A Faster Interior Point Method for Semidefinite Programming

Haotian Jiang
jhtdavid@uw.edu
University of Washington

Tarun Kathuria
tarunkathuria@berkeley.edu
University of California, Berkeley

Yin Tat Lee
yintat@uw.edu
University of Washington, Microsoft Research Redmond

Swati Padmanabhan
pswati@uw.edu
University of Washington

Zhao Song
magic.linuxkde@gmail.com
Columbia University, Princeton University, and Institute for Advanced Study

***Abstract*—**

Semidefinite programs (SDPs) are a fundamental class of optimization problems with important recent applications in approximation algorithms, quantum complexity, robust learning, algorithmic rounding, and adversarial deep learning. This paper presents a faster interior point method to solve generic SDPs with variable size $n \times n$ and m constraints in time

$$\widetilde{O}(\sqrt{n}(mn^2 + m^\omega + n^\omega)\log(1/\epsilon)),$$

where ω is the exponent of matrix multiplication and ϵ is the relative accuracy. In the predominant case of $m \geq n$, our runtime outperforms that of the previous fastest SDP solver, which is based on the cutting plane method [JLSW20].

Our algorithm's runtime can be naturally interpreted as follows: $O(\sqrt{n}\log(1/\epsilon))$ is the number of iterations needed for our interior point method, mn^2 is the input size, and $m^\omega + n^\omega$ is the time to invert the Hessian and slack matrix in each iteration. These constitute natural barriers to further improving the runtime of interior point methods for solving generic SDPs.

I. Introduction

Semidefinite programs (SDPs) constitute a class of convex optimization problems that optimize a linear objective over the intersection of the cone of positive semidefinite matrices with an affine space. SDPs generalize linear programs and have a plethora of applications in operations research, control theory, and theoretical computer science [VB96]. Applications in theoretical computer science include improved approximation algorithms for fundamental problems (e.g., Max-Cut [GW95], coloring 3-colorable graphs [KMS94], and sparsest cut [ARV09]), quantum complexity theory [JJUW11], robust learning and estimation [CG18], [CDG19], [CDGW19], algorithmic discrepancy and rounding [BDG16], [BG17], [Ban19], and adversarial deep learning [RSL18]. We formally define SDPs with variable size $n \times n$ and m constraints as follows.

Definition I.1 (Semidefinite programming)**.** *Given symmetric*[1] *matrices $C, A_1, \cdots, A_m \in \mathbb{R}^{n\times n}$ and $b_i \in \mathbb{R}$ for all $i \in [m]$, the goal is to solve the convex optimization problem*

$$\max \langle C, X\rangle \text{ subject to } X \succeq 0, \langle A_i, X\rangle = b_i \ \forall i \in [m] \quad (1)$$

where $\langle A, B\rangle := \sum_{i,j} A_{i,j}B_{i,j}$ is the trace product.

Cutting plane and interior point methods: Two prominent methods for solving SDPs, with runtimes depending *logarithmically* on the accuracy parameter ϵ, are the *cutting plane method* and the *interior point method*.

The cutting plane method maintains a convex set containing the optimal solution. In each iteration, the algorithm queries a separation oracle, which returns a hyperplane that divides the convex set into two subsets. The convex set is then updated to contain the subset with the optimal solution. This process is repeated until the volume of the maintained set becomes small enough and a near-optimal solution can be found. Since Khachiyan proved [Kha80] that the ellipsoid method solves linear programs in polynomial time, cutting plane methods have played a crucial role in both discrete and continuous optimization [GLS81], [GV02].

In contrast, interior point methods add a barrier function to the objective and, by adjusting the weight of this barrier function, solve a different optimization problem in each iteration. The solutions to these successive problems form a well-defined *central path*. Since Karmarkar proved [Kar84] that interior point methods can solve linear programs in polynomial time, these methods have become an active research area. Their advantage is that the number of iterations is usually the square root of the number of dimensions, as opposed to the linear dependence on dimensions in cutting plane methods.

Since cutting plane methods use less structural information than interior point methods, they are slower at solving almost all problems where interior point methods are known to apply. However, SDPs remain one of the most

[1]We can, without loss of generality, assume that $C, A_1, \cdots, A_m$ are symmetric, since given any $M \in \{C, A_1, \cdots, A_m\}$, we have $\sum_{i,j} M_{ij}X_{ij} = \sum_{i,j} M_{ij}X_{ji} = \sum_{i,j}(M^\top)_{ij}X_{ij}$, and therefore we can replace M with $(M + M^\top)/2$.

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00089

fundamental optimization problems where the state of the art is, in fact, the opposite: the current fastest cutting plane methods[2] of [LSW15], [JLSW20] solve a general SDP in time $m(mn^2 + m^2 + n^\omega)$, while the fastest SDP solvers based on interior point methods in the work of [NN92] and [Ans00] achieve runtimes of $\sqrt{n}(m^2n^2 + mn^\omega + m^\omega)$ and $(mn)^{1/4}(m^4n^2+m^3n^\omega)$, respectively, which are slower in the most common regime of $m \in [n, n^2]$ (see Table II). This apparent paradox raises the following natural question:

How fast can SDPs be solved using interior point methods?

A. Our results

We present a faster interior point method for solving SDPs. Our main result is the following theorem (a formal statement of which is given in the full version).

Theorem I.2 (Main result, informal)**.** *There is an interior point method that solves a general SDP with variable size $n \times n$ and m constraints in time[3] $O^*(\sqrt{n}(mn^2+m^\omega+n^\omega))$.*

Our runtime can be roughly interpreted as follows:

- $\sqrt{n}$ is the iteration complexity of the interior point method with the log barrier function.
- mn^2 is the input size.
- m^ω is the cost of inverting the Hessian of the log barrier.
- n^ω is the cost of inverting the slack matrix.

Thus, the terms in the runtime of our algorithm arise as a natural barrier to further speeding up SDP solvers. See Section I-B2, I-B3 and I-B4 for more detail.

Table I compares our result with previous SDP solvers. The first takeaway of this table and Theorem I.2 is that our interior point method always runs faster than that in [NN92] and is faster than that in [NN94] and [Ans00] when $m \geq n^{1/13}$. A second consequence is that whenever $m \geq \sqrt{n}$, our interior point method is faster than the current fastest cutting plane method [LSW15], [JLSW20]. We note that $n \leq m \leq n^2$ is satisfied in most SDP applications known to us, such as classical combinatorial optimization problems over graphs (in which m is usually the number of edges, and $\Omega(n) \leq m \leq O(n^2)$ as long as the graph is connected), experiment design problems in statistics and machine learning, and sum-of-squares problems. An explicit comparison to previous algorithms in the cases of $m = n$ and $m = n^2$ is shown in Table II.

Even in the more general case where the SDP might not be dense, where $\mathrm{nnz}(A)$ is the input size (i.e., the total number of non-zeroes in all matrices A_i for $i \in [m]$ and C), our interior point method runs faster than the current fastest cutting plane methods[LSW15], [JLSW20], which run in time $O^*(m(\mathrm{nnz}(A) + m^2 + n^\omega))$.

Theorem I.3 (Comparison with Cutting Plane Method)**.** *When $m \geq n$, there is an interior point method that solves an SDP with $n \times n$ matrices, m constraints, and $\mathrm{nnz}(A)$ input size, faster than the current best cutting plane method [LSW15], [JLSW20], over all regimes of $\mathrm{nnz}(A)$.*

B. Technique overview

1) Interior point method for solving SDPs: By removing redundant constraints, we can, without loss of generality, assume $m \leq n^2$ in the primal formulation of the SDP (1). Thereafter, instead of solving the primal SDP, which has variable size $n \times n$, we solve its dual formulation, which has dimension $m \leq n^2$:

$$\min b^\top y \text{ subject to } S = \sum_{i=1}^m y_i A_i - C, \text{ and } S \succeq 0. \quad (2)$$

Interior point methods solve (2) by minimizing the penalized objective function:

$$\min_{y \in \mathbb{R}^m} f_\eta(y), \text{ where } f_\eta(y) := \eta \cdot b^\top y + \phi(y), \quad (3)$$

where $\eta > 0$ is a parameter and $\phi : \mathbb{R}^m \to \mathbb{R}$ is a *barrier* function that approaches infinity as y approaches the boundary of the feasible set $\{y \in \mathbb{R}^m : \sum_{i=1}^m y_i A_i \succeq C\}$. These methods first obtain an approximate minimizer of f_η for some small $\eta > 0$, which they then use as an initial point to minimize $f_{(1+c)\eta}$, for some constant $c > 0$, via the Newton method. This process repeats until the parameter η in (3) becomes sufficiently large, at which point the minimizer of f_η is provably close to the optimal solution of (2). The iterates y generated by this method follow a *central path*. Different choices of the barrier function ϕ lead to different run times in solving (3), as we next describe.

The log barrier: Nesterov and Nemirovski [NN92] use the *log barrier* function,

$$\phi(y) = g(y) := -\log\det\left(\sum_{i=1}^m y_i A_i - C\right), \quad (4)$$

in (3) and, in $O(\sqrt{n}\log(n/\epsilon))$ iterations, obtain a feasible dual solution y that satisfies $b^\top y \leq b^\top y^* + \epsilon$, where $y^* \in \mathbb{R}^m$ is the optimal solution for (2). Within each iteration, the costliest step is to compute the inverse of the Hessian of the log barrier function for the Newton step. For each $(j,k) \in [m] \times [m]$, the (j,k)-th entry of H is given by

$$H_{j,k} = \mathrm{tr}[S^{-1}A_jS^{-1}A_k]. \quad (5)$$

The analysis of [NN92] first computes $S^{-1/2}A_jS^{-1/2}$ for all $j \in [m]$, which takes time $O^*(mn^\omega)$, and then calculates the m^2 trace products $\mathrm{tr}[S^{-1}A_jS^{-1}A_k]$ for all $(j,k) \in [m] \times [m]$, each of which takes $O(n^2)$ time. Inverting the Hessian costs $O^*(m^\omega)$, which results in a total runtime of $O^*(\sqrt{n}(m^2n^2 + mn^\omega + m^\omega))$.

[2] [JLSW20] improves upon the runtime of [LSW15] in terms of the dependence on $\log(n/\epsilon)$, while the polynomial factors are the same in both runtimes.

[3] We use O^* to hide $n^{o(1)}$ and $\log^{O(1)}(n/\epsilon)$ factors and $\widetilde{O}$ to hide $\log^{O(1)}(n/\epsilon)$ factors, where ϵ is the accuracy parameter.

Table I

SUMMARY OF KEY SDP ALGORITHMS. CPM STANDS FOR CUTTING PLANE METHOD, AND IPM STANDS FOR INTERIOR POINT METHOD. IN THE RUNTIMES, n DENOTES THE SIZE OF THE VARIABLE MATRIX, AND $m \leq n^2$ DENOTES THE NUMBER OF CONSTRAINTS. RUNTIMES HIDE $n^{o(1)}$, $m^{o(1)}$ AND $\mathrm{poly}\log(1/\epsilon)$ FACTORS, WHERE ϵ IS THE ACCURACY PARAMETER. [ANS00] SIGNIFICANTLY SIMPLIFIES THE PROOFS IN [NN94, SECTION 5.5], AND BOTH USE A GENERALIZED VERSION OF THE VOLUMETRIC BARRIER FUNCTION INTRODUCED IN [VAI89A]. NEITHER [ANS00] NOR [NN94] EXPLICITLY ANALYZED THE RUNTIMES OF THEIR IPM, AND THEIR RUNTIMES SHOWN HERE ARE OUR BEST ESTIMATES. WE REMARK THAT [JLSW20] IMPROVES UPON THE RUNTIME OF [LSW15] IN TERMS OF THE DEPENDENCE ON $\log(n/\epsilon)$, WHICH IS NOT SHOWN IN THE TABLE.

Year	References	Method	#Iters	Cost per iter
1979	[Sho77], [YN76], [Kha80]	CPM	m^2	$mn^2 + m^2 + n^\omega$
1988	[KTE88], [NN89]	CPM	m	$mn^2 + m^{3.5} + n^\omega$
1989	[Vai89a]	CPM	m	$mn^2 + m^\omega + n^\omega$
1992	[NN92]	IPM	$\sqrt{n}$	$m^2n^2 + mn^\omega + m^\omega$
1994	[NN94]	IPM	$(mn)^{1/4}$	$m^4n^2 + m^3n^\omega$
2000	[Ans00]	IPM	$(mn)^{1/4}$	$m^4n^2 + m^3n^\omega$
2003	[KM03]	CPM	m	$mn^2 + m^\omega + n^\omega$
2015	[LSW15]	CPM	m	$mn^2 + m^2 + n^\omega$
2020	[JLSW20]	CPM	m	$mn^2 + m^2 + n^\omega$
2020	Our result	IPM	$\sqrt{n}$	$mn^2 + m^\omega + n^\omega$

Table II

TOTAL RUNTIMES FOR THE ALGORITHMS IN TABLE I FOR SDPS WHEN $m = n$ AND $m = n^2$, WHERE n DENOTES THE SIZE OF MATRICES AND m DENOTES THE NUMBER OF CONSTRAINTS. THE RUNTIMES SHOWN IN THE TABLE HIDE $n^{o(1)}$, $m^{o(1)}$ AND $\mathrm{poly}\log(1/\epsilon)$ FACTORS, WHERE ϵ IS THE ACCURACY PARAMETER AND ASSUME ω TO EQUAL ITS CURRENTLY BEST KNOWN UPPER BOUND OF 2.373.

Year	References	Method	Runtime	
			$m = n$	$m = n^2$
1979	[Sho77], [YN76], [Kha80]	CPM	n^5	n^8
1988	[KTE88], [NN89]	CPM	$n^{4.5}$	n^9
1989	[Vai89a]	CPM	n^4	$n^{6.746}$
1992	[NN92]	IPM	$n^{4.5}$	$n^{6.5}$
1994	[NN94]	IPM	$n^{6.5}$	$n^{10.75}$
2000	[Ans00]	IPM	$n^{6.5}$	$n^{10.75}$
2003	[KM03]	CPM	n^4	$n^{6.746}$
2015	[LSW15]	CPM	n^4	n^6
2020	[JLSW20]	CPM	n^4	n^6
2020	Our result	IPM	$n^{3.5}$	$n^{5.246}$

The volumetric barrier: Vaidya [Vai89a] introduced the *volumetric barrier* for a polyhedral set $\{x \in \mathbb{R}^n : Ax \geq c\}$, where $A \in \mathbb{R}^{m\times n}$ and $c \in \mathbb{R}^m$. Nesterov and Nemirovski [NN94] studied the following extension of the volumetric barrier to the convex subset $\{y \in \mathbb{R}^m : \sum_{i=1}^m y_i A_i \succeq C\}$ of the polyhedral cone:

$$V(y) = \frac{1}{2}\log\det(\nabla^2 g(y)),$$

where $g(y)$ is the log barrier function defined in (4). They proved that choosing $\phi(y) = \sqrt{n}V(y)$ in (3) makes the interior point method converge in $\widetilde{O}(\sqrt{m}n^{1/4})$ iterations, which is smaller than the $\widetilde{O}(\sqrt{n})$ iteration complexity of [NN92] when $m \leq \sqrt{n}$. They also studied the *combined volumetric-logarithmic barrier*

$$V_\rho(y) = V(y) + \rho \cdot g(y)$$

and showed that taking $\phi(y) = \sqrt{n/m}\cdot V_\rho(y)$ for $\rho = (m-1)/(n-1)$ yields an iteration complexity of $\widetilde{O}((mn)^{1/4})$. when $m \leq n$, this iteration complexity is lower than $\widetilde{O}(\sqrt{n})$ of [NN92]. We refer readers to the much simpler proofs in [Ans00] for these results.

However, the volumetric barrier (and thus the combined volumetric-logarithmic barrier) leads to complicated expressions for the gradient and Hessian that make each iteration costly. For instance, the Hessian of the volumetric barrier is

$$\nabla^2 V(y) = 2Q(y) + R(y) - 2T(y),$$

where $Q(y)$, $R(y)$, and $T(y)$ are $m \times m$ matrices such that for each $(j,k) \in [m] \times [m]$,

$$\begin{aligned} Q(y)_{j,k} &= \mathrm{tr}\left[\mathcal{A}H^{-1}\mathcal{A}^\top\left((B_j S B_k)\,\widehat{\otimes} S^{-1}\right)\right], \\ R(y)_{j,k} &= \mathrm{tr}\left[\mathcal{A}H^{-1}\mathcal{A}^\top\left(B_j \widehat{\otimes} B_k\right)\right], \\ T(y)_{j,k} &= \mathrm{tr}\left[\mathcal{A}H^{-1}\mathcal{A}^\top(B_j \widehat{\otimes} S^{-1})\mathcal{A}H^{-1}\mathcal{A}^\top(B_k \widehat{\otimes} S^{-1})\right]. \end{aligned} \tag{6}$$

where $B_j := S^{-1}A_j S^{-1}$.

Here, $\mathcal{A} \in \mathbb{R}^{n^2\times m}$ is the $n^2 \times m$ matrix whose ith column is obtained by flattening A_i into a vector of length n^2, and $\widehat{\otimes}$ is the symmetric Kronecker product

$$A \widehat{\otimes} B := \frac{1}{2}(A \otimes B + B \otimes A),$$

where $\otimes$ is the Kronecker product (see full version for formal definition). Due to the complicated formulas in (6), efficient computation of Newton step in each iteration of the

interior point method is difficult; in fact, each iteration runs slower than the Nesterov-Nemirovski interior point method by a factor of m^2. Since most applications of SDPs known to us have the number of constraints m be at least linear in n, the total runtime of interior point methods based on the volumetric barrier and the combined volumetric-logarithmic barrier is inevitably slow.

2) Our techniques: Given the inefficiency of implementing the volumetric and volumetric-logarithmic barriers discussed above, this paper uses the log barrier in (4). We now describe some of our key techniques that improve the runtime of the Nesterov-Nemirovski interior point method [NN92].

Hessian computation using fast rectangular matrix multiplication: As noted in Section I-B1, the runtime bottleneck in [NN92] is computing the inverse of the Hessian of the log barrier function, where the Hessian is described in (5). In [NN92], each of these m^2 entries is computed separately, resulting in a runtime of $O(m^2n^2)$ per iteration.

Instead contrast, we show below how to group these computations using rectangular matrix multiplication. The expression from (5) can be re-written as

$$H_{j,k} = \mathrm{tr}[S^{-1/2}A_jS^{-1/2} \cdot S^{-1/2}A_kS^{-1/2}]. \tag{7}$$

We first compute the key quantity $S^{-1/2}A_jS^{-1/2} \in \mathbb{R}^{n\times n}$ for all $j \in [m]$ by stacking all matrices $A_j \in \mathbb{R}^{n\times n}$ into a tall matrix of size $mn \times n$, and then compute the product of $S^{-1/2} \in \mathbb{R}^{n\times n}$ with this tall matrix. This matrix product can be computed in time $\mathcal{T}_{\mathrm{mat}}(n, mn, n)$[4] using fast rectangular matrix multiplication. We then flatten each $S^{-1/2}A_jS^{-1/2}$ into a row vector of length n^2 and stack all m vectors to form a matrix $\mathcal{B}$ of size $m \times n^2$, i.e., the j-th row of $\mathcal{B}$ is $\mathcal{B}_j = \mathsf{vec}(S^{-1/2}A_jS^{-1/2})$. It follows that the Hessian can be computed as

$$H = \mathcal{B}\mathcal{B}^\top, \tag{8}$$

which takes time $\mathcal{T}_{\mathrm{mat}}(m, n^2, m)$ by applying fast rectangular matrix multiplication. By leveraging recent developments in this area [GU18], this approach already improves upon the runtime in [NN92].

Thus far, we have reduced the per iteration cost of $O^*(m^2n^2 + mn^\omega)$ for Hessian computation down to

$$\mathcal{T}_{\mathrm{mat}}(n, mn, n) + \mathcal{T}_{\mathrm{mat}}(m, n^2, m).$$

Low rank update on the slack matrix: The fast rectangular matrix multiplication approach noted above, however, is still not very efficient, because the Hessian must be computed from scratch in each iteration of the interior point method. If there are T iterations in total, it then takes time

$$T \cdot (\mathcal{T}_{\mathrm{mat}}(n, mn, n) + \mathcal{T}_{\mathrm{mat}}(m, n^2, m)).$$

[4]We define $\mathcal{T}_{\mathrm{mat}}(n, r, m)$ to be the number of operations needed to compute the product of matrices of dimensions $n \times r$ and $r \times m$.

To further improve the runtime, we need to efficiently update the Hessian for the current iteration from the Hessian computed in the previous one. Generally, this is not possible, as the slack matrix $S \in \mathbb{R}^{n\times n}$ in (7) might change arbitrarily in the Nesterov-Nemirovski interior point method.

To overcome this problem, we propose a new interior point method that maintains an approximate slack matrix $\widetilde{S} \in \mathbb{R}^{n\times n}$, which is a spectral approximation of the true slack matrix $S \in \mathbb{R}^{n\times n}$ such that $\widetilde{S}$ admits a *low-rank update* in each iteration. Where needed, we will now use the subscript t to denote a matrix in the t-th iteration. Our algorithm updates only the directions in which $\widetilde{S}_t$ deviates too much from S_{t+1}; the changes to S_t for the remaining directions are not propagated in $\widetilde{S}_t$. This process of selective update ensures a low-rank change in $\widetilde{S}_t$ even when S_t suffers from a full-rank update; it also guarantees the proximity of the algorithm's iterates to the central path. Specifically, for each iteration $t \in [T]$, we define the *difference matrix*

$$Z_t = S_t^{-1/2}\widetilde{S}_tS_t^{-1/2} - I \quad \in \mathbb{R}^{n\times n},$$

which intuitively captures how far the approximate slack matrix $\widetilde{S}_t$ is from the true slack matrix S_t. We maintain the invariant $\|Z_t\|_{\mathrm{op}} \le c$ for some sufficiently small constant $c > 0$. In the $(t+1)$-th iteration when S_t gets updated to S_{t+1}, our construction of $\widetilde{S}_{t+1}$ involves a novel approach of zeroing out some of the largest eigenvalues of $|Z_t|$ to bound the rank of the update on the approximate slack matrix.

We prove that with this approach, the updates on $\widetilde{S} \in \mathbb{R}^{n\times n}$ over all $T = \widetilde{O}(\sqrt{n})$ iterations satisfy the following *rank inequality* (see full version for the formal statement).

Theorem I.4 (Rank inequality, informal version)**.** *Let $\widetilde{S}_1, \widetilde{S}_2, \cdots, \widetilde{S}_T \in \mathbb{R}^{n\times n}$ denote the sequence of approximate slack matrices generated in our interior point method. For each $t \in [T-1]$, denote by $r_t = \mathrm{rank}(\widetilde{S}_{t+1} - \widetilde{S}_t)$ the rank of the update on $\widetilde{S}_t$. Then, the sequence $r_1, r_2, \cdots, r_T$ satisfies*

$$\sum_{t=1}^{T} \sqrt{r_t} = \widetilde{O}(T).$$

The key component to proving Theorem I.4 is the potential function $\Phi : \mathbb{R}^{n\times n} \to \mathbb{R}_{\ge 0}$

$$\Phi(Z) := \sum_{\ell=1}^{n} \frac{|\lambda(Z)|_{[\ell]}}{\sqrt{\ell}},$$

where $|\lambda(Z)|_{[\ell]}$ is the ℓ-th in the list of eigenvalues of $Z \in \mathbb{R}^{n\times n}$ sorted in decreasing order of their absolute values. We show an upper bound on the increase in this potential when S is updated, a lower bound on its decrease when $\widetilde{S}$ is updated, and combine the two with non-negativity of the potential to obtain Theorem I.4.

Specifically, first we prove that whenever S is updated in an iteration, the potential function *increases* by at most $\widetilde{O}(1)$ (see full version). The proof of this statement crucially uses

the structural property of interior point method that slack matrices in consecutive steps are sufficiently close to each other. Formally, for any iteration $t \in [T]$, we show in the full version that the consecutive slack matrices S_t and S_{t+1} satisfy

$$\|S_t^{-1/2} S_{t+1} S_t^{-1/2} - I\|_F = O(1) \tag{9}$$

and combine this bound with the Hoffman-Wielandt theorem [HJ12], which relates the ℓ_2 distance between the spectrum of two matrices with the Frobenius norm of their difference (see full version). Next, when $\widetilde{S}$ gets updated, we prove that our method of zeroing out the r_t largest eigenvalues of $|Z_t|$, thereby incurring a rank-r_t update to $\widetilde{S}_t$, results in a potential decrease of at least $\widetilde{O}(\sqrt{r_t})$ (see full version).

Maintaining rectangular matrix multiplication for Hessian computation: Given the low-rank update on $\widetilde{S}$ described above, we show how to efficiently update the *approximate Hessian* $\widetilde{H}$, defined as

$$\widetilde{H}_{j,k} = \mathrm{tr}[\widetilde{S}^{-1} A_j \widetilde{S}^{-1} A_k] \tag{10}$$

for each entry $(j,k) \in [m] \times [m]$. The approximate slack matrix $\widetilde{S}$ being a spectral approximation of the true slack matrix S implies that the approximate Hessian $\widetilde{H}$ is also a spectral approximation of the true Hessian H (see full version). This approximate Hessian therefore suffices for our algorithm to approximately follow the central path.

To efficiently update the approximate Hessian $\widetilde{H}$ in (10), we notice that a rank-r update on $\widetilde{S}$ implies a rank-r update on $\widetilde{S}^{-1}$ via the Woodbury matrix identity (see full version). The change in $\widetilde{S}^{-1}$ can be expressed as

$$\Delta(\widetilde{S}^{-1}) = V_+ V_+^\top - V_- V_-^\top, \tag{11}$$

where $V_+, V_- \in \mathbb{R}^{n \times r}$. Plugging (11) into (10), we can express $\Delta \widetilde{H}_{j,k}$ as the sum of multiple terms, among the costliest of which are those of the form $\mathrm{tr}[\widetilde{S}^{-1} A_j V V^\top A_k]$, where $V \in \mathbb{R}^{n \times r}$ is either V_+ or V_-. We compute $\mathrm{tr}[\widetilde{S}^{-1} A_j V V^\top A_k]$ for all $(j,k) \in [m] \times [m]$ in time $\mathcal{T}_{\mathrm{mat}}(r, n, mn)$ by first computing $V^\top A_k$ for all $k \in [m]$ by horizontally concatenating all A_k's into a wide matrix of size $n \times mn$. We then compute the product of $\widetilde{S}^{-1/2}$ with $A_j V$ for all $j \in [m]$, which can be done in time $\mathcal{T}_{\mathrm{mat}}(n, n, mr)$, which equals $\mathcal{T}_{\mathrm{mat}}(n, mr, n)$ (see full version). Finally, by flattening each $\widetilde{S}^{-1/2} A_j V$ into a vector of length nr and stacking all these vectors to form a matrix $\widetilde{\mathcal{B}} \in \mathbb{R}^{m \times nr}$ with j-th row

$$\widetilde{\mathcal{B}}_j = \mathsf{vec}(\widetilde{S}^{-1/2} A_j V),$$

the task of computing $\mathrm{tr}[\widetilde{S}^{-1} A_j V V^\top A_k]$ for all $(j,k) \in [m] \times [m]$ reduces to computing $\widetilde{\mathcal{B}} \widetilde{\mathcal{B}}^\top$, which costs $\mathcal{T}_{\mathrm{mat}}(m, nr, m)$.

In this way, we reduce the runtime of $T \cdot (\mathcal{T}_{\mathrm{mat}}(n, mn, n) + \mathcal{T}_{\mathrm{mat}}(m, n^2, m))$ for computing the Hessian using fast rectangular matrix multiplication down to

$$\sum_{t=1}^{T} \left(\mathcal{T}_{\mathrm{mat}}(r_t, n, mn) + \mathcal{T}_{\mathrm{mat}}(n, mr_t, n) + \mathcal{T}_{\mathrm{mat}}(m, nr_t, m) \right), \tag{12}$$

where r_t is the rank of the update on $\widetilde{S}_t$. Applying Theorem I.4 with several properties of fast rectangular matrix multiplication that we prove in the full version , we upper bound the runtime in (12) by

$$O^*(\sqrt{n}(mn^2 + m^\omega + n^\omega)),$$

which implies Theorem I.2. In Section I-B3 and I-B4, we discuss bottlenecks to further improving our runtime.

3) Bottlenecks of our interior point method: In most cases, the costliest term in our runtime is the per iteration cost of mn^2, which corresponds to reading the entire input in each iteration. Our subsequent discussions therefore focus on the steps in our algorithm that require at least mn^2 time per iteration.

Slack matrix computation: When y is updated in each iteration of our interior point method, we need to compute the true slack matrix S as

$$S = \sum_{i \in [m]} y_i A_i - C.$$

Computing S is needed to update the approximate slack matrix $\widetilde{S}$ so that $\widetilde{S}$ remains a spectral approximation to S. As S might suffer from full-rank changes, it naturally requires mn^2 time to compute in each iteration. This is the first appearance of the mn^2 cost per iteration.

Gradient computation: Recall from (3) that our interior point method follows the central path defined via the penalized objective function

$$\min_{y \in \mathbb{R}^m} f_\eta(y) \quad \text{where} \quad f_\eta(y) := \eta b^\top y + \phi(y),$$

for a parameter $\eta > 0$ and $\phi(y) = -\log\det S$. In each iteration, to perform the Newton step, the gradient of the penalized objective is computed as

$$g_\eta(y)_j = \eta \cdot b_j - \mathrm{tr}[S^{-1} A_j] \tag{13}$$

for each coordinate $j \in [m]$. Even if we are given S^{-1}, it still requires mn^2 time to compute (13) for all $j \in [m]$. This is the second appearance of the per iteration cost of mn^2.

Approximate Hessian computation: Recall from Section I-B2 that updating the approximate slack matrix S by rank r means the time needed to update the approximate Hessian is dominated by computing the term

$$\Delta_{j,k} = \mathrm{tr}[\widetilde{S}^{-1/2} A_j V \cdot V^\top A_k \widetilde{S}^{-1/2}],$$

where $V \in \mathbb{R}^{n \times r}$ is a tall, skinny matrix that comes from the spectral decomposition of $\Delta \widetilde{S}^{-1}$. Computing $\Delta_{j,k}$ for

all $(j,k) \in [m] \times [m]$ requires reading at least A_j for all $j \in [m]$, which takes time mn^2. This is the third bottleneck that leads to the mn^2 term in the cost per iteration.

4) LP techniques are unlikely to improve SDP runtime: The preceeding discussion of bottlenecks suggests that reading the entire input in each iteration, which takes mn^2 time per iteration, stands as a natural barrier to further improving the runtime of SDP solvers based on interior point methods.

In the context of linear programming (LP), several recent results [CLS19], [BLSS20], [JSWZ20] yield faster interior point methods that bypass reading the entire input in every iteration. Two techniques crucial to these results are: (1) showing that the Hessian (projection matrix) admits low-rank updates, and (2) speeding computation of the Hessian via sampling.

We now describe these techniques in the context of SDP and argue that they are unlikely to improve our runtime.

Showing that the Hessian admits low-rank updates: We saw in Section I-B2 that constructing an approximate slack matrix $\widetilde{S}$ that admits low-rank updates in each iterations leveraged the fact that the true slack matrix S changes "slowly" throughout our interior point method as described in (9). One natural question that follows is whether a similar upper bound can be obtained for the Hessian. If such a result could be proved, then one could maintain an approximate Hessian that admitted low-rank updates, which would speed up the approximate Hessian computation. Indeed, in the context of LP, such a bound for the Hessian can be proved (e.g., [BLSS20, Lemma 47]).

Unfortunately, it is impossible to prove such a statement for the Hessian in the context of SDP. To show this, it is convenient to express the Hessian using the Kronecker product full version as

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

where $\mathcal{A} \in \mathbb{R}^{n^2 \times m}$ is the $n^2 \times m$ matrix whose ith column is obtained by flattening A_i into a vector of length n^2. By proper scaling, we can assume without loss of generality that the current slack matrix is $S = I$, and the slack matrix in the next iteration is $S_{\text{new}} = I + \Delta S$, which satisfies $\|\Delta S\|_F = c$ for some tiny constant $c > 0$. Consider the simple example where $\mathcal{A} = I$ (we are assuming here that $m = n^2$ so that $\mathcal{A}$ is a square matrix), which implies that the change in the Hessian can be approximately computed as

$$\begin{aligned}
&\left\| H^{-1/2} \Delta H H^{-1/2} \right\|_F^2 \\
&\approx \operatorname{tr}\left[((I - \Delta S) \otimes (I - \Delta S) - I \otimes I)^2 \right] \\
&\approx \operatorname{tr}\left[(I \otimes \Delta S + \Delta S \otimes I)^2 \right] \\
&\geq 2 \cdot \operatorname{tr}[I^2] \cdot \operatorname{tr}\left[(\Delta S)^2 \right] \\
&= 2n \left\| \Delta S \right\|_F^2 \gg 1.
\end{aligned}$$

This large change indicates that we are unlikely to obtain an approximation to the Hessian that admits low-rank updates, which is a key difference between LP and SDP.

Sampling for faster Hessian computation: Recall from (8) that the Hessian can be computed as

$$H = \mathcal{B} \cdot \mathcal{B}^\top,$$

where the jth row of $\mathcal{B} \in \mathbb{R}^{m \times n^2}$ is $\mathcal{B}_j = \mathsf{vec}(S^{-1/2} A_j S^{-1/2})$ for all $j \in [m]$. We might attempt to approximately compute H faster by sampling a subset of columns of $\mathcal{B}$ indexed by $L \subseteq [n^2]$ and compute the product for only the sampled columns. This could reduce the dimension of the matrix multiplication and speed up the Hessian computation. Indeed, sampling techniques have been successfully used to obtain faster LP solvers [CLS19], [BLSS20].

For SDP, however, sampling is unlikely to speed up the Hessian computation. In general, we must sample at least m columns (i.e. $|L| \geq m$) of $\mathcal{B}$ to spectrally approximate H or the computed matrix will not be full rank. However, this requires computing the entries of $S^{-1/2} A_j S^{-1/2}$ that correspond to $L \subseteq [n^2]$ for all $j \in [m]$, which requires reading all A_j's and thus still takes $O(mn^2)$ time.

C. Related work

Linear Programming.: Linear Programming is a class of fundamental problems in convex optimization. There is a long list of work focused on fast algorithms for linear programming [Dan47], [Kha80], [Kar84], [Vai87], [Vai89b], [LS14], [LS15], [Sid15], [Lee16], [CLS19], [LSZ19], [Son19], [Bra20], [BLSS20], [JSWZ20].

Cutting Plane Method.: Cutting plane method is a class of optimization methods that iteratively refine a convex set that contains the optimal solution by querying a separation oracle. Since its introduction in the 1950s, there is a long line of work on obtaining fast cutting plane methods [Sho77], [YN76], [Kha80], [KTE88], [NN89], [Vai89a], [AV95], [BV02], [LSW15], [JLSW20].

First-Order SDP Algorithms.: As the focus of this paper, cutting plane methods and interior point methods solve SDPs in time that depends *logarithmically* on $1/\epsilon$, where ϵ is the accuracy parameter. A third class of algorithms, the *first-order methods*, solve SDPs at runtimes that depend *polynomially* on $1/\epsilon$. While having worse dependence on $1/\epsilon$ compared to IPM and CPM, these first-order algorithms usually have better dependence on the dimension. There is a long list of work on first-order methods for general SDP or special classes of SDP (e.g. Max-Cut SDP [AK07], [GH16], [AZL17], [CDST19], [LP19], [YTF+19], positive SDPs [JY11], [PT12], [ALO16], [JLL+20].)

Acknowledgment

We thank Aaron Sidford for many helpful discussions and Deeksha Adil, Sally Dong, Sandy Kaplan, and Kevin

Tian for useful feedback on the writing. We gratefully acknowledge funding from CCF-1749609, CCF-1740551, DMS-1839116, Microsoft Research Faculty Fellowship, and Sloan Research Fellowship. Zhao Song is partially supported by Ma Huateng Foundation, Schmidt Foundation, Simons Foundation, NSF, DARPA/SRC, Google and Amazon.

REFERENCES

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

[ALO16] 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)*, pages 1824–1831. https://arxiv.org/pdf/1507.02259.pdf, 2016.

[Ans00] Kurt M Anstreicher. The volumetric barrier for semidefinite programming. *Mathematics of Operations Research*, 25(3):365–380, 2000.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):1–37, 2009.

[AV95] David S Atkinson and Pravin M Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69(1-3):1–43, 1995.

[AZL17] Zeyuan Allen-Zhu and Yuanzhi Li. Follow the compressed leader: faster online learning of eigenvectors and faster mmwu. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 116–125, 2017.

[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)*, pages 1125–1135. https://arxiv.org/pdf/1811.01597.pdf, 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)*, pages 788–799. https://arxiv.org/pdf/1605.02882.pdf, 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)*, pages 914–926. https://arxiv.org/pdf/1611.01805.pdf, 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)*. https://arxiv.org/pdf/2002.02304.pdf, 2020.

[Bra20] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. https://arxiv.org/pdf/1910.11957.pdf, 2020.

[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)*, pages 2755–2771. SIAM, https://arxiv.org/pdf/1811.09380.pdf, 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)*. https://arxiv.org/pdf/1906.04661.pdf, 2019.

[CDST19] Yair Carmon, John C. Duchi, Aaron Sidford, and Kevin Tian. A rank-1 sketch for matrix multiplicative weights. In *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, pages 589–623, 2019.

[CG18] Yu Cheng and Rong Ge. Non-convex matrix completion against a semi-random adversary. In *Conference On Learning Theory (COLT)*, pages 1362–1394. https://arxiv.org/pdf/1803.10846.pdf, 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)*. https://arxiv.org/pdf/1810.07896.pdf, 2019.

[Dan47] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1947.

[GH16] Dan Garber and Elad Hazan. Sublinear time algorithms for approximate semidefinite programming. *Mathematical Programming*, 158(1-2):329–361, 2016.

[GLS81] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[GU18] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, 2018.

[GV02] Jean-Louis Goffin and Jean-Philippe Vial. Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. *Optimization methods and software*, 17(5):805–867, 2002.

[GW95] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[HJ12] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2012.

[JJUW11] Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP = PSPACE. *Journal of the ACM (JACM)*, 58(6):1–27, 2011.

[JLL+20] Arun Jambulapati, Yin Tat Lee, Jerry Li, Swati Padmanabhan, and Kevin Tian. Positive semidefinite programming: Mixed, parallel, and width-independent. In *STOC*. https://arxiv.org/pdf/2002.04830.pdf, 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.

[JSWZ20] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470*, 2020.

[JY11] Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, 2011.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing (STOC)*, pages 302–311, 1984.

[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)*, pages 2–13. IEEE, 1994.

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

[Lee16] Yin Tat Lee. *Faster algorithms for convex and combinatorial optimization*. PhD thesis, Massachusetts Institute of Technology, 2016.

[LP19] Yin Tat Lee and Swati Padmanabhan. An $O(m/\epsilon^{3.5})$-cost algorithm for semidefinite programs with diagonal constraints. In *arXiv preprint*. https://arxiv.org/pdf/1903.01859.pdf, 2019.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $O(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 424–433. https://arxiv.org/pdf/1312.6677.pdf, https://arxiv.org/pdf/1312.6713.pdf, 2014.

[LS15] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 230–249. https://arxiv.org/pdf/1503.01752.pdf, 2015.

[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)*, pages 1049–1065. https://arxiv.org/pdf/1508.04874.pdf, 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)*. https://arxiv.org/pdf/1905.04447, 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.

[PT12] Richard Peng and Kanat Tangwongsan. Faster and simpler width-independent parallel algorithms for positive semidefinite programming. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures (SPAA)*, pages 101–108, 2012.

[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. https://arxiv.org/pdf/1811.01057.pdf, 2018.

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

[Sid15] Aaron Daniel Sidford. *Iterative methods, combinatorial optimization, and linear programming beyond the universal barrier*. PhD thesis, Massachusetts Institute of Technology, 2015.

[Son19] Zhao Song. *Matrix Theory : Optimization, Concentration and Algorithms*. PhD thesis, The University of Texas at Austin, 2019.

[Vai87] Pravin M Vaidya. An algorithm for linear programming which requires $o(((m+n)n^2+(m+n)^{1.5}n)l)$ arithmetic operations. In *28th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1987.

[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.

[VB96] Lieven Vandenberghe and Stephen P. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[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.