A Robust IPM Framework to Solve SDP

Tianyi Zhou

Abstract

This is an explanatory paper for [HJS+22]. In [HJS+22], they introduce a new robust interior point method analysis for semidefinite programming (SDP). Under this new framework, they improve the running time of semidefinite programming (SDP) with variable size $n \times n$ and m constraints up to ϵ accuracy. They show that for the case $m = \Omega(n^2)$, we can solve SDPs in m^ω time. This suggests solving SDP is nearly as fast as solving the linear system with equal number of variables and constraints.

In this paper, We give a detailed explanation of their new robust interior point method analysis for semidefinite programming (SDP). We show how their IPM algorithm is combined with the logarithmic barrier. Particularly, we show how this paper does the low-rank update for the slack matrix and the Hessian matrix efficiently.

Contents

1 Introduction

Semidefinite programming (SDP) is a valuable tool for optimizing a linear objective function within the positive semidefinite (PSD) cone's intersection with an affine space. Its versatility makes it particularly intriguing for both theoretical and practical applications, including operations research, machine learning, and theoretical computer science. Semidefinite programming problems are commonly used to model or approximate a vast range of problems. In machine learning, SDP has a diverse array of applications, including adversarial machine learning, learning structured distribution, sparse Principal Component Analysis (PCA)[AW08, dEGJL07], and robust learning[DKK+16, DHL19, JLT20]. Recent studies in machine learning have leveraged SDP to solve these problems efficiently and effectively.

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 [JJUW11], robust learning and estimation [CG18, CDG19, CDGW19], graph sparsification [LS17], algorithmic discrepancy and rounding [BDG16, BG17, Ban19], sum of squares optimization [BS16, FKP19], terminal embeddings [CN21], and matrix discrepancy [HRS21].

SDP is formally defined as follows:

Definition 1.1 (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:*

$$\max_{X\in\mathbb{R}^{n\times n}} \ \langle C, X\rangle \text{ subject to } \ \langle A_i, X\rangle = b_i, \quad \forall i \in [m], \ X \succeq 0, \tag{1}$$

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 $C, A_1, \cdots, A_m$ 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 (with logarithmic accuracy dependence in the running time). The first line of work is using the cutting plane method, such as [Sho77, YN76, Kha80, KTE88, NN89, Vai89a, BV02, KM03, LSW15, 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, 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, [HJS+22] ask the following question:

Can we efficiently solve SDP <u>without</u> computing exact gradient, Hessian, and Newton steps?

[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 $\sum_{i,j} A_{ij}X_{ij} = \sum_{i,j} A_{ij}X_{ji} = \sum_{i,j}(A^\top)_{ij}X_{ij}$ since X is symmetric, so we can replace A with $(A + A^\top)/2$.

They answer the above question by introducing new framework for both IPM analysis and variable maintenance. For IPM analysis, they build a robust IPM framework for arbitrary barrier functions that supports errors in computing gradient, Hessian, and Newton steps. For variable maintenance, they 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[2]. 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? They 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.

Our objective is to present a comprehensive account of the this robust interior point technique for semidefinite programming (SDP). We meticulously elucidate the manner in which the IPM approach is fused with the logarithmic barrier. Furthermore, we expound on this paper's proficient low-rank update procedure for both the slack and Hessian matrices.

1.1 Results

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

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

Remark 1.3. *For any $m \geq n^{2-0.5/\omega} \approx n^{1.79}$ with current $\omega \approx 2.37286$ [Wil12, LG14, AW21], our algorithm runs faster than [JKL+20].*

2 Preliminary

2.1 Notations

Basic matrix notations. For a square matrix X, we use $\mathrm{tr}[X]$ to denote the trace of X.

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

[2] We 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.

[3] We use $O^*(\cdot)$ to hide $n^{o(1)}$ and $\log^{O(1)}(mn/\epsilon)$ factors, and $\widetilde{O}(\cdot)$ to hide $\log^{O(1)}(mn/\epsilon)$ factors.

We say a symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD, denoted as $A \succeq 0$) if for any vector $x \in \mathbb{R}^n$, $x^\top A x \geq 0$. 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 $x \in \mathbb{R}^n$, $x^\top A x > 0$.

We define $\mathbb{S}^{n \times n}_{\succ 0}$ to be the set of all n-by-n symmetric positive definite matrices.

Let us define $\mathbb{S}^{n \times n}_{\succeq 0}$ to be 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 $A_{S_1,:} \in \mathbb{R}^{|S_1| \times n}$ to be the submatrix of A that only has rows in S_1, and $A_{:,S_2} \in \mathbb{R}^{m \times |S_2|}$ 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.

Next, we state some useful fact that will be used in the paper.

Fact 2.1 (Spectral norm implies Loewner order)**.** *Let $A, B \in \mathbb{R}^{n \times n}$ be two symmetric PSD matrices. Then, for any $\epsilon \in (0, 1)$,*

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

implies

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

Fact 2.2 (Basic properties of Kronecker product)**.** *The Kronecker product $\otimes$ satisfies the following properties.*

1. *For matrices $A \in \mathbb{R}^{a \times n}$ and $B \in \mathbb{R}^{b \times m}$, we have $(A \otimes B)^\top = A^\top \otimes B^\top \in \mathbb{R}^{nm \times ab}$.*
2. *For matrices $A \in \mathbb{R}^{a \times n}$, $B \in \mathbb{R}^{b \times m}$, $C \in \mathbb{R}^{n \times c}$, $D \in \mathbb{R}^{m \times d}$, we have $(A \otimes B) \cdot (C \otimes D) = (AC \otimes BD) \in \mathbb{R}^{ab \times cd}$.*

Fact 2.3 (Woodbury matrix identity)**.** *Given two integers n and k. Let $n \geq k$. For square matrix $A \in \mathbb{R}^{n \times n}$, tall matrix $B \in \mathbb{R}^{n \times k}$, square matrix $C \in \mathbb{R}^{k \times k}$, fat matrix $D \in \mathbb{R}^{k \times n}$,*

$$(A + BCD)^{-1} = A^{-1} - A^{-1} B (C^{-1} + D A^{-1} B)^{-1} D A^{-1}.$$

3 Robust IPM to Solve SDP

This section delves into the methods suggested to overcome the bottleneck of m^ω cost per iteration in [JKL+20]. The first bottleneck pertains to reducing the computational load of inverting the Hessian matrix in each run-through by utilizing the Hessian inverse calculated in the previous iteration. The approach involves low-rank updates through which we reveal that the difference between the inverses of Hessian matrices calculated using Kronecker products is low-rank. Consequently, we illustrate how we efficiently update the Hessian inverse utilizing the Woodbury identity in this paper while thoroughly describing the process of computing the Hessian inverse through Woodbury identity and fast matrix rectangular multiplication in Section 6.1. Additionally, we present an ameliorated amortization scheme for PSD matrices which outclasses the earlier m^ω amortized cost, as our proof sketch in Section B demonstrates. Finally, we represent the Hessian matrix in Kronecker product form as shown below:

$$H = \mathsf{A} \cdot (S^{-1} \otimes S^{-1}) \cdot \mathsf{A}^\top.$$

The subject of low-rank approximation of Kronecker product is an intriguing one that has gained attention, as evident in the study conducted by [SWZ19]. We will expound on how the utilization of low-rank techniques in updating the slack matrix can bring about an update of the Hessian matrix that involves the Kronecker product. To achieve this, we will implement a three-step approach for constructing the low-rank update of H. Our initial proposal entails the implementation of a general robust SDP Algorithm 1.

Algorithm 1 Informal version of Alg. 3. An implementation of GeneralRobustSDP

1: **procedure** SolveSDP($\mathsf{A} \in \mathbb{R}^{m \times n^2}$, $b \in \mathbb{R}^m$, $C \in \mathbb{R}^{n \times n}$)
2:     **for** $t = 1 \to T$ **do** ▷ $T = \widetilde{O}(\sqrt{n})$
3:         $\eta^{\mathrm{new}} \leftarrow \eta \cdot (1 + 1/\sqrt{n})$
4:         $g_{\eta^{\mathrm{new}}}(y)_j \leftarrow \eta^{\mathrm{new}} \cdot b_j - \mathrm{tr}[S^{-1} \cdot A_j], \ \ \forall j \in m$ ▷ Gradient computation
5:         $\delta_y \leftarrow -\widetilde{H}^{-1} \cdot g_{\eta^{\mathrm{new}}}(y)$ ▷ Compute Newton step
6:         $y^{\mathrm{new}} \leftarrow y + \delta_y$ ▷ Update dual variables
7:         $S^{\mathrm{new}} \leftarrow \sum_{i \in [m]} (y^{\mathrm{new}})_i A_i - C$ ▷ Compute slack matrix
8:         Compute $V_1, V_2 \in \mathbb{R}^{n \times r_t}$ such that $\widetilde{S}^{\mathrm{new}} = \widetilde{S} + V_1 \cdot V_2^\top$ ▷ Section 4
9:         Compute $V_3, V_4 \in \mathbb{R}^{n \times r_t}$ such that $(\widetilde{S}^{\mathrm{new}})^{-1} = (\widetilde{S})^{-1} + V_3 \cdot V_4^\top$ ▷ Section 5
10:         Compute $\mathsf{A}Y_1, \mathsf{A}Y_2 \in \mathbb{R}^{m \times n r_t}$ such that $\widetilde{H}^{\mathrm{new}} = \widetilde{H} + (\mathsf{A}Y_1) \cdot (\mathsf{A}Y_2)^\top$ ▷ Section 6
11:         $(\widetilde{H}^{\mathrm{new}})^{-1} \leftarrow \widetilde{H}^{-1} + \text{low-rank update}$ ▷ Sec. 6.1
12:         $y \leftarrow y^{\mathrm{new}}$, $S \leftarrow S^{\mathrm{new}}$, $\widetilde{S} \leftarrow \widetilde{S}^{\mathrm{new}}$, $\widetilde{H}^{-1} \leftarrow (\widetilde{H}^{\mathrm{new}})^{-1}$ ▷ Update variables
13:     **end for**
14: **end procedure**

In Line 8, we approximate the slack matrix. In Line 9, we approximate the inverse of slack matrix. In Line 10, we approximate the Hessian matrix. In Line 11, we approximate the inverse of Hessian matrix.

4 Low-rank Update of Slack Matrix

We utilize an estimated slack matrix that produces a low-rank update. In the t-th iteration of Algorithm 1, the current estimated slack matrix is represented as $\widetilde{S}$ while the new accurate slack matrix is denoted as S^{new}. Both matrices are implemented to obtain the new approximate slack matrix, $\widetilde{S}^{\mathrm{new}}$. We outline $Z = (S^{\mathrm{new}})^{-1/2} \widetilde{S} (S^{\mathrm{new}})^{-1/2} - I$ to capture differences in the slack matrix. The spectral decomposition of Z is computed: $Z = U \cdot \mathrm{diag}(\lambda) \cdot U^\top$. It has been illustrated in [JKL+20] that only certain eigenvalues of Z are meaningful, catalogued as $\lambda_1, \ldots, \lambda_{r_t}$, and the remaining eigenvalues are designated as zero. Thus, we acquire a low-rank approximation of Z: $\widetilde{Z} = U \cdot \mathrm{diag}(\widetilde{\lambda}) \cdot U^\top$, where $\widetilde{\lambda} = [\lambda_1, \cdots, \lambda_{r_t}, 0, \ldots, 0]^\top$. We now implement $\widetilde{Z}$ to update the estimated slack matrix via a low-rank matrix:

$$\widetilde{S}^{\mathrm{new}} = \widetilde{S} + (S^{\mathrm{new}})^{1/2} \cdot \widetilde{Z} \cdot (S^{\mathrm{new}})^{1/2} = \widetilde{S} + V_1 \cdot V_2^\top,$$

with V_1 and V_2 having a size of $n \times r_t$. As $\widetilde{Z}$ constitutes a decent approximation of Z, $\widetilde{S}^{\mathrm{new}}$ constitutes a PSD approximation of S^{new}. Thus, it assures that y remains in proximity to the

Algorithm 2 Low Rank Slack Update

1: **procedure** LowRankSlackUpdate($S^{\text{new}}, \widetilde{S}$)
2: $\triangleright$ $S^{\text{new}}, \widetilde{S} \in \mathbb{S}^{n \times n}_{\geq 0}$ are positive definite matrices
3: $\epsilon_S \leftarrow 10^{-5}$ $\triangleright$ Spectral approximation constant
4: $Z^{\text{mid}} \leftarrow (S^{\text{new}})^{-1/2} \cdot \widetilde{S} \cdot (S^{\text{new}})^{-1/2} - I_n$
5: Compute spectral decomposition $Z^{\text{mid}} = U \cdot \mathrm{diag}(\lambda) \cdot U^\top$
6: $\triangleright$ $\lambda = [\lambda_1, \cdots, \lambda_n]^\top \in \mathbb{R}^n$ are the eigenvalues of Z^{mid}, and $U \in \mathbb{R}^{n \times n}$ is orthogonal
7: Let $\pi : [n] \to [n]$ be a sorting permutation such that $|\lambda_{\pi(i)}| \geq |\lambda_{\pi(i+1)}|$
8: **if** $|\lambda_{\pi(1)}| \leq \epsilon_S$ **then**
9: $\widetilde{S}^{\text{new}} \leftarrow \widetilde{S}$
10: **else**
11: $r \leftarrow 1$
12: **while** $r \leq n/2$ and $(|\lambda_{\pi(2r)}| > \epsilon_S$ or $|\lambda_{\pi(2r)}| > (1 - 1/\log n)|\lambda_{\pi(r)}|)$ **do**
13: $r \leftarrow r + 1$
14: **end while**
15: $(\lambda^{\text{new}})_{\pi(i)} \leftarrow \begin{cases} 0, & \text{if } i = 1, 2, \cdots, 2r; \\ \lambda_{\pi(i)}, & \text{otherwise.} \end{cases}$
16: $L \leftarrow \mathrm{supp}(\lambda^{\text{new}} - \lambda)$ $\triangleright$ $|L| = 2r$
17: $V_1 \leftarrow ((S^{\text{new}})^{1/2} \cdot U \cdot \mathrm{diag}(\lambda^{\text{new}} - \lambda))_{:,L}$ $\triangleright$ $V_1 \in \mathbb{R}^{n \times 2r}$
18: $V_2 \leftarrow ((S^{\text{new}})^{1/2} \cdot U)_{:,L}$ $\triangleright$ $V_2 \in \mathbb{R}^{n \times 2r}$
19: $\triangleright$ $V_1 \cdot V_2^\top = (S^{\text{new}})^{1/2} \cdot U \cdot \mathrm{diag}(\lambda^{\text{new}} - \lambda) \cdot U^\top \cdot (S^{\text{new}})^{1/2}$
20: **end if**
21: **return** $\widetilde{S}^{\text{new}}$
22: **end procedure**

central path. Furthermore, we have denoted $\widetilde{S}$ and S^{new}, respectively, as the current approximate slack matrix and the new exact slack matrix throughout this process.

4.1 Approximate slack maintenance

The following lemma gives a closed-form formula for the updated $\widetilde{S}^{\text{new}}$ in each iteration.

Lemma 4.1 (Closed-form formula of slack update)**.** *In each iteration of Algorithm 3, the update of the slack variable $\widetilde{S}^{\text{new}}$ (on Line 18) satisfies*

$$\widetilde{S}^{\text{new}} = \widetilde{S} + (S^{\text{new}})^{1/2} \cdot U \cdot \mathrm{diag}(\lambda^{\text{new}} - \lambda) \cdot U^\top \cdot (S^{\text{new}})^{1/2},$$

where $\widetilde{S}$ is the slack variable in previous iteration, and $U, \lambda, \lambda^{\text{new}}$ are defined in Algorithm 2.

Moreover, it implies that $\widetilde{S}^{\text{new}}$ is a symmetric matrix in each iteration.

Proof. From Line 17 and 18 of Algorithm 2 we have $V_1 \cdot V_2^\top = (S^{\text{new}})^{1/2} \cdot U \cdot \mathrm{diag}(\lambda^{\text{new}} - \lambda) \cdot U^\top \cdot (S^{\text{new}})^{1/2}$. Therefore

$$\widetilde{S}^{\text{new}} = \widetilde{S} + V_1 \cdot V_2^\top = \widetilde{S} + (S^{\text{new}})^{1/2} \cdot U \cdot \mathrm{diag}(\lambda^{\text{new}} - \lambda) \cdot U^\top \cdot (S^{\text{new}})^{1/2}.$$

In the first iteration, we have $\widetilde{S} = S$ which is a symmetric matrix.

By the definition of V_1, V_2, we know that $V_1 \cdot V_2^\top$ is symmetric. Hence, S^{new} is also symmetric in each iteration. $\square$

The following lemma proves that we always have $\widetilde{S} \approx S$ throughout the algorithm.

Lemma 4.2 (Approximate Slack)**.** *In each iteration of Algorithm 3, the approximate slack variable $\widetilde{S}$ satisfies that $\alpha_S^{-1} S \preceq \widetilde{S} \preceq \alpha_S S$, where $\alpha_S = 1 + 10^{-5}$.*

Proof. Notice that

$$\begin{aligned}
\widetilde{S}^{\text{new}} &= \widetilde{S} + (S^{\text{new}})^{1/2} \cdot U \cdot \text{diag}(\lambda^{\text{new}} - \lambda) \cdot U^\top \cdot (S^{\text{new}})^{1/2} \\
&= \Big(S^{\text{new}} + (S^{\text{new}})^{1/2} Z^{\text{mid}} (S^{\text{new}})^{1/2}\Big) + (S^{\text{new}})^{1/2} \cdot U \cdot \text{diag}(\lambda^{\text{new}} - \lambda) \cdot U^\top \cdot (S^{\text{new}})^{1/2} \\
&= S^{\text{new}} + (S^{\text{new}})^{1/2} \cdot U \cdot \text{diag}(\lambda^{\text{new}}) \cdot U^\top \cdot (S^{\text{new}})^{1/2},
\end{aligned}$$

where the first step comes from Lemma 4.1, the second step comes from definition $Z^{\text{mid}} = (S^{\text{new}})^{-1/2} \cdot \widetilde{S} \cdot (S^{\text{new}})^{-1/2} - I$ (Line 4 of Algorithm 2), and the final step comes from $Z^{\text{mid}} = U \cdot \text{diag}(\lambda_1, \cdots, \lambda_n) \cdot U^\top$ (Line 5 of Algorithm 2).

By Line 15 of Algorithm 2 we have $(\lambda^{\text{new}})_i \leq \epsilon_S$ for all $i \in [n]$, so

$$\Big\|(S^{\text{new}})^{-1/2} \cdot \widetilde{S}^{\text{new}} \cdot (S^{\text{new}})^{-1/2} - I\Big\|_2 = \Big\|U \cdot \text{diag}(\lambda^{\text{new}}) \cdot U^\top\Big\|_2 \leq \epsilon_S.$$

This implies that for $\alpha_S = 1 + \epsilon_S$, by Fact 2.1, in each iteration of Algorithm 3 the slack variable $\widetilde{S}$ satisfies $\alpha_S^{-1} S \preceq \widetilde{S} \preceq \alpha_S S$. $\square$

5 Low-rank Update of Inverse of Slack

Using Woodbury identity, we can show that

$$(\widetilde{S}^{\text{new}})^{-1} = (\widetilde{S} + V_1 \cdot V_2^\top)^{-1} = \widetilde{S}^{-1} + V_3 V_4^\top,$$

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

5.1 Approximate Hessian Inverse Maintenance

The following lemma shows that the maintained matrix G equals to the inverse of approximate Hessian.

Lemma 5.1 (Close-form formula for Hessian inverse)**.** *In each iteration of Algorithm 3, we have $G = \widetilde{H}^{-1} \in \mathbb{R}^{m \times m}$, where $\widetilde{H} := \mathsf{A} \cdot (\widetilde{S}^{-1} \otimes \widetilde{S}^{-1}) \cdot \mathsf{A}^\top \in \mathbb{R}^{m \times m}$.*

Proof. We prove this lemma by induction.

In the beginning of the algorithm, the initialization of G (Line 7 of Algorithm 3) satisfies the formula $G = \widetilde{H}^{-1}$.

Assume the induction hypothesis that $G = \widetilde{H}^{-1}$ in the beginning of each iteration, next we will prove that $G^{\text{new}} = (\widetilde{H}^{\text{new}})^{-1}$. Note that G^{new} is updated on Line 23 of Algorithm 3. And $\widetilde{H}^{\text{new}} := \mathsf{A} \cdot ((\widetilde{S}^{\text{new}})^{-1} \otimes (\widetilde{S}^{\text{new}})^{-1}) \cdot \mathsf{A}^\top \in \mathbb{R}^{m \times m}$, where $\widetilde{S}^{\text{new}}$ is updated on Line 18 of Algorithm 3.

We first compute $(\widetilde{S}^{\text{new}})^{-1} \in \mathbb{R}^{n \times n}$:

$$\begin{aligned}
(\widetilde{S}^{\text{new}})^{-1} &= (\widetilde{S} + V_1 V_2^\top)^{-1} \\
&= \widetilde{S}^{-1} - \widetilde{S}^{-1} V_1 \cdot (I + V_2^\top \widetilde{S}^{-1} V_1)^{-1} \cdot V_2^\top \widetilde{S}^{-1} \\
&= \widetilde{S}^{-1} + V_3 \cdot V_4^\top, \qquad (2)
\end{aligned}$$

where the reason of the first step is $\widetilde{S}^{\mathrm{new}} = \widetilde{S} + V_1 V_2^\top$ (Line 18 of Algorithm 3), the second step follows from Woodbury identity (Fact 2.3), and the third step follows from $V_3 = -\widetilde{S}^{-1} V_1 (I + V_2^\top \widetilde{S}^{-1} V_1)^{-1} \in \mathbb{R}^{n \times r_t}$ and $V_4 = \widetilde{S}^{-1} V_2 \in \mathbb{R}^{n \times r_t}$ (Line 19 and 20 of Algorithm 3).

We then compute a close-form formula of $(\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1} \in \mathbb{R}^{n^2 \times n^2}$:

$$\begin{aligned}
& (\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1} \\
= & (\widetilde{S}^{-1} + V_3 V_4^\top) \otimes (\widetilde{S}^{-1} + V_3 V_4^\top) \\
= & \widetilde{S}^{-1} \otimes \widetilde{S}^{-1} + \widetilde{S}^{-1} \otimes (V_3 V_4^\top) + (V_3 V_4^\top) \otimes \widetilde{S}^{-1} + (V_3 V_4^\top) \otimes (V_3 V_4^\top) \\
= & \widetilde{S}^{-1} \otimes \widetilde{S}^{-1} + (\widetilde{S}^{-1/2} \otimes V_3) \cdot (\widetilde{S}^{-1/2} \otimes V_4^\top) + (V_3 \otimes \widetilde{S}^{-1/2}) \cdot (V_4^\top \otimes \widetilde{S}^{-1/2}) \\
& + (V_3 \otimes V_3) \cdot (V_4^\top \otimes V_4^\top) \\
= & \widetilde{S}^{-1} \otimes \widetilde{S}^{-1} + Y_1 Y_2^\top,
\end{aligned} \tag{3}$$

where the first step follows from Eq. (2), the second step follows from linearity of Kronecker product, the third step follows from mixed product property of Kronecker product (Part 2 of Fact 2.2), the fourth step follows from $Y_1 = [(\widetilde{S}^{-1/2} \otimes V_3), (V_3 \otimes \widetilde{S}^{-1/2}), (V_3 \otimes V_3^\top)] \in \mathbb{R}^{n^2 \times (2nr_t + r_t^2)}$ and $Y_2 = [(\widetilde{S}^{-1/2} \otimes V_4), (V_4 \otimes \widetilde{S}^{-1/2}), (V_4 \otimes V_4^\top)] \in \mathbb{R}^{n^2 \times (2nr_t + r_t^2)}$ (Line 21 and 22 of Algorithm 3), and the transpose of Kronecker product (Part 1 of Fact 2.2).

Thus we can compute $(\widetilde{H}^{\mathrm{new}})^{-1} \in \mathbb{R}^{m \times m}$ as follows:

$$\begin{aligned}
(\widetilde{H}^{\mathrm{new}})^{-1} & = \big(\mathsf{A} \cdot \big((\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1}\big) \cdot \mathsf{A}^\top\big)^{-1} \\
& = \Big(\mathsf{A} \cdot (\widetilde{S}^{-1} \otimes \widetilde{S}^{-1}) \cdot \mathsf{A}^\top + \mathsf{A} \cdot Y_1 Y_2^\top \cdot \mathsf{A}^\top\Big)^{-1} \\
& = G - G \cdot \mathsf{A} Y_1 \cdot (I + Y_2^\top \mathsf{A}^\top \cdot \mathsf{A} Y_1)^{-1} \cdot Y_2^\top \mathsf{A}^\top \cdot G \\
& = G^{\mathrm{new}},
\end{aligned} \tag{4}$$

where the first step follows from the definition of $\widetilde{H}^{\mathrm{new}}$, the second step follows from Eq. (3), the third step follows from Woodbury identity (Fact 2.3) and the induction hypothesis that $G = \widetilde{H}^{-1} = (\mathsf{A} \cdot (\widetilde{S}^{-1} \otimes \widetilde{S}^{-1}) \cdot \mathsf{A}^\top)^{-1} \in \mathbb{R}^{m \times m}$, the fourth step follows from the definition of G^{new} on Line 23 of Algorithm 3.

The proof is then completed. $\square$

6 Low-rank Update of Hessian

Using the linearity and the mixed product property (Part 2 of Fact 2.2) of Kronecker product, we can find a low-rank update to $(\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1}$. More precisely, we can rewrite $(\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1}$ as follows:

$$(\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1} = (\widetilde{S}^{-1} + V_3 V_4^\top) \otimes (\widetilde{S}^{-1} + V_3 V_4^\top) = \widetilde{S}^{-1} \otimes \widetilde{S}^{-1} + \mathcal{S}_{\mathrm{diff}}.$$

The term $\mathcal{S}_{\mathrm{diff}}$ is the difference that we want to compute, we can show

$$\begin{aligned}
\mathcal{S}_{\mathrm{diff}} & = \widetilde{S}^{-1} \otimes (V_3 V_4^\top) + (V_3 V_4^\top) \otimes \widetilde{S}^{-1} + (V_3 V_4^\top) \otimes (V_3 V_4^\top) \\
& = (\widetilde{S}^{-1/2} \otimes V_3) \cdot (\widetilde{S}^{-1/2} \otimes V_4^\top) + (V_3 \otimes \widetilde{S}^{-1/2}) \cdot (V_4^\top \otimes \widetilde{S}^{-1/2}) + (V_3 \otimes V_3) \cdot (V_4^\top \otimes V_4^\top) \\
& = Y_1 \cdot Y_2^\top
\end{aligned}$$

where Y_1 and Y_2 both have size $n^2 \times nr_t$. In this way we get a low-rank update to the Hessian:

$$\widetilde{H}^{\mathrm{new}} = \mathsf{A} \cdot ((\widetilde{S}^{\mathrm{new}})^{-1} \otimes (\widetilde{S}^{\mathrm{new}})^{-1}) \cdot \mathsf{A}^\top = \widetilde{H} + (\mathsf{A} Y_1) \cdot (\mathsf{A} Y_2)^\top.$$

6.1 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 $\widetilde{H}^{-1}$:

$$(\widetilde{H}^{\text{new}})^{-1} = \left(\widetilde{H} + (\mathsf{A}Y_1) \cdot (\mathsf{A}Y_2)^\top\right)^{-1} = \widetilde{H}^{-1} - \widetilde{H}^{-1} \cdot \mathsf{A}Y_1 \cdot (I + Y_2^\top \mathsf{A}^\top \cdot \mathsf{A}Y_1)^{-1} \cdot Y_2^\top \mathsf{A}^\top \cdot \widetilde{H}^{-1}$$

The second term in the above equation has rank nr. Thus $(\widetilde{H}^{\text{new}})^{-1} - \widetilde{H}^{-1}$ has a rank nr decomposition. To compute $(\widetilde{H}^{\text{new}})^{-1}$ in each iteration, we first compute $\mathsf{A}Y_1, \mathsf{A}Y_2 \in \mathbb{R}^{m \times nr_t}$ and multiply it with $\widetilde{H}^{-1} \in \mathbb{R}^{m \times m}$ to get $\widetilde{H}^{-1} \cdot \mathsf{A}Y_1, \widetilde{H}^{-1} \cdot \mathsf{A}Y_2 \in \mathbb{R}^{m \times nr_t}$. Then we compute $I + (Y_2^\top \mathsf{A}^\top) \cdot (\mathsf{A}Y_1) \in \mathbb{R}^{nr_t \times nr_t}$ and find its inverse $(I + Y_2^\top \mathsf{A}^\top \cdot \mathsf{A}Y_1)^{-1} \in \mathbb{R}^{nr_t \times nr_t}$. Finally, we multiply $\widetilde{H}^{-1} \cdot \mathsf{A}Y_1, \widetilde{H}^{-1} \cdot \mathsf{A}Y_2 \in \mathbb{R}^{m \times nr_t}$ and $(I + Y_2^\top \mathsf{A}^\top \cdot \mathsf{A}Y_1)^{-1} \in \mathbb{R}^{nr_t \times nr_t}$ together to obtain $(\widetilde{H})^{-1}\mathsf{A}Y_1 \cdot (I + Y_2^\top \mathsf{A}^\top \cdot \mathsf{A}Y_1)^{-1} \cdot Y_2^\top \mathsf{A}^\top (\widetilde{H})^{-1} \in \mathbb{R}^{m \times m}$, as desired. Using fast matrix multiplication in each aforementioned step, the total computation cost is bounded by

$$O(\mathcal{T}_{\text{mat}}(m, n^2, nr_t) + \mathcal{T}_{\text{mat}}(m, m, nr_t) + (nr_t)^\omega). \tag{5}$$

References

[Ans00] Kurt M Anstreicher. The volumetric barrier for semidefinite programming. *Mathematics of Operations Research*, 2000.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 2009.

[AW08] Arash A Amini and Martin J Wainwright. High-dimensional analysis of semidefinite relaxations for sparse principal components. In *2008 IEEE International Symposium on Information Theory (ISIT)*, pages 2454–2458. IEEE, 2008.

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[Ban19] Nikhil Bansal. On a generalization of iterated and randomized rounding. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2019.

[BDG16] Nikhil Bansal, Daniel Dadush, and Shashwat Garg. An algorithm for komlós conjecture matching banaszczyk. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2016.

[BG17] Nikhil Bansal and Shashwat Garg. Algorithmic discrepancy beyond partial coloring. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2017.

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020.

[Bra20] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

[Bra21] Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 1–13. SIAM, 2021.

[BS16] Boaz Barak and David Steurer. Proofs, beliefs, and algorithms through the lens of sum-of-squares. *Course notes: http://www. sumofsquares.org/public/index.html*, 2016.

[BV02] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing (STOC)*, pages 109–115. ACM, 2002.

[CDG19] Yu Cheng, Ilias Diakonikolas, and Rong Ge. High-dimensional robust mean estimation in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2019.

[CDGW19] Yu Cheng, Ilias Diakonikolas, Rong Ge, and David Woodruff. Faster algorithms for high-dimensional robust covariance estimation. In *Conference on Learning Theory (COLT)*, 2019.

[CG18] Yu Cheng and Rong Ge. Non-convex matrix completion against a semi-random adversary. In *Conference On Learning Theory (COLT)*, 2018.

[CLS19] 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.

[CN21] Yeshwanth Cherapanamjeri and Jelani Nelson. Terminal embeddings in sublinear time. In *FOCS*, 2021.

[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.

[DHL19] Yihe Dong, Samuel Hopkins, and Jerry Li. Quantum entropy scoring for fast robust mean estimation and improved outlier detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6067–6077, 2019.

[DKK+16] I Diakonikolas, G Kamath, DM Kane, J Li, A Moitra, and A Stewart. Robust estimators in high dimensions without the computational intractability. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 655–664, 2016.

[FKP19] Noah Fleming, Pravesh Kothari, and Toniann Pitassi. *Semialgebraic Proofs and Efficient Algorithm Design.* Foundations and Trends in Theoretical Computer Science, 2019.

[GW94] Michel X Goemans and David P Williamson. .879-approximation algorithms for max cut and max 2sat. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 422–431, 1994.

[HJS+22] Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving sdp faster: A robust ipm framework and efficient implementation. In *2022 IEEE*

63rd Annual Symposium on Foundations of Computer Science (FOCS), pages 233–244. IEEE, 2022.

[HRS21] Samuel B Hopkins, Prasad Raghavendra, and Abhishek Shetty. Matrix discrepancy from quantum communication. *arXiv preprint arXiv:2110.10099*, 2021.

[JJUW11] Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP = PSPACE. *Journal of the ACM (JACM)*, 2011.

[JKL+20] Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *FOCS*, 2020.

[JLSW20] 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.

[JLT20] 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.

[JSWZ21] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. In *STOC*, 2021.

[Kha80] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[KM03] Kartik Krishnan and John E Mitchell. Properties of a cutting plane method for semidefinite programming. *submitted for publication*, 2003.

[KMS94] 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.

[KTE88] Leonid G Khachiyan, Sergei Pavlovich Tarasov, and I. I. Erlikh. The method of inscribed ellipsoids. *Soviet Math. Dokl*, 37(1):226–230, 1988.

[LG14] 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.

[LS17] Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 678–687, 2017.

[LS19] Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt (rank) linear system solves. *arXiv preprint arXiv:1910.08033*, 2019.

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

[LSZ19] 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.

[NN89] Yurii 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.

[NN92] Yurii Nesterov and Arkadi Nemirovski. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[NN94] Yurii 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.

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

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

[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.

[Wil12] 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.

[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.

A Missing Algorithms

In the Appendix, we put some main algorithms and lemmas in [HJS+22] to help the explanatory content in the main sections.

We first give the one-step error control conditions as the following:

Lemma A.1 (One step error control of the robust framework)**.** *Let the potential function of IPM defined by*

$$\Psi(z, y, \eta) := \|\mathsf{g}(y, \eta)\|_{(\nabla^2 \phi(z))^{-1}}.$$

Given any parameters $\alpha_S \in [1, 1 + 10^{-4}]$, $c_H \in [10^{-1}, 1]$, $\epsilon_g, \epsilon_\delta \in [0, 10^{-4}]$, *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* $\widetilde{H} \in \mathbb{S}^{n \times n}_{>0}$ *satisfies* $c_H \cdot \nabla^2 \phi(y) \preceq \widetilde{H} \preceq \nabla^2 \phi(y)$,
- **Condition 2.** *a vector* $\widetilde{g} \in \mathbb{R}^m$ *satisfies* $\|\widetilde{g} - \mathsf{g}(y, \eta^{\mathrm{new}})\|_{(\nabla^2 \phi(y))^{-1}} \leq \epsilon_g \cdot \|\mathsf{g}(y, \eta^{\mathrm{new}})\|_{(\nabla^2 \phi(y))^{-1}}$,
- **Condition 3.** *a vector* $\widetilde{\delta}_y \in \mathbb{R}^m$ *satisfies* $\|\widetilde{\delta}_y - \widetilde{H}^{-1} \widetilde{g}\|_{\nabla^2 \phi(y)} \leq \epsilon_\delta \cdot \|\widetilde{H}^{-1} \widetilde{g}\|_{\nabla^2 \phi(y)}$.

Then $\eta^{\mathrm{new}} = \eta(1 + \frac{\epsilon_N}{20\sqrt{\theta}})$ *and* $y^{\mathrm{new}} = y - \widetilde{\delta}_y$ *satisfy*

$$\Psi(y^{\mathrm{new}}, y^{\mathrm{new}}, \eta^{\mathrm{new}}) \leq \epsilon_N.$$

We state our main result of Algorithm 3 as follows:

Theorem A.2 (Main result for Algorithm 3)**.** *Given symmetric matrices* $C, A_1, \cdots, A_m \in \mathbb{R}^{n \times n}$, *and a vector* $b \in \mathbb{R}^m$. *Define matrix* $\mathsf{A} \in \mathbb{R}^{m \times n^2}$ *by stacking the* m *vectors* $\mathrm{vec}[A_1], \cdots, \mathrm{vec}[A_m] \in \mathbb{R}^{n^2}$ *as rows. Consider the following SDP instance:*

$$\begin{aligned} \max_{X \in \mathbb{R}^{n \times n}} \ & \langle C, X \rangle \\ \text{s.t.} \ & \langle A_i, X \rangle = b_i, \ \forall i \in [m], \\ & X \succeq 0, \end{aligned}$$

There is a SDP algorithm (Algorithm 3) that runs in time

$$O^*\Big(\big(\sqrt{n}(m^2 + n^4) + m^\omega + n^{2\omega}\big) \cdot \log(1/\epsilon)\Big).$$

and outputs a PSD matrix $X \in \mathbb{R}^{n \times n}$ *that satisfies*

$$\langle C, X \rangle \geq \langle C, X^* \rangle - \epsilon \cdot \|C\|_2 \cdot R \quad \textit{and} \quad \sum_{i=1}^m |\langle A_i, X \rangle - b_i| \leq 4n\epsilon \cdot \Big(R \sum_{i=1}^m \|A_i\|_1 + \|b\|_1\Big), \tag{6}$$

where X^* *is an optimal solution of the SDP instance, and* $\|A_i\|_1$ *is the Schatten* 1*-norm of matrix* A_i.

Lemma A.3. *For an SDP instance defined in Definition 1.1 (*m $n \times n$ *constraint matrices, let* X^* *be any optimal solution to SDP.), assume it has two properties :*

Algorithm 3 Our SDP solver with log barrier.

procedure SolveSDP($m, n, C, \{A_i\}_{i=1}^m$, $\mathsf{A} \in \mathbb{R}^{m \times n^2}$, $b \in \mathbb{R}^m$)

$\quad$ ▷ Initialization

$\quad$ Construct $\mathsf{A} \in \mathbb{R}^{m \times n^2}$ by stacking m vectors $\mathrm{vec}[A_1], \mathrm{vec}[A_2], \cdots, \mathrm{vec}[A_m] \in \mathbb{R}^{n^2}$

$\quad$ $\eta \leftarrow \frac{1}{n+2}, \quad T \leftarrow \frac{40}{\epsilon_N}\sqrt{n}\log(\frac{n}{\epsilon})$

$\quad$ Find initial feasible dual vector $y \in \mathbb{R}^m$ according to Lemma A.3

$\quad$ $S \leftarrow \sum_{i \in [m]} y_i \cdot A_i - C, \quad \widetilde{S} \leftarrow S$ ▷ $S, \widetilde{S} \in \mathbb{R}^{n \times n}$

$\quad$ $G \leftarrow (\mathsf{A} \cdot (\widetilde{S}^{-1} \otimes \widetilde{S}^{-1}) \cdot \mathsf{A}^\top)^{-1}$ ▷ $G \in \mathbb{R}^{m \times m}$

$\quad$ ▷ Maintain $G = \widetilde{H}^{-1}$ where $\widetilde{H} := \mathsf{A} \cdot (\widetilde{S}^{-1} \otimes \widetilde{S}^{-1}) \cdot \mathsf{A}^\top$

$\quad$ **for** $t = 1 \to T$ **do** ▷ Iterations of approximate barrier method

$\quad\quad$ $\eta^{\mathrm{new}} \leftarrow \eta \cdot (1 + \frac{\epsilon_N}{20\sqrt{n}})$

$\quad\quad$ **for** $j = 1, \cdots, m$ **do**

$\quad\quad\quad$ $g_{\eta^{\mathrm{new}}}(y)_j \leftarrow b_j \cdot \eta^{\mathrm{new}} - \mathrm{tr}[S^{-1} \cdot A_j]$ ▷ Gradient computation, $g_{\eta^{\mathrm{new}}}(y) \in \mathbb{R}^m$

$\quad\quad$ **end for**

$\quad\quad$ $\delta_y \leftarrow -G \cdot g_{\eta^{\mathrm{new}}}(y)$ ▷ Update on $y \in \mathbb{R}^m$

$\quad\quad$ $y^{\mathrm{new}} \leftarrow y + \delta_y$

$\quad\quad$ $S^{\mathrm{new}} \leftarrow \sum_{i \in [m]} (y^{\mathrm{new}})_i \cdot A_i - C$

$\quad\quad$ $V_1, V_2 \leftarrow$ LowRankSlackUpdate($S^{\mathrm{new}}, \widetilde{S}$) ▷ $V_1, V_2 \in \mathbb{R}^{n \times r_t}$. Algorithm 2.

$\quad\quad$ $\widetilde{S}^{\mathrm{new}} \leftarrow \widetilde{S} + V_1 V_2^\top$ ▷ Approximate slack computation

$\quad\quad$ $V_3 \leftarrow -\widetilde{S}^{-1} V_1 (I + V_2^\top \widetilde{S}^{-1} V_1)^{-1}$ ▷ $V_3 \in \mathbb{R}^{n \times r_t}$

$\quad\quad$ $V_4 \leftarrow \widetilde{S}^{-1} V_2$ ▷ $V_4 \in \mathbb{R}^{n \times r_t}$

$\quad\quad$ $Y_1 \leftarrow [(\widetilde{S}^{-1/2} \otimes V_3), (V_3 \otimes \widetilde{S}^{-1/2}), (V_3 \otimes V_3^\top)]$ ▷ $Y_1 \in \mathbb{R}^{n^2 \times (2nr_t + r_t^2)}$

$\quad\quad$ $Y_2 \leftarrow [(\widetilde{S}^{-1/2} \otimes V_4), (V_4 \otimes \widetilde{S}^{-1/2}), (V_4 \otimes V_4^\top)]$ ▷ $Y_2 \in \mathbb{R}^{n^2 \times (2nr_t + r_t^2)}$

$\quad\quad$ $G^{\mathrm{new}} \leftarrow G - G \cdot \mathsf{A} Y_1 \cdot (I + Y_2^\top \mathsf{A}^\top \mathsf{A} Y_1)^{-1} \cdot Y_2^\top \mathsf{A}^\top \cdot G$ ▷ $G^{\mathrm{new}} \in \mathbb{R}^{m \times m}$

$\quad\quad$ ▷ Hessian inverse computation using Woodbury identity

$\quad\quad$ $y \leftarrow y^{\mathrm{new}}$

$\quad\quad$ $S \leftarrow S^{\mathrm{new}}$

$\quad\quad$ $\widetilde{S} \leftarrow \widetilde{S}^{\mathrm{new}}$

$\quad\quad$ $G \leftarrow G^{\mathrm{new}}$ ▷ Update variables

$\quad$ **end for**

$\quad$ **return** an approximate solution to the original problem ▷ Lemma A.3

end procedure

1. *Bounded diameter: for any feasible solution $X \in \mathbb{R}^{n \times n}_{\succeq 0}$, it has $\|X\|_2 \le R$.*

2. *Lipschitz objective: the objective matrix $C \in \mathbb{R}^{n \times n}$ has bounded spectral norm, i.e., $\|C\|_2 \le L$.*

Given $\epsilon \in (0, 1/2]$, we can construct the following modified *SDP instance in dimension $n+2$ with $m+1$ constraints:*

$$\max_{\overline{X} \succeq 0} \ \langle \overline{C}, \overline{X} \rangle$$
$$\text{s.t. } \langle \overline{A}_i, \overline{X} \rangle = \overline{b}_i, \ \forall i \in [m+1],$$

where

$$\overline{A}_i = \begin{bmatrix} A_i & 0_n & 0_n \\ 0_n^\top & 0 & 0 \\ 0_n^\top & 0 & \frac{b_i}{R} - \mathrm{tr}[A_i] \end{bmatrix} \ \ \forall i \in [m], \ \ \textit{and} \ \ \overline{A}_{m+1} = \begin{bmatrix} I_n & 0_n & 0_n \\ 0_n^\top & 1 & 0 \\ 0_n^\top & 0 & 0 \end{bmatrix}.$$

Algorithm 4 The general robust barrier method framework for SDP.

procedure GeneralRobustSDP($\mathsf{A} \in \mathbb{R}^{m \times n^2}$, $b \in \mathbb{R}^m$, $C \in \mathbb{R}^{n \times n}$)
  Choose η and T
  Find initial feasible dual vector $y \in \mathbb{R}^m$ ▷ **Condition 0** in Lemma A.1
  for $t = 1 \to T$ **do do** ▷ Iterations of approximate barrier method
    $\eta^{\text{new}} \leftarrow \eta \cdot (1 + \frac{\epsilon_N}{20\sqrt{\theta}})$
    $\widetilde{S} \leftarrow$ ApproxSlack()
    $\widetilde{H} \leftarrow$ ApproxHessian() ▷ **Condition 1** in Lemma A.1
    $\widetilde{g} \leftarrow$ ApproxGradient() ▷ **Condition 2** in Lemma A.1
    $\widetilde{\delta}_y \leftarrow$ ApproxDelta() ▷ **Condition 3** in Lemma A.1
    $y^{\text{new}} \leftarrow y + \delta_y$
    $y \leftarrow y^{\text{new}}$ ▷ Update variables
  end for
end procedure

$$\overline{b} = \begin{bmatrix} \frac{1}{R} b \\ n+1 \end{bmatrix} , \ \overline{C} = \begin{bmatrix} \frac{\epsilon}{L} \cdot C & 0_n & 0_n \\ 0_n^\top & 0 & 0 \\ 0_n^\top & 0 & -1 \end{bmatrix}.$$

Moreover, it has three properties:

1. $(\overline{X}_0, \overline{y}_0, \overline{S}_0)$ *are feasible primal and dual solutions of the modified instance, where*

$$\overline{X}_0 = I_{n+2} \ , \ \overline{y}_0 = \begin{bmatrix} 0_m \\ 1 \end{bmatrix} \ , \ \overline{S}_0 = \begin{bmatrix} I_n - C \cdot \frac{\epsilon}{L} & 0_n & 0 \\ 0_n^\top & 1 & 0 \\ 0_n^\top & 0 & 1 \end{bmatrix}. \tag{7}$$

2. *For any feasible primal and dual solutions* $(\overline{X}, \overline{y}, \overline{S})$ *with duality gap at most* ϵ^2*, the matrix* $\widehat{X} = R \cdot \overline{X}_{[n]\times[n]}$*, where* $\overline{X}_{[n]\times[n]}$ *is the top-left* n*-by-*n *block submatrix of* $\overline{X}$*. The matrix* $\widehat{X}$ *has three properties*

$$\begin{aligned} \langle C, \widehat{X} \rangle &\geq \langle C, X^* \rangle - LR \cdot \epsilon, \\ \widehat{X} &\succeq 0, \\ \sum_{i \in [m]} |\langle A_i, \widehat{X} \rangle - b_i| &\leq 4n\epsilon \cdot \Big(R \sum_{i \in [m]} \|A_i\|_1 + \|b\|_1 \Big), \end{aligned}$$

3. *If we take* $\epsilon \leq \epsilon_N^2$ *in Eq. (7), then the initial dual solution satisfies the induction invariant:*

$$g(\overline{y}_0, \eta) H(\overline{y}_0)^{-1} g(\overline{y}_0, \eta) \leq \epsilon_N^2.$$

B 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 $\widetilde{S}$ and $\widetilde{H}$ to $\widetilde{S}_{\text{new}}$ and $\widetilde{H}_{\text{new}}$ is proportional to the magnitude of low-rank change in $\widetilde{S}$, namely $r_t = \text{rank}(\widetilde{S}_{\text{new}} - \widetilde{S})$. To deal with r_t, we propose a general amortization method which extends the analysis of several previous work [CLS19, LSZ19, JKL+20]. We first prove a tool to characterize intrinsic properties of the low-rank updates, which may be of independent interest.

Theorem B.1 (General amortized guarantee)**.** *Given a sequence of approximate slack matrices* $\widetilde{S}^{(1)}, \widetilde{S}^{(2)}, \ldots, \widetilde{S}^{(T)} \in \mathbb{R}^{n \times n}$ *generated by Algorithm 3, let* $r_t = \operatorname{rank}(\widetilde{S}^{(t+1)} - \widetilde{S}^{(t)})$ *denotes the rank of update on* $\widetilde{S}^{(t)}$*. Then for any non-increasing vector* $g \in \mathbb{R}^n_+$*, we have*

$$\sum_{t=1}^{T} r_t \cdot g_{r_t} \leq \widetilde{O}(T \cdot \|g\|_2).$$

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

Proof. For any matrix Z, let $|\lambda(Z)|_{[i]}$ denotes its i-th largest absolute eigenvalue. We use the following potential function $\Phi_g(Z) := \sum_{i=1}^{n} g_i \cdot |\lambda(Z)|_{[i]}$. Further, for convenient, we define $\Phi_g(S_1, S_2) := \Phi_g(S_1^{-1/2} S_2 S_1^{-1/2} - I)$. Our proof consists of the following two parts:

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

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

$$\Phi_g(S^{\text{new}}, \widetilde{S}^{\text{new}}) - \Phi_g(S, \widetilde{S}) = \Phi_g(S^{\text{new}}, \widetilde{S}) - \Phi_g(S, \widetilde{S}) + \Phi_g(S^{\text{new}}, \widetilde{S}^{\text{new}}) - \Phi_g(S^{\text{new}}, \widetilde{S}).$$

Note that $\Phi_g(S, \widetilde{S}) = 0$ holds in the beginning of our algorithm and $\Phi_g(S, \widetilde{S}) \geq 0$ holds throughout the algorithm, combining the observations above we have $T \cdot \|g\|_2 - \sum_{t=1}^{T} r_t \cdot g_{r_t} \geq 0$ as desired. □

Amortized analysis. Next we show how to use Theorem B.1 to prove that our algorithm has an amortized cost of $m^{\omega - 1/4} + m^2$ 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. (5)) is $\mathcal{T}_{\text{mat}}(m, m, nr_t)$. We use fast rectangular matrix multiplication to upper bound this term by

$$\mathcal{T}_{\text{mat}}(m, m, nr_t) \leq m^2 + m^{2 - \frac{\alpha(\omega-2)}{1-\alpha}} \cdot n^{\frac{\omega-2}{1-\alpha}} \cdot r_t^{\frac{\omega-2}{1-\alpha}}.$$

We define a non-increasing sequence $g \in \mathbb{R}^n$ as $g_i = i^{\frac{\omega-2}{1-\alpha} - 1}$. This g is tailored for the above equation, and its ℓ_2 norm is bounded by $\|g\|_2 \leq n^{\frac{(\omega-2)}{1-\alpha} - 1/2}$. Then using Theorem B.1 we have

$$\sum_{t=1}^{T} r_t^{\frac{\omega-2}{1-\alpha}} = \sum_{t=1}^{T} r_t \cdot r_t^{\frac{\omega-2}{1-\alpha} - 1} = \sum_{t=1}^{T} r_t \cdot g_{r_t} \leq T \cdot n^{\frac{(\omega-2)}{1-\alpha} - 1/2}.$$

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

$$\sum_{t=1}^{T} \mathcal{T}_{\text{mat}}(m, m, nr_t) \leq T \cdot (m^2 + m^{2 - \frac{\alpha(\omega-2)}{1-\alpha}} \cdot n^{\frac{2(\omega-2)}{1-\alpha} - 1/2}) = T \cdot (m^2 + m^{\omega - 1/4}).$$

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

Volumetric barrier was first proposed by Vaidya [Vai89a] for the polyhedral, and was generalized to the spectrahedra $\{y \in \mathbb{R}^m : y_1 A_1 + \cdots + y_m A_m \succeq 0\}$ by Nesterov and Nemirovski [NN94]. They showed that the volumetric barrier ϕ_{vol} can make the interior point method converge in $\sqrt{m} n^{1/4}$ 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 B.2. *There is an SDP algorithm based on hybrid barrier which takes $(mn)^{1/4}\log(1/\epsilon)$ iterations with cost-per-iteration $O^*\left(m^2 n^\omega + m^4\right)$.*

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 $n^{\omega+2.5}$ time, while [Ans00] takes $n^{6.5}$ time.

The hybrid barrier function is as follows:

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

where $\phi_{\mathrm{vol}}(y) = \frac{1}{2}\log\det(\nabla^2 \phi_{\log}(y))$. According to our general IPM framework (Algorithm 4), we need to efficiently compute the gradient and Hessian of $\phi(y)$. Recall from [Ans00] that the gradient of the volumetric barrier is:

$$(\nabla \phi_{\mathrm{vol}}(y))_i = -\mathrm{tr}[H(S)^{-1} \cdot \mathsf{A}(S^{-1} A_i S^{-1} \otimes S^{-1})\mathsf{A}^\top] \quad \forall i \in [m].$$

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

$$\begin{aligned} Q(S)_{i,j} &= \mathrm{tr}[H(S)^{-1}\mathsf{A}(S^{-1}A_i S^{-1} A_j S^{-1} \otimes_S S^{-1})\mathsf{A}^\top], \\ R(S)_{i,j} &= \mathrm{tr}[H(S)^{-1}\mathsf{A}(S^{-1}A_i S^{-1} \otimes_S S^{-1} A_j S^{-1})\mathsf{A}^\top], \\ T(S)_{i,j} &= \mathrm{tr}[H(S)^{-1}\mathsf{A}(S^{-1}A_i S^{-1} \otimes_S S^{-1})\mathsf{A}^\top H(S)^{-1}\mathsf{A}(S^{-1}A_j S^{-1} \otimes_S S^{-1})\mathsf{A}^\top]. \end{aligned} \tag{8}$$

Here, $\otimes_S$ is the symmetric Kronecker product[4].

A straight-forward implementation of the hybrid barrier-based SDP algorithm can first compute the matrices $S^{-1}A_i$ and $S^{-1}A_i S^{-1} A_j$ for all $i \in \{1, 2, \cdots, m\}$ for all $j \in \{1, 2, \cdots, m\}$ in time $O(m^2 n^\omega)$. The gradient $\nabla\phi(y)$ and the Hessian of $\phi_{\log}(y)$ can be computed by taking traces of these matrices. To compute $\nabla\phi_{\mathrm{vol}}(y), Q(S), R(S), T(S)$, 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 $\mathrm{tr}[S^{-1}A_i S^{-1} A_j S^{-1} A_k]$ and $\mathrm{tr}[S^{-1}A_i S^{-1} A_j S^{-1} A_k S^{-1} A_l]$ for $i, j, k, l \in [m]$. Hence, we can spend $O(m^4 n^2)$-time computing these traces and then get $\nabla\phi_{\mathrm{vol}}(y), Q(S), R(S), T(S)$ in $O(m^{\omega+2})$-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 $\delta_y = -(\nabla^2\phi(y))^{-1}(\eta b - \nabla\phi(y))$.

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

1. Computing the traces $\mathrm{tr}[S^{-1}A_i S^{-1} A_j S^{-1} A_k S^{-1} A_l]$ for $i, j, k, l \in [m]$.

[4] $X \otimes_S Y := \frac{1}{2}(X \otimes Y + Y \otimes X)$.

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. (8)) is much more complicated than the log barrier's Hessian $H(S)$. For $H(S)$, if we replace S with its approximation $\widetilde{S}$, then $H(\widetilde{S})$ 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 $\widetilde{S}$. We can resolve this challenge by carefully choosing the approximation place: if we approximate the second S in the trace, i.e., $\mathrm{tr}[S^{-1}A_i\widetilde{S}^{-1}A_jS^{-1}A_kS^{-1}A_l]$, 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 A.1) is satisfied. Notice that in each iteration, we only need to maintain the change of $\mathrm{tr}[S^{-1}A_i\widetilde{S}^{-1}A_jS^{-1}A_kS^{-1}A_l]$, which by the low-rank guarantee, can be written as

$$\mathrm{tr}[A_lS^{-1}A_i \cdot V_3V_4^\top \cdot A_jS^{-1}A_kS^{-1}],$$

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

$$\left\{A_lS^{-1}A_iV_3 \in \mathbb{R}^{n\times r_t}\right\}_{i,l\in[m]} \text{ and } \left\{V_4^\top A_jS^{-1}A_kS^{-1} \in \mathbb{R}^{r_t\times n}\right\}_{j,k\in[m]}.$$

It takes $m^2 \cdot \mathcal{T}_{\mathrm{mat}}(n, n, r_t)$-time. And we can compute all the traces $\mathrm{tr}[S^{-1}A_i\widetilde{S}^{-1}A_jS^{-1}A_kS^{-1}A_l]$ simultaneously in $\mathcal{T}_{\mathrm{mat}}(m^2, nr_t, m^2)$ by batching them together and using fast matrix multiplication on a m^2-by-nr_t matrix and a nr_t-by-m^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 B) changes more drastically in the hybrid barrier case. And we can only get $\sum_{t=1}^{T} r_t \cdot g_{r_t} \le O(T \cdot (n/m)^{1/4} \cdot \|g\|_2 \cdot \log n)$.

For the second issue, we note that computing the $T(S)$ matrix is the most time-consuming step, which need $m^{\omega+2}$-time. In [Ans00], it is proved that $\frac{1}{3}Q(S) \preceq \nabla^2\phi_{\mathrm{vol}}(y) \preceq Q(S)$. 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 $O(m^4)$-time, which improves the $m^{\omega+2}$ term in the straight forward implementation.

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 reweighing the constraints in the log barrier based on the leverage score, and the Lee-Sidford barrier uses Lewis weights for reweighing 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.