S}^{\text{new}})^{-1} = (\tilde{S})^{-1} + V_3 \cdot V_4^T

10: \quad \text{Compute } AY_1, AY_2 \in \mathbb{R}^{m \times n_r} \text{ such that } \tilde{H}^{\text{new}} = \tilde{H} + (AY_1) \cdot (AY_2)^T

11: \quad (\tilde{H}^{\text{new}})^{-1} \leftarrow \tilde{H}^{-1} + \text{low-rank update}

12: \quad y \leftarrow y^{\text{new}}, S \leftarrow S^{\text{new}], \tilde{S} \leftarrow S^{\text{new]], \tilde{H}}^{-1} \leftarrow (\tilde{H}^{\text{new}})^{-1}\)

13: \textbf{end for}

14: \textbf{end procedure}

\[
\begin{align*}
\text{where } Y_1 \text{ and } Y_2 \text{ both have size } n^2 \times n_r. \text{ In this way we get a low-rank update to the Hessian:} \\
\tilde{H}^{\text{new}} &= A \cdot ((\tilde{S}^{\text{new}})^{-1} \otimes (\tilde{S}^{\text{new}})^{-1}) \cdot A^T \\
&= \tilde{H} + (AY_1) \cdot (AY_2)^T.
\end{align*}
\]

\section*{B. Computing Hessian inverse efficiently}

In this section we show how to compute the Hessian inverse efficiently.

Using Woodbury identity again, we have a low rank update to \( \tilde{H}^{-1} \):

\[
(\tilde{H}^{\text{new}})^{-1} = (\tilde{H} + (AY_1) \cdot (AY_2)^T)^{-1} = \tilde{H}^{-1} - \tilde{H}^{-1} \cdot AY_1 \cdot (I + Y_2^T \cdot AY_1)^{-1} \cdot Y_2^T \cdot A^T \cdot \tilde{H}^{-1}.
\]

The second term in the above equation has rank \( n_r \). Thus \( (\tilde{H}^{\text{new}})^{-1} - \tilde{H}^{-1} \) has a rank \( nr \) decomposition. To compute \( (\tilde{H}^{\text{new}})^{-1} \) in each iteration, we first compute \( AY_1, AY_2 \in \mathbb{R}^{m \times n_r} \) and multiply it with \( \tilde{H}^{-1} \in \mathbb{R}^{m \times m} \) to get \( \tilde{H}^{-1} \cdot AY_1, \tilde{H}^{-1} \cdot AY_2 \in \mathbb{R}^{n \times n_r} \). Then we compute \( (I + Y_2^T \cdot A^T \cdot AY_1)^{-1} \in \mathbb{R}^{n_r \times n_r} \) and find its inverse \( (I + Y_2^T \cdot A^T \cdot AY_1)^{-1} \in \mathbb{R}^{n \times n} \). Finally, we multiply \( \tilde{H}^{-1} \cdot AY_1, \tilde{H}^{-1} \cdot AY_2 \in \mathbb{R}^{m \times n_r} \) and \( (I + Y_2^T \cdot A^T \cdot AY_1)^{-1} \in \mathbb{R}^{n_r \times n_r} \) together to obtain \( (\tilde{H}^{-1} \cdot AY_1, (I + Y_2^T \cdot A^T \cdot AY_1)^{-1}, Y_2^T \cdot A^T \cdot \tilde{H}^{-1} \in \mathbb{R}^{m \times m} \), as desired. Using fast matrix multiplication in each aforementioned step, the total computation cost is bounded by

\[
O(T_{\text{mat}}(m, n^2, n_r) + T_{\text{mat}}(m, m, n_r) + (n_r)^2).
\]

\section*{C. General amortization method}

As mentioned in the previous sections, our algorithm relies on the maintenance of the slack matrix and the inverse of the Hessian matrix via low-rank updates. In each iteration, the time to update \( S \) and \( H \) to \( S^{\text{new}} \) and \( H^{\text{new}} \) is proportional to the magnitude of low-rank change in \( S \), namely

\[
r_t = \text{rank}(\tilde{S}^{\text{new}} - \tilde{S}).
\]

To deal with \( r_t \), we propose a general amortization method which extends the analysis of several previous work [CLS19], [LSZ19], [JKL\textsuperscript{+}20]. We first prove a tool to characterize intrinsic properties of the low-rank updates, which may be of independent interest.

\textbf{Theorem IV.1} (Informal version). \textit{Given a sequence of \textit{approximate slack matrices} } \( S^{(1)}, S^{(2)}, \ldots, S^{(T)} \in \mathbb{R}^{n \times n} \textit{generated by Algorithm in the full version } [\text{HJS}^{+22}], \textit{let } r_t = \text{rank}(S^{(t+1)} - S^{(t)}) \textit{denotes the rank of update on } S^{(t)}. \textit{Then for any non-increasing vector } g \in \mathbb{R}_+^n \textit{ (i.e., } g_1 \geq \cdots \geq g_n), \textit{we have}

\[
\sum_{t=1}^T r_t \cdot g_t \leq \tilde{O}(T \cdot \|g\|^2).
\]

Next, we show a proof sketch of Theorem IV.1.

\textbf{Proof.} For any matrix } \( Z \), \textit{let } \|\lambda(Z)\|_i \textit{ denotes its } i\text{-th largest absolute eigenvalue. We use the following potential function } \Phi_g(Z) := \sum_i^n g_i \cdot |\lambda(Z)|_{ii}. \textit{ Further, for convenience, we define } \Phi_g(S^{(1)}, S^{(2)}) := \Phi_g(S^{1/2} S^{1/2} - I). \textit{ Our proof consists of the following two parts:}

- \textit{The change of the exact slack matrix increases the potential by a small amount, specifically } \Phi_g(S^{\text{new}}, \tilde{S}) - \Phi_g(S, \tilde{S}) \leq \|g\|_2.
- \textit{The change of the approximate slack matrix decreases the potential proportionally to the update rank, specifically } \Phi_g(S^{\text{new}}, \tilde{S}^{\text{new}}) - \Phi_g(S^{\text{new}}, \tilde{S}) \leq -r_t \cdot g_t.

\textit{In each iteration, the change of potential is composed of the changes of the exact and the approximate slack matrices:}

\[
\Phi_g(S^{\text{new}}, \tilde{S}^{\text{new}}) - \Phi_g(S, \tilde{S}) = \Phi_g(S^{\text{new}}, \tilde{S}) - \Phi_g(S, \tilde{S}) + \Phi_g(S^{\text{new}}, \tilde{S}^{\text{new}}) - \Phi_g(S^{\text{new}}, \tilde{S}).
\]

\textit{Note that } \Phi_g(S, \tilde{S}) = 0 \textit{ holds in the beginning of our algorithm and } \Phi_g(S, \tilde{S}) \geq 0 \textit{ holds throughout the algorithm, combining the observations above we have } T \cdot \|g\|^2 - \sum_{t=1}^T r_t \cdot g_t \geq 0 \textit{ as desired.} \qed
a) Amortized analysis. Next we show how to use Theorem IV.1 to prove that our algorithm has an amortized cost of $m^{o(1/4)} + m^2$ cost per iteration when $m = \Omega(n^2)$. Note that in this case there are $\sqrt{n} = m^{1/4}$ iterations.

When $m = \Omega(n^2)$, the dominating term in our cost per iteration (see Eq. (4)) is $T_{\text{mat}}(m, m, nr_t)$. We use fast rectangular matrix multiplication (For more details, see Section VII-D) to upper bound this term by

$$T_{\text{mat}}(m, m, nr_t) \leq m^2 + m^2 - \frac{n(n-2)}{m} \cdot n^{5/2} \cdot r_t^m.$$

We define a non-increasing sequence $g \in \mathbb{R}$ as $g_t = \frac{n-2}{m}$. This $g$ is tailored for the above equation, and its $\ell_2$ norm is bounded by $\|g\| \leq O(n^{5/2})$.

Combining this and the previous equation, and since we assume $m = \Omega(n^2)$, we have

$$T \sum_{t=1}^T T_{\text{mat}}(m, m, nr_t) \leq T \cdot (m^2 + m^2 - \frac{n(n-2)}{m} \cdot n^{5/2} - 1/2).$$

Since $T = O(m^{1/4})$, we proved the desired computational complexity in Theorem 1.2.

V. SOLVING SDP WITH HYBRID BARRIER

Volumetric barrier was first proposed by Vaidya [Vai89a] for the polyhedral, and was generalized to the spectrahedra $\{y \in \mathbb{R}^m : y_1 A_1 + \cdots + y_n A_n \preceq 0\}$ by Nesterov and Nemirovski [NN94]. They showed that the volumetric barrier $\phi_{\text{vol}}$ makes the interior point method converge in $O(m^{11/4})$ iterations, while the log barrier $\phi_{\log}$ need $\sqrt{n}$ iterations. By combining the volumetric barrier and the log barrier, they also showed that the hybrid barrier achieves $(mn)^{1/4}$ iterations, Anstreicher [Ans00] gave a much simplified proof of this result.

We show that the hybrid barrier also fits into our robust IPM framework. And we can apply our newly developed low-rank update and amortization techniques in the log barrier case to efficiently implement the SDP solver based on hybrid barrier. The informal version of our result is stated in below.

Theorem 1 (Informal version). There is an SDP algorithm based on hybrid barrier which takes $(mn)^{1/4} \log(1/e)$ iterations with cost-per-iteration $O^{*}(m^2 n^{\omega} + m^4)$. In particular, our algorithm improves [Ans00] in nearly all parameter regimes. For example, if $m = n^2$, our new algorithm takes $n^{8.75}$ time while [Ans00] takes $n^{10.75}$ time. If $m = n$, our new algorithm takes $n^{5.1}$ time, while [Ans00] takes $n^{5.5}$ time.

The hybrid barrier function is as follows:

$$\phi(y) := 225 \sqrt{\frac{n}{m}} \cdot \left( \phi_{\text{vol}}(y) + \frac{m-1}{m} \cdot \phi_{\log}(y) \right).$$

where $\phi_{\text{vol}}(y) = \frac{1}{2} \log \det(\nabla^2 \phi_{\text{vol}}(y))$. According to our general IPM framework (Algorithm 1), we need to efficiently compute the gradient and Hessian of $\phi(y)$. Recall from [Ans00] that the gradient of the volumetric barrier is:

$$(\nabla \phi_{\text{vol}}(y))_i = -\text{tr}(H(S)^{-1} \cdot A(S^t A_i S^{-1} \otimes S^{-1}) A^T)$$

\forall i \in [m].

And the Hessian can be written as $\nabla^2 \phi_{\text{vol}}(y) = 2Q(S) + R(S) - 2T(S)$, where for any $i, j \in [m]$,

$$(Q(S)_{i,j}) = \text{tr}(H(S)^{-1} A(S^t A_i S^{-1} \otimes S^{-1} A_j S^{-1} A_j^T) A^T),$$

$$(R(S)_{i,j}) = \text{tr}(H(S)^{-1} A(S^t A_i S^{-1} \otimes S^{-1} A_j S^{-1} A_j^T) A^T),$$

$$T(S)_{i,j} = \text{tr}(H(S)^{-1} A(S^t A_i S^{-1} \otimes S^{-1} A_j S^{-1} A_j)^T) A^T H(S)^{-1} A(S^t A_i S^{-1} \otimes S^{-1} A_j S^{-1} A_j^T) A^T).$$

Here, $\otimes_S$ is the symmetric Kronecker product.

A straight-forward implementation of the hybrid barrier-based SDP algorithm can first compute the matrices $S^{-1} A_i$ and $S^{-1} A_j S^{-1} A_i$ for all $i, j \in [m]$, which takes $O(m^2 n^3)$-time. The gradient $\nabla \phi(y)$ and the Hessian of $\phi_{\log}(y)$ can be computed by taking traces of these matrices. To compute $\nabla^2 \phi_{\text{vol}}(y)$, $Q(S), R(S), T(S)$, we observe that each entry of these matrices can be written as the inner-product between $H(S)^{-1}$ and some matrices formed in terms of $\text{tr}(S^{-1} A_i S^{-1} A_j S^{-1} S^{-1} A_i)$ and $\text{tr}(S^{-1} A_i S^{-1} A_j S^{-1} A_j S^{-1} A_i)$ for $i, j, k, l \in [m]$. Hence, we can spend $O(m^2 n^2)$-time computing these traces and then get $\nabla \phi_{\text{vol}}(y), Q(S), R(S), T(S)$ in $O(m^2 n^2)$-time. After obtaining the gradient and Hessian of the hybrid barrier function, we finish the implementation of IPM SDP solver by computing the Newton direction $\delta_y = -((\nabla^2 \phi(y))^{-1} (\eta - \nabla \phi(y)))$. (More details are given in the full version).

To speedup the straight-forward implementation, we observe two bottleneck steps in each iteration:

1. Computing the traces $\text{tr}(S^{-1} A_i S^{-1} A_j S^{-1} S^{-1} A_i)$ for $i, j, k, l \in [m].$
2. Computing the matrices $Q(S), R(S), T(S)$.

To handle the first issue, we use the low-rank update and amortization techniques introduced in the previous section to approximate the change of the slack matrix $S$ by a low-rank matrix. One challenge for the volumetric barrier is that its Hessian (Eq. (5)) is much more complicated than the log barrier’s Hessian $H(S)$. For $H(S)$, if we replace $S$ with its approximation $\tilde{S}$, then $H(\tilde{S})$ will be a PSD approximation of $H(S)$. However, this may not hold for the volumetric barrier’s Hessian if we simply replace all the $S$ in $\nabla^2 \phi(y)$ by its approximation $\tilde{S}$. We can resolve this challenge by carefully choosing the approximation place: if we approximate the second $\tilde{S}$ in the trace, i.e., $\text{tr}(S^{-1} A_i S^{-1} A_j S^{-1} S^{-1} A_i)$, then the resulting matrix will be a PSD approximation of $\nabla^2 \phi(y)$. In other words, the Condition 1 in our robust IPM framework (Lemma III.1) is satisfied. Notice that in each iteration, we only need to maintain the change of $A \otimes S B := \frac{1}{2}(A \otimes B + B \otimes A)$.  

238
\( \text{tr}[S^{-1}A_iS^{-1}A_jS^{-1}A_kS^{-1}A_l] \), which by the low-rank guarantee, can be written as

\[
\text{tr}[A_iS^{-1}A_j \cdot V_3V_4^\top \cdot A_jS^{-1}A_kS^{-1}],
\]

where \( V_3, V_4 \in \mathbb{R}^{n \times r_l} \). Then, we can first compute the matrices

\[
\{ A_iS^{-1}A_jV_3 \in \mathbb{R}^{n \times r_l} \}_{i, j \in [m]}
\]

and

\[
\{ V_4^\top A_jS^{-1}A_kS^{-1} \in \mathbb{R}^{r_l \times n} \}_{j, k \in [m]}.
\]

It takes \( m^2 \cdot T_{\text{mat}}(n, n, r_l) \)-time. And we can compute all the traces \( \text{tr}[S^{-1}A_iS^{-1}A_jS^{-1}A_kS^{-1}A_l] \) simultaneously in \( T_{\text{mat}}(m^2, nr_l, r^2) \) by batching them together and using fast matrix multiplication on a \( m^2 \)-by-\( nr_l \) matrix and a \( nr_l \)-by-\( n^2 \) matrix. A similar amortized analysis in the log barrier case can also be applied here to get the amortized cost-per-iteration for the low-rank update. One difference is that the potential function \( \Phi_g(Z) \) (defined in Section IV-C) changes more drastically in the hybrid barrier case. And we can only get \( \sum_{t=1}^{T} r_t \cdot g_{t,S} \leq O(T \cdot (n/m)^{1/4} \cdot \|g\|_2 \cdot \log n) \).

To handle the second issue, we note that computing the \( T(S) \) matrix is the most time-consuming step, which need \( m^{w+2} \)-time. In [Ans00], it is proved that \( \frac{1}{3}Q(S) \preceq \nabla^2 \phi_{\text{vol}}(y) \preceq Q(S) \). With this PSD approximation, our robust IPM framework enables us to use \( Q(S) \) as a “proxy Hessian” of the volumetric barrier. That is, in each iteration, we only compute \( Q(S) \) and ignore \( R(S) \) and \( T(S) \). And computing \( Q(S) \) only takes \( O(m^3) \)-time, which improves the \( m^{w+2} \) term in the straight forward implementation.

Combining them together, we obtain the running time in Theorem VI. More details are provided in the full version [HJS+22].

a) Lee-Sidford barrier for SDP?: In LP, the hybrid barrier was improved by Lee and Sidford [LS19] to achieve \( O^*(\sqrt{\min\{m,n\}}) \) iterations. For SDP, we hope to design a barrier function with \( O^*(\sqrt{m}) \) iterations. However, the Lee-Sidford barrier function does not have a direct correspondence in SDP due to the following reasons. First, [LS19] defined the barrier function in the dual space of LP which is a polyhedron, while for SDP, the dual space is a spectrahedron. Thus, the geometric intuition of the Lee-Sidford barrier (John’s ellipsoid) may not be helpful to design the corresponding barrier for SDP. Second, efficient implementation of Lee-Sidford barrier involves a primal-dual central path method [BLSS20]. However, the cost of following primal-dual central path in SDP is prohibitive since this involves solving Lyapunov equations in \( \mathbb{R}^{n \times n} \). Third, the Lewis weights play an important role in the Lee-Sidford barrier. Notice that in LP, the volumetric barrier can be considered as reweighting the constraints in the log barrier based on the leverage score, and the Lee-Sidford barrier uses Lewis weights for reweighting to improve the volumetric barrier. However, in SDP, we have observed that the leverage score vector becomes the leverage score matrix. Thus, we may need some matrix version of Lewis weights to define the Lee-Sidford barrier for SDP. In the full version [HJS+22], we study some properties of the leverage score matrix and give an algorithm to efficiently compute it in each iteration of the IPM, which might be the first step towards improving the SDP hybrid barrier.

VI. RELATED WORK

a) Other SDP solvers.: The interior point method is a second-order algorithm. Second-order algorithms usually have logarithmic dependence on the error parameter 1/\( \epsilon \). First-order algorithms do not need to use second-order information, but they usually have polynomial dependence on 1/\( \epsilon \). There is a long list of work focusing on first-order algorithms [AK07], [GH16], [AZL17], [CDST19], [LP20], [YTF+19], [JY11], [ALO16], [JLL+20]. Solving SDPs has also attracted attention in the parallel setting [JY11], [JY12], [ALO16], [JLL+20].

b) Cutting plane method.: Cutting plane method is a class of optimization algorithms that iteratively queries a separation oracle to cut the feasible set that contains the optimal solution. There has been a long line of work to obtain fast cutting plane methods [Sho77], [YN76], [Kha80], [KTE88], [NN89], [Vai89a], [AV95], [BVO2], [LSW15], [JLSW20].

c) Low-rank approximation: Low-rank approximation is a well-studied topic in numerical linear algebra [Sar06], [CW13], [BWZ16], [RSW16], [SWZ17], [SWZ19]. Many different settings of that problem have been studied. In this paper, we are dealing with Kronecker product type low rank approximation.

d) Applications of SDP.: As described by [JKL+20], \( m = \Omega(n^2) \) is an essential case of using SDP to solve many practical combinatorial optimization problems. Here we provide a list of examples, e.g., the sparsest cut [ARV09], the \( c \)-balanced graph separation problem [FHL08] and the minimum uncut [ACMM05] can be solved by SDP with \( m = \Omega(n^3) \). The optimal experiment design [VBW98], Haplotype frequencies estimation [HH06] and embedding of finite metric spaces into \( \ell^2 \) [LLR95] need to solve SDPs with \( m = \Omega(n^2) \).

VII. PRELIMINARY

A. Notations

a) Basic matrix notations.: We use \( \text{tr}[\cdot] \) to denote the trace of a matrix.

We use \( \| \cdot \|_2 \) and \( \| \cdot \|_F \) to denote the spectral norm and Frobenious norm of a matrix. We use \( \| \cdot \|_1 \) to denote the Schatten-1 norm of a matrix, i.e., \( \|A\|_1 = \text{tr}[(A^*A)^{1/2}] \).

We say a symmetric matrix \( A \in \mathbb{R}^{n \times n} \) is positive semidefinite (PSD, denoted as \( A \succeq 0 \)) if for any vector \( x \in \mathbb{R}^n \), \( x^\top Ax \geq 0 \). We say a symmetric matrix \( A \in \mathbb{R}^{n \times n} \) is positive definite (PD, denoted as \( A \succ 0 \)) if for any vector \( x \in \mathbb{R}^n \), \( x^\top Ax > 0 \). We use \( S_{++}^{n \times n} \) to denote the set of all \( n \)-by-\( n \) symmetric positive definite matrices and \( S_{\geq 0}^{n \times n} \) to denote the set of all \( n \)-by-\( n \) symmetric positive semi-definite matrices.

For a matrix \( A \in \mathbb{R}^{m \times n} \), we use \( \lambda(A) \in \mathbb{R}^n \) to denote the eigenvalues of \( A \).

For any vector \( v \in \mathbb{R}^n \), we use \( v[i] \) to denote the \( i \)-th largest entry of \( v \).
For a matrix $A \in \mathbb{R}^{m \times n}$, and subsets $S_1 \subseteq [m], S_2 \subseteq [n]$, we define $A_{S_1, S_2} \in \mathbb{R}^{|[S_1] \times |S_2|}$ to be the submatrix of $A$ that only has rows in $S_1$ and columns in $S_2$. We also define $A_{S_1, S_2} \in \mathbb{R}^{m \times |S_2|}$ to be the submatrix of $A$ that only has rows in $S_1$, and $A_{S_1, S_2} \in \mathbb{R}^{m \times |S_1|}$ to be the submatrix of $A$ that only has columns in $S_2$.

For two symmetric matrices $A, B \in \mathbb{R}^{n \times n}$, we say $A \preceq B$ (or equivalently, $B \succeq A$), if $B - A$ is a PSD matrix.

**Fact VII.1** (Spectral norm implies Loewner order). Let $A, B \in \mathbb{R}^{n \times n}$ be two symmetric PSD matrices. Then, for any $\epsilon \in (0, 1)$,

$$\left\| A^{-1/2}BA^{-1/2} - I \right\|_2 \leq \epsilon$$

implies

$$(1 - \epsilon)A \preceq B \preceq (1 + \epsilon)A.$$  

**Fact VII.2** (Trace property of matrix Loewner order). Given symmetric PSD matrices $A, B \in \mathbb{R}^n$. Suppose $(1 + \epsilon)^{-1} \cdot A \preceq \epsilon A \preceq (1 + \epsilon) \cdot A$, then

$$(1 + \epsilon)^{-1} \cdot \text{tr}[AB] \leq \text{tr}[\tilde{A}B] \leq (1 + \epsilon) \cdot \text{tr}[AB].$$

**Proof.** Consider the spectral decomposition of $B$: $B = \sum_{i=1}^n \lambda_i v_i v_i^\top$, where $\lambda_i \geq 0$. Then

$$\text{tr}[\tilde{A}B] = \text{tr}[\tilde{A} \cdot (\sum_{i=1}^n \lambda_i v_i v_i^\top)]$$

$$= \sum_{i=1}^n \lambda_i v_i^\top \tilde{A} v_i$$

$$\leq (1 + \epsilon) \cdot \sum_{i=1}^n \lambda_i v_i^\top A v_i$$

$$= \text{tr}[AB].$$

Similarly, $\text{tr}[\tilde{A}B] \geq (1 + \epsilon)^{-1} \cdot \text{tr}[AB]$. \qed

**b) Matrix related operations:** For two matrices $A, B \in \mathbb{R}^{m \times n}$, we define the matrix inner product $(A, B) := \text{tr}[A^\top B]$.

We use vec$[\cdot]$ to denote matrix vectorization: for a matrix $A \in \mathbb{R}^{m \times n}$, vec$[A]$ is defined to be vec$[A]_{(j-1)n+1} = A_{i,j}$ for any $i \in [m]$ and $j \in [n]$, i.e.,

$$\text{vec}[A] = \begin{bmatrix} A_{1,1} \\ \vdots \\ A_{m,n} \end{bmatrix} \in \mathbb{R}^{mn}.$$  

We use $\otimes$ to denote matrix Kronecker product: for matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$, $A \otimes B \in \mathbb{R}^{pm \times qn}$ is defined to be $(A \otimes B)_{(i-1)n+1,s(q-1)t+1} = A_{i,j} \cdot B_{s,t}$ for any $i \in [m]$, $j \in [n]$, $s \in [p]$, $t \in [q]$, i.e.,

$$A \otimes B = \begin{bmatrix} A_{1,1} \cdot B & A_{1,2} \cdot B & \cdots & A_{1,n} \cdot B \\ A_{2,1} \cdot B & A_{2,2} \cdot B & \cdots & A_{2,n} \cdot B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} \cdot B & A_{m,2} \cdot B & \cdots & A_{m,n} \cdot B \end{bmatrix} \in \mathbb{R}^{pm \times qn}.$$  

**Definition VII.3** (Stacking matrices). Let $A_1, A_2, \cdots, A_m \in \mathbb{R}^{n \times n}$ be $m$ symmetric matrices. We use $A \in \mathbb{R}^{m \times n^2}$ to denote the matrix that is constructed by stacking the $m$ vectorizations vec$[A_1], \cdots, \text{vec}[A_m]$ in $\mathbb{R}^{n^2}$ as rows of $A$, i.e.,

$$A := \begin{bmatrix} \text{vec}[A_1]^\top \\ \vdots \\ \text{vec}[A_m]^\top \end{bmatrix} \in \mathbb{R}^{m \times n^2}.$$  

**Fact VII.4.** For any $\epsilon_1, \epsilon_2 \in (0, 1/10)$. Let $\Sigma \in \mathbb{R}^{n \times n}$ be a diagonal matrix with non-negative entries and such that $\|\Sigma - I\|_F \leq \epsilon_1$. Let $X \in \mathbb{R}^{n \times n}$ such that $\|X\|_2 \leq \epsilon_2$. Then

$$\|\Sigma X \Sigma - X\|_F \leq 3 \cdot \epsilon_1 \cdot \epsilon_2.$$  

**Proof.** Denote $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$. We have

$$\|\Sigma X \Sigma - X\|_F$$

$$\leq \| (\Sigma - I) X (\Sigma - I) + (\Sigma - I) X + (\Sigma - I) \|_F$$

$$\leq \| (\Sigma - I) X \|_F + 2 \cdot \| (\Sigma - I) X \|_F$$

$$\leq 3 \cdot \| (\Sigma - I) X \|_F$$

$$\leq 3 \cdot (\epsilon_1^2 \cdot \| (\Sigma - I)^2 \|_2^{1/2})$$

$$\leq 3 \cdot (\epsilon_2 \cdot \sum_{i=1}^n (\sigma_i - 1)^2)^{1/2}$$

$$\leq 3 \cdot \epsilon_2 \cdot \sum_{i=1}^n (\sigma_i - 1)^2^{1/2}$$

$$\leq 3 \cdot \epsilon_2 \cdot \epsilon_1.$$  

where the second step uses triangle inequality, the third step uses $- I \preceq \Sigma - I \preceq I$, the fifth step uses $\|X\|_2 \leq \epsilon_2$, the penultimate step uses $\sigma_i \geq 0$, and the last step uses $\|\Sigma^2 - I\|_F \leq \epsilon_1$. \qed

**B. Tools: Woodbury identity**

**Fact VII.5** (Woodbury matrix identity, [Woo49], [Woo50]). For matrices $M \in \mathbb{R}^{n \times n}$, $U \in \mathbb{R}^{n \times k}$, $C \in \mathbb{R}^{k \times n}$, $V \in \mathbb{R}^{k \times n}$,

$$(M + UCV)^{-1} = M^{-1} - M^{-1}U(C^{-1} + VM^{-1}U)^{-1}VM^{-1}.$$  

**C. Tools: Properties of matrix operations**

**Fact VII.6** (Matrix inner product). For two matrices $A, B \in \mathbb{R}^{m \times n}$, we have $(A, B) = \text{tr}[A^\top B] = \text{vec}[A]^\top \text{vec}[B]$.

**Fact VII.7** (Basic properties of Kronecker product). The Kronecker product $\otimes$ satisfies the following properties.

1. For matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{k \times m}$, we have $(A \otimes B)^\top = A^\top \otimes B^\top \in \mathbb{R}^{mk \times mn}$.

2. For matrices $A \in \mathbb{R}^{n \times k}$, $B \in \mathbb{R}^{k \times m}$, $C \in \mathbb{R}^{n \times e}$, $D \in \mathbb{R}^{m \times d}$, we have $(A \otimes B) \cdot (C \otimes D) = (AC \otimes BD) \in \mathbb{R}^{ab \times cd}$.

**Fact VII.8** (Spectral properties of Kronecker product). The Kronecker product satisfies the following spectral properties.

1. For matrices $A, B$, if $A$ and $B$ are PSD matrices, then $A \otimes B$ is also PSD.
2) For PSD matrices $A, B \in \mathbb{R}^{n \times n}$ with $A \preceq B$, we have $A \otimes A \preceq B \otimes B$.

The following result is often used in SDP-related calculations.

**Fact VII.9** (Kronecker product and vector multiplication), Given $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times k}, C \in \mathbb{R}^{k \times l}, D \in \mathbb{R}^{l \times m}$, we have

1) $\vec{[ABC]} = (C^\top \otimes A) \cdot \vec{[B]}$.
   
   Note that $ABC \in \mathbb{R}^{m \times n}, C \otimes A \in \mathbb{R}^{ml \times nk}$, and $\vec{[B]} \in \mathbb{R}^{nk}$.

2) $\text{tr}[ABC]D = \vec{[D]}^\top \cdot (C \otimes A) \cdot \vec{[B]}$.
   
   Note that $ABC \in \mathbb{R}^{mn \times m}, \vec{[D]} \in \mathbb{R}^{ml}, C \otimes A \in \mathbb{R}^{ml \times nk}$, and $\vec{[B]} \in \mathbb{R}^{nk}$.

We state a standard fact for Kronecker product.

**Fact VII.10** (PSD property of Kronecker product). Given a matrix $A \in \mathbb{R}^{m \times n}$, let $S, \tilde{S} \in \mathbb{R}^{m \times n}$ be two PSD matrices. If matrix $H \in \mathbb{R}^{(S^{-1} \otimes S^{-1}) \cdot A^\top \cdot A \cdot S \cdot \tilde{S} \cdot \tilde{S}^{-1}}$. If matrix $H$ is non-negative for any $v \in \mathbb{R}^m$, both $H$ and $\tilde{H}$ are PSD matrices.

Let $b := \vec{[\sum_{i=1}^m v_i A_i]}^\top$. We have

$$
\begin{align*}
\| \vec{[\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2}]} \cdot (\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2})b \|_2^2 & = b^\top (\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2}) \cdot (\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2}) \cdot b \\
& = \mathbf{\alpha}^2 \cdot \| (\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2})b \|_2^2.
\end{align*}
$$

And

$$
\| (\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2})b \|_2^2 \geq \mathbf{\alpha}^2 \cdot \| (\tilde{S}^{-1/2} \otimes \tilde{S}^{-1/2})b \|_2^2.
$$

Combining Eqs. (6)-(9), we come to

$$
\frac{\alpha^2 \cdot v^\top Hv \leq v^\top \tilde{H}v \leq \alpha^2 \cdot v^\top Hv).
$$

Since $v$ can be arbitrarily chosen from $\mathbb{R}^m$, we complete the proof.

We state another fact for Kronecker product in below:

**Fact VII.11** (Kronecker product with equivalence for matrix norm). Given a matrix $A \in \mathbb{R}^{m \times n}$, vector $b \in \mathbb{R}^m$ and a parameter $\eta > 0$. Let $g(y, \eta) \in \mathbb{R}^m$ be defined as

$$
g(y, \eta)_i = \eta b_i - \text{tr}[S^{-1} A_i] \quad \forall i \in [m].
$$

Let $X \in \mathbb{R}^{n \times n}$ denote a matrix that

$$
(X, A_i) = \eta b_i \quad \forall i \in [m].
$$

Let $H := A(S^{-1} \otimes S^{-1})A^\top$. If matrix $S$ is a PSD matrix, then we have

$$
g(y, \eta)^\top H^{-1} g(y, \eta) = v^\top B^\top (BB^\top)^{-1} Bv,
$$

where $v := \vec{[S^{1/2}X S^{1/2} - I]} \in \mathbb{R}^n$ and $B \in \mathbb{R}^{m \times n}$ is a matrix that $i$-th row is $B_i = \vec{[S^{-1/2}A_i S^{-1/2}]} \in \mathbb{R}^n$.

**Proof.** We start with re-writing $g(y, \eta) \in \mathbb{R}^m$ as follows: for each $i \in [m]$

$$
g(y, \eta)_i = b_i \eta - \text{tr}[S^{-1} A_i] = \text{tr}[X A_i] - \text{tr}[S^{-1} A_i] = \text{tr}[X - S^{-1}] A_i = \text{tr}[S^{-1/2} X S^{1/2} - I] A_i = \text{tr}[S^{-1/2} X S^{1/2} - I] B_i.
$$

Thus, using the definition of $v$, we have

$$
g(y, \eta) = Bv.
$$

Our next step is to rewrite $H$ as follows: for each $i, j \in [m] \times [m]$

$$
H_{i,j} = \text{tr}[A_i S^{-1} A_j S^{-1}]
= \text{tr}[S^{-1/2} A_i S^{-1/2} \cdot S^{-1/2} A_j S^{-1/2}]
= \text{tr}[B_i B_j],
$$

which implies that $H = BB^\top$.
Thus, putting it all together, we have
\[ g(y, \eta)^\top H^{-1} g(y, \eta) = v^\top B^\top (BB^\top)^{-1} B v. \]
Therefore, we complete the proof. □

D. Tools: Fast matrix multiplication

We use \(T_{\text{mat}}(a, b, c)\) to denote the time of multiplying an \(a \times b\) matrix with another \(b \times c\) matrix. Fast matrix multiplication [Cop82], [Wil12], [LG14], [GU18], [CGLZZ20], [AW21] is a fundamental tool in theoretical computer science.

For \(k \in \mathbb{R}_+,\) we define \(\omega(k) \in \mathbb{R}_+\) to be the value such that \(\forall n \in \mathbb{N}_+,\ T_{\text{mat}}(n, n, n^k) = O(n^{\omega(k)}).\)

For convenience we define three special values of \(\omega(k)\). We define \(\omega\) to be the fast matrix multiplication exponent, i.e., \(\omega := \omega(1).\) We define \(\alpha \in \mathbb{R}_+\) to be the dual exponent of matrix multiplication, i.e., \(\omega(\alpha) = 2.\) We define \(\beta := \omega(2).\)

The following fact can be found in Lemma 3.6 of [JKL+20], also see [BCS97].

Fact VII.12 (Convexity of \(\omega(k)\)). The function \(\omega(k)\) is convex.

The following fact can be found in Lemma A.5 of [CLS19].

Fact VII.13 (Upper bound of \(T_{\text{mat}}(n, n, r)\)). For any \(r \leq n,\) we have that
\[ T_{\text{mat}}(n, n, r) \leq n^2 + r \cdot \frac{\sqrt{2}}{\pi n} \cdot n^2 - \frac{\alpha(r - 2)}{(r - 1)} \cdot n^\omega. \]

The following fact can be found in Lemma A.4 of [CLS19].

Fact VII.14 (Relation of \(\omega\) and \(\alpha\)). \(\frac{\omega}{1 - \alpha} - 1 \leq 0;\) that is, \(\omega + \alpha \leq 3.\)

The following fact can be found in Lemma 3.9-3.10 of [JKL+20].

Fact VII.15 (Upper bound of \(T_{\text{mat}}(m, n^2, m)\)). For any two integers \(m, n\) such that \(m \leq n,\)
\[ T_{\text{mat}}(m, n^2, m) \leq O(mn^{\omega + \alpha(1)}). \]

ACKNOWLEDGMENTS

The authors would like to thank FOCS 2022 anonymous reviewers. The authors would also like thank Haotian Jiang, Yin Tat Lee, Omri Weinstein, Hengjie Zhang, and Lichen Zhang for very useful discussions. This work is done while Baihe Huang is an undergraduate student at Peking University (advised by Zhao Song).

REFERENCES

[AACM05] Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. \(O(\sqrt{n \log n})\) approximation algorithms for min uncut, min 2cnf deletion, and directed cut problems. In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pages 573–581, 2005.

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In Proceedings of the 39th Annual ACM Symposium on Computing (STOC), 2007.

[AL05] 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 Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2016.
Advances in Neural Information Processing Systems
Proceedings of the 52nd
FOCS
Math-
2016 IEEE 57th Annual
Advances in Neural Information Processing Systems
V oprosy kibernetiki, Moscow
Soviet Math. Dokl
Optimization
USSR Computational Mathematics and Mathematical
Interior-point polyno-
arXiv preprint

[2012]

Dan Garber and Elad Hazan. Sublinear time algorithms for

[2019]

Noah Fleming, Pravesh Kothari, and Toniann Pitassi.

[2020]

Arun Jambulapati, Jerry Li, and Kevin Tian. Robust sub-gaussian
approximation and regression in input sparsity time. In Symposium on
Theory of Computing Conference (STOC), 2013.

deGJL07
Alexandre d’Aspremont, Laurent El Ghaoui, Michael I Jordan, and
Gert RG Lanckriet. A direct formulation for sparse pca using semidefinite programming. SIAM review, 49(3):434–448, 2007.

[2019]

[2019]

[2019]

Haotian Jiang, Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical
linear programs in the current matrix multiplication time. In Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS), 2011.

[2019]

[2022]

Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In Proceedings of the 39th international symposium on symbolic and algebraic computation (ISSAC), pages 296–303. ACM, 2014.

[2011]

Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. Combinatorica, 15(2):215–245, 1995.

[2019]

Aditi Laddha, Yin Tat Lee, and Santosh Vempala. Strong self-concordance and sampling. In Proceedings of the 32nd Annual ACM SIGSACT Symposium on Theory of Computing, pages 1212–1222, 2020.

[2011]

Yin Tat Lee and Swati Padmanabhan. An O((n+\epsilon^3)^3)-cost algorithm for semidefinite programs with diagonal constraints. In Conference on Learning Theory (COLT), Proceedings of Machine Learning Research. PMLR, 2020.

[2017]

Yin Tat Lee and He Sun. An spd-based algorithm for linearized spectral sparsification. In Proceedings of the 49th Annual ACM SIGSACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017, pages 678–687, 2017.

[2019]

Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt (rank) linear system solves. arXiv preprint arXiv:1910.08033, 2019.

[2015]

Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In 36th Annual IEEE Symposium of Foundations of Computer Science (FOCS), 2015.

[2019]

Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In Annual Conference on Learning Theory (COLT), 2019.

[2016]

Harisharun Narayanan. Randomized interior point methods for sampling and optimization. The Annals of Applied Probability, 26(1):597–641, 2016.

[2021]

Samuel B Hopkins, Prasad Raghavendra, and Ashish Shetty. Matrix discrepancy from quantum communication. arXiv preprint arXiv:2110.00099, 2021.

[2011]

Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP = PSPACE. Journal of the ACM (JACM), 2011.

[2020]

Arun Jambulapati, Yin Tat Lee, Jerry Li, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In FOCS, 2020.

[2020]

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 STOC, 2020.

[2020]

Arun Jambulapati, Jerry Li, and Kevin Tian. Robust sub-gaussian principal component analysis and width-independent schatten packing. Advances in Neural Information Processing Systems (NeurIPS), 33, 2020.

[2022]

Shunhua Jiang, Bento Natura, and Omri Weinstein. A faster interior-point method for sum-of-squares optimization. arXiv preprint arXiv:2202.08489, 2022.

[2021]

Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. In STOC, 2021.

[2011]

Rahul Jain and Penghui Yao. A parallel approximation algorithm for mixed packing and covering semidefinite programs. CoRR, abs/1201.6090, 2012.

[2000]

Leonid G Khachiyan. Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics, 20(1):53–72, 1980.

[2003]

Karik Krishnan and John E Mitchell. Properties of a cutting plane method for semidefinite programming. submitted for publication, 2003.

[2014]

David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. In Proceedings 35th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 1994.

[2006]

Leonid G Khachiyan, Sergei Pavlovich Tarasov, and I. I. Erlikh. The method of inscribed ellipsoids. Soviet Math. Dokl, 37(1):226–230, 1988.

[2014]

François Le Gall. Powers of tensors and fast matrix multiplication. In Proceedings of the 39th international symposium on symbolic and algebraic computation (ISSAC), pages 296–303. ACM, 2014.

[2020]

Aditi Laddha, Yin Tat Lee, and Santosh Vempala. Strong self-concordance and sampling. In Proceedings of the 32nd Annual ACM SIGSACT Symposium on Theory of Computing, pages 1212–1222, 2020.

[2011]

Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt (rank) linear system solves. arXiv preprint arXiv:1910.08033, 2019.

[2015]

Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In 36th Annual IEEE Symposium of Foundations of Computer Science (FOCS), 2015.

[2019]

Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In Annual Conference on Learning Theory (COLT), 2019.

[2016]

Harisharun Narayanan. Randomized interior point methods for sampling and optimization. The Annals of Applied Probability, 26(1):597–641, 2016.

[2021]

Yu Nesterov. Polynomial-time iterative methods in linear and quadratic programming. Voprosy kibernetiki, Moscow, pages 102–125, 1988.

[2022]

YY Nesterov. Polynomial methods in the linear and quadratic-programming. Soviet Journal of Computer and Systems Sciences, 26(5):98–101, 1988.

[1980]

Don Coppersmith. Rapid multiplication of rectangular matrices. Mathematics of operations research, 1(2):481–500, 1976.

[1988a]

Yu Nesterov. Polynomial-time iterative methods in linear and quadratic programming. Voprosy kibernetiki, Moscow, pages 102–125, 1988.

[1988b]

YY Nesterov. Polynomial methods in the linear and quadratic-programming. Soviet Journal of Computer and Systems Sciences, 26(5):98–101, 1988.

[1989]

Yuri Nesterov and Arkadi Nemirovski. Self-concordant functions and polynomial time methods in convex programming. preprint, central economic & mathematical institute, ussr acad. Sci. Moscow, USSR, 1989.

[2019]

Yuri Nesterov and Arkadi Nemirovski. Conic formulation of a convex programming problem and duality. Optimization Methods and Software, 1(2):95–115, 1992.

[1994]

Yuri Nesterov and Arkadi Nemirovski. Interior-point polynomial algorithms in convex programming, volume 13. Siam, 1994.
[PV21] Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication. In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 504–521. SIAM, 2021.

[Rent01] James Renegar. A Mathematical View of Interior-point Methods in Convex Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[RSL18] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In Advances in Neural Information Processing Systems (NeurIPS), pages 10877–10887, 2018.

[RSW16] Ilya Razenshteyn, Zhao Song, and David P Woodruff. Weighted low rank approximations with provable guarantees. In Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, pages 250–263, 2016.

[Sar06] Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2006.

[Sho77] Naum Z Shor. Cut-off method with space extension in convex programming problems. Cybernetics and systems analysis, 13(1):94–96, 1977.

[SWZ17] Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise $\ell_1$-norm error. In Proceedings of the 49th Annual Symposium on the Theory of Computing (STOC). ACM, 2017.

[SWZ19] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In ACM-SIAM Symposium on Discrete Algorithms (SODA), 2019.

[SY21] Zhao Song and Zheng Yu. Oblivious sketching-based central path method for solving linear programming. In ICML, 2021.

[Vai89a] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 338–343, 1989.

[Vai89b] Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In 30th Annual Symposium on Foundations of Computer Science (FOCS), pages 332–337. IEEE, 1989.

[VBW98] Lieven Vandenberghe, Stephen Boyd, and Shao-Po Wu. Determinant maximization with linear matrix inequality constraints. SIAM journal on matrix analysis and applications, 19(2):499–533, 1998.

[Will2] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Proceedings of the forty-fourth annual ACM symposium on Theory of Computing (STOC), pages 887–898. ACM, 2012.

[Woo49] Max A Woodbury. The stability of out-input matrices. Chicago, IL, 9, 1949.

[Woo50] Max A. Woodbury. Inverting modified matrices. Princeton University, Princeton, N. J., 1950. Statistical Research Group, Memo. Rep. no. 42.

[YN76] David B Yudin and Arkadi S Nemirovski. Evaluation of the information complexity of mathematical programming problems. Ekonomika i Matematicheskie Metody, 12:128–142, 1976.

[YTF+19] Alp Yurtsever, Joel A. Tropp, Olivier Fercoq, Madeleine Udell, and Volkan Cevher. Scalable semidefinite programming, 2019.