An Improved Cutting Plane Method for Convex Optimization, Convex-Concave Games, and Its Applications

Haotian Jiang
University of Washington, USA
jhtdavid@cs.washington.edu

Zhao Song
Princeton University and Institute for Advanced Study
USA
zhao@ias.edu

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

Sam Chiu-wai Wong
Microsoft Research Redmond, USA
samwon@microsoft.com

ABSTRACT

Given a separation oracle for a convex set $K \subset \mathbb{R}^n$ that is contained in a box of radius $R$, the goal is to either compute a point in $K$ or prove that $K$ does not contain a ball of radius $\epsilon$. We propose a new cutting plane algorithm that uses an optimal $O(n \log(\kappa))$ evaluations of the oracle and an additional $O(n^2)$ time per evaluation, where $\kappa = nR/\epsilon$.

- This improves upon Vaidya’s $O(n \log(\kappa) + n^{\omega+1} \log(\kappa))$ time algorithm [Vaidya, FOCS 1989a] in terms of polynomial dependence on $n$, where $\omega < 2.373$ is the exponent of matrix multiplication and $SO$ is the time for oracle evaluation.
- This improves upon Lee-Sidford-Wong’s $O(n \log(\kappa) + n^3 \log(O(1)(\kappa)))$ time algorithm [Lee, Sidford and Wong, FOCS 2015] in terms of dependence on $\kappa$.

For many important applications in economics, $\kappa = \Omega(\exp(n))$ and this leads to a significant difference between $\log(\kappa)$ and $\log(O(1)(\kappa))$. We also provide evidence that the $n^2$ time per evaluation cannot be improved and thus our running time is optimal.

A bottleneck of previous cutting plane methods is to compute leverage scores, a measure of the relative importance of past constraints. Our result is achieved by a novel multi-layered data structure for leverage score maintenance, which is a sophisticated combination of diverse techniques such as random projection, batched low-rank update, inverse maintenance, polynomial interpolation, and fast rectangular matrix multiplication. Interestingly, our method requires a combination of different fast rectangular matrix multiplication algorithms.

Our algorithm not only works for the classical convex optimization setting, but also generalizes to convex-concave games. We apply our algorithm to improve the runtimes of many interesting problems, e.g., Linear Arrow-Debreu Markets, Fisher Markets, and Walrasian equilibrium.

1 INTRODUCTION

The cutting plane methods are a class of optimization primitives for solving Linear and Convex Programming, which is encapsulated by the feasibility problem of finding a point in a given convex set $K$ equipped with a separation oracle. In each iteration, cutting plane methods query the separation oracle which returns a hyperplane separating the query point from $K$. Since Khachiyan’s breakthrough result [Kha80] on the ellipsoid method, cutting plane methods have played a central role in theoretical computer science, showing that a plethora of problems from diverse areas admit polynomial time algorithms. Early prominent examples include linear programming and submodular function minimization.

While more applications of the cutting plane methods are still being discovered to this date, progress on faster cutting plane methods had stagnated until the recent work of Lee, Sidford and Wong [LSW15]. Prior to their work, cutting plane methods were considered too slow to be of relevance in theory or in practice (other than establishing polynomiality). [LSW15] debunked this by showing how their cutting plane method is faster than tailor-made algorithms for a long list of well-studied problems.

The fastest algorithm before their work was Vaidya’s algorithm [Vai89a], which runs in $O(n^{\omega+1})$ time. Here $\omega < 2.373$ is the exponent of matrix multiplication [CW87, Wil12, DS13, LG14]. Leveraging recent advances in optimization and numerical linear algebra, LSW lowered the dependence on the dimension $n$ at the expense of...
This extra overhead of \( \log^2(\kappa) \), as exemplified by various problems in combinatorial optimization in their paper, translates into only a \( \log^{2}\kappa \) factor in the maximum value \( M \) of the input by taking \( e = O(1/M) \). This is because solutions to (polynomial-time solvable) problems in combinatorial optimization are guaranteed to be integral. Despite being a nuisance, this small overhead is relatively mild and can even be absorbed completely for unweighted problems and strongly polynomial time algorithms. Indeed, armed with this faster cutting plane method, they improved the state-of-the-art running times for a host of problems such as semidefinite programming, matroid intersection and submodular minimization.

### Importance of \( \log(\kappa) \) vs \( \log^2(\kappa) \)

For many other problems including linear programming and market equilibrium computation, \( \epsilon \) must be taken as \( 1/M^{O(n)} \), resulting in an additional factor \( n^2 \) between \( \log(\kappa) \) and \( \log^2(\kappa) \). For these applications, LSW is slower than Vaidya’s by a factor of \( \tilde{O}(n^{1+\epsilon}) \). This raises a natural question: is there a cutting plane method that simultaneously runs in \( O(n \log(\kappa)) \) calls and \( O(n^3 \log(\kappa)) \) time? That is, can we achieve the best of both worlds of Vaidya’s and LSW methods in terms of the dependence on \( n \) and \( \kappa \)?

In this paper, we answer this question in the affirmative. Somewhat surprisingly, we are able to remove \( \log^{4.5}(n) \) dependence in LSW as well.

**Theorem 1.1 (Main result).** There is a cutting plane method which runs in time \( O(n \cdot \log(\kappa) + n^3 \log(\kappa)) \), where \( \log(\kappa) \) is the time complexity of the separation oracle.

As with previous methods, our result achieves the asymptotic optimal oracle complexity of \( O(n \cdot \log(\kappa)) \). Moreover, we conjecture that the runtime is likely the best possible. Note that in each iteration where a separation oracle call is made, even basic matrix operations require \( O(n^2) \) time. Thus our runtime is essentially tight unless properties like sparsity can be surprisingly exploited to update the feasible region and compute the next query point for the separation oracle.

Our result is obtained by marrying Vaidya’s method with recent advances in numerical linear algebra. Similar to LSW, we implement each iteration of Vaidya’s via tools from fast numerical linear algebra. The main issue is that such tools rely on approximation and would lead to errors which must be carefully controlled.

Our first innovation is to run Vaidya’s method in phases, each of which consist of a certain number of iterations. Between phases we “recompute” to eliminate the errors accumulated within a phase. Because of this recomputation we can afford to tolerate higher errors in an iteration. Secondly, we present a sophisticated data structure that enables us to implement each iteration of Vaidya’s efficiently. Our data structure leverages recent advances on applying numerical techniques to optimization. We hope that these numerical tools, as well as our approach to applying them, would play a greater role in future development of optimization.

#### 1.1 Applications

We highlight some of the key applications of our faster cutting plane method. Interestingly, even though our cutting plane method is a general purpose algorithm, we are able to improve the runtimes of tailor-made algorithms for various problems.

Using a standard reduction of convex minimization to the feasibility problem ([Nem94] and Theorem 42 of [LSW15]), we can minimize a convex function with an optimal \( \tilde{O}(n) \) subgradient oracle and an additional \( O(n^{4}) \) time per oracle call.

**Theorem 1.2.** Let \( f \) be a convex function on \( \mathbb{R}^n \) and \( S \) be a convex set that contains a minimizer of \( f \). Suppose we have a subgradient oracle for \( f \) with cost \( T \) and \( S \subset B(0,R) \). Using \( B(0,R) \) as the initial polytope for our Cutting Plane Method, for any \( 0 < \alpha < 1 \), we can compute \( x \in S \) such that \( f(x) = \min_{y \in S} f(y) \leq \alpha \left( \max_{y \in S} f(y) - \min_{y \in S} f(y) \right) \), with high probability in \( n \) and a running time of \( \Theta(T \cdot \log(\kappa) + n^3 \log(\kappa)) \), where \( \kappa = n\gamma/\alpha \) and \( \gamma = R/\text{minwidth}(S) \).

Our convex minimization result can be further generalized to convex-concave games.

**Theorem 1.3.** Given convex sets \( X \subset B(0,R) \subset \mathbb{R}^n \) and \( Y \subset B(0,R) \subset \mathbb{R}^m \) such that both \( X \) and \( Y \) contain a ball of radius \( r \). Let \( f(x,y) : X \times Y \rightarrow \mathbb{R} \) be an \( L \)-Lipschitz function that is convex in \( x \) and concave in \( y \). Define \( \kappa = \max_{x \in X} L^{-1} \max_{y \in Y} f(x,y) \). For any \( \epsilon \in (0,1/2] \), we can find \( (\tilde{x}, \tilde{y}) \) such that \( \max_{x \in X} \min_{y \in Y} f(x,y) \leq eL \) in time \( O(T \cdot (n+m) \log(\kappa) + (n+m)^3 \log(\kappa)) \) with high probability in \( n + m \) where \( T \) is the cost of computing subgradient \( \nabla f \).

Leveraging this improved dependence on \( \kappa \) (and hence \( \epsilon \)), our cutting plane method can be used to improve the runtimes of a wide range of problems, especially those on market equilibrium computation (see Table 2, 3 and 4 for a summary of previous runtimes). In all our applications, \( 1/\epsilon \) needs to be exponentially large in \( n \) which renders the \( \log(\kappa) \) factor polynomially large.

We show the following runtime improvement for the problem of computing a market equilibrium in linear exchange markets:

**Theorem 1.4.** There exists a weakly polynomial algorithm that computes a market equilibrium in linear exchange markets in time \( O(mn^2 \log(nU)) \).

The celebrated result of Arrow and Debreu [AD54] shows the existence of a market equilibrium for a broad class of utility functions. Since then researchers have attempted to design efficient algorithms to compute market equilibria. One prominent special case of linear utilities has enjoyed significant attention as demonstrated by the long line of work in Table 2. Essentially, this problem can be captured by a convex program with linear constraints [GV16]. While this convex program exhibits certain advantageous features over previous ones [Cor89, Jai07, NP83], it had not led to improved runtimes since the objective is not separable [GV19]. Moreover, the number of variables in the convex program can be as large as \( O(n^3) \), which prohibits a fast runtime for the cutting plane method if applied directly. Our approach in Theorem 1.4 is to transform the
Table 1: Algorithms for the Feasibility Problem. Let $\kappa = nR/\epsilon$. All methods can be used to solves a more general problem where only a membership oracle is given [LSV18].

| Reference | Year | Algorithm | Complexity |
|-----------|------|-----------|------------|
| [Sho77, YN76, Kha80] | 1979 | Ellipsoid Method | $n^\omega \log(k) + n^4 \log(k)$ |
| [KTE88, NN89] | 1988 | Inscribed Ellipsoid | $n \log(k) + (n \log(k))^4.3$ |
| [Vai89a] | 1989 | Volumetric Center | $n \log(k) + n^\omega \log(k)$ |
| [AV95] | 1995 | Analytic Center | $n \log^2(k) + n^{\omega+1} \log^2(k) + (n \log(k))^{2.10.2}$ |
| [BV02] | 2002 | Random Walk | $n \log(k) + n^{\omega} \log(k)$ |
| [LSW15] | 2015 | Hybrid Center | $n \log(k) + n^{\omega} \log(k)$ |

Table 2: Linear Arrow-Debreu Markets. Let $n$ be the number of agents and $m$ the number of edges. Each operation of the given algorithms involves $O(n \log(nU))$-bit numbers.

| Reference | Year | #Operations | Poly Type |
|-----------|------|-------------|-----------|
| [Eav75] | 1975 | Finite | Not poly |
| [Jai07] | 2007 | Polynomial | Weakly poly |
| [Ye08] | 2008 | $n^6 \log(nU)$ | Weakly poly |
| [DPSV08] | 2008 | Polynomial | Weakly poly |
| [DM15] | 2015 | $n^\omega \log(nU)$ | Weakly poly |
| [DGM16] | 2016 | $n^6 \log^4(nU)$ | Weakly poly |
| [GV19] | 2019 | $mn^3 \log^3(n)$ | Strongly poly |
| Theorem 1.4 | 2019 | $mn^2 \log(nU)$ | Weakly poly |

Table 3: Fisher Markets with Spending Constraint Utilities. Let $n$ denote the total number of buyers and sellers, and $m$ the total number of segments. $T_{\text{max-flow}}$ denotes the number of operations needed for a max-flow computation. Each operation of the given algorithms involves $O(n \log(nU))$-bit numbers.

| Reference | Year | # Operations | Poly Type |
|-----------|------|--------------|-----------|
| [Vaz10] | 2010 | $n^2(n + m)^2 \log(U) \cdot T_{\text{max-flow}}$ | Weakly poly |
| [Vég16] | 2016 | $mn^2 + m^2(n \log(n) + m)$ | Strongly poly |
| [Wan16] | 2016 | $mn^2 + m^2 \log(n)/n \log(n)$ | Strongly poly |
| Theorem 1.5 | 2019 | $mn^2 \log(nU)$ | Weakly poly |

Table 4: Walrasian equilibrium for general buyer valuations and fixed supply. Let $T_{\text{AD}}$ denotes the runtime of aggregate demand oracle, $n$ denotes the number of goods.

| Reference | Year | # Operations | Poly Type |
|-----------|------|--------------|-----------|
| [KJC82] | 1982 | Finite | Not poly |
| [Par99] | 1999 | Finite | Pseudo poly |
| [PU02] | 2002 | Finite | Pseudo poly |
| [AM02] | 2002 | Finite | Pseudo poly |
| [dVSV07] | 2007 | Finite | Pseudo poly |
| [LW17] | 2017 | $n^2 T_{\text{AD}} \log(SMn) + n^6 \log^{C(1)}(SMn)$ | Weak poly |
| Theorem 1.6 | 2019 | $n^2 T_{\text{AD}} \log(SMn) + n^4 \log(SMn)$ | Weak poly |

convex program into a convex-concave game with $O(n)$ variables and apply Theorem 1.3.

A similar technique can be applied to the problem of computing market equilibrium for Fisher markets with spending utility constraints where a convex program is given in [BDX10]. However, directly transforming it into a convex-concave game does not reduce the dimension of the variables, as both the number of variables and constraints in the original convex program is $\Theta(m)$, where $m$ is the total number of segments and it can be much larger than $n^2$. In order to reduce the dimension of the convex-concave game to $O(n)$, we express part of the variables as functions of $O(n)$ variables and show that this does not increase the time of the first-order oracle. This leads to the following runtime improvement:
Theorem 1.5. There exists a weakly polynomial algorithm that computes a market equilibrium in Fisher markets with spending constraint utilities in time $O(mn^2 \log(nU))$.

Yet another economics application of our cutting plane method is the problem of computing a Walrasian equilibrium in a market with fixed supply. In this economy, buyers may have arbitrary valuation functions and we would like to compute prices so that the market clears, i.e. the aggregate demand of the buyers matches the fixed supply. Recently Paes Leme and Wong [LW17] gave a polynomial time algorithm for this problem provided that an equilibrium actually exists. They achieved this by showing that the cutting plane method can be modified so that the convex program for the equilibrium can be solved under the aggregate demand oracle.

By leveraging our faster cutting plane method we obtain an improved runtime:

Theorem 1.6. There is an algorithm for computing a market equilibrium in an economy with general buyer valuation in the aggregate demand model that runs in time $O(n^3T_{AD} \log(SMn) + n^3 \log(SMn))$.

1.2 Previous Works

The cutting plane methods solve the following feasibility problem that conveniently abstracts the applications to specific scenarios.

Feasibility Problem: Given a separation oracle for a set $K$ contained in a box of radius $R$ either find a point $x \in K$ or prove that $K$ does not contain a ball of radius $\epsilon$.

All cutting plane methods maintain a candidate region $\Omega$ and solve the feasibility problem by iteratively refining $\Omega$ based on the present $\Omega$ and the new separating hyperplane. In each iteration:
1. The separation oracle is queried at some point $x \in \Omega$.
2. If $x \in K$ we have solved the feasibility problem.
3. Otherwise, the separation oracle returns a separating hyperplane from which $\Omega$ is further refined and the next query point $x$ is computed.

Previous works differ in how $x$ is selected and how $\Omega$ is refined. For instance, the classic ellipsoid method maintains $\Omega$ as an ellipsoid and $x$ as its center. Given $\Omega$ and the new separating hyperplane, the new $\Omega$ is chosen to be the smallest ellipsoid containing their intersection. Table 1 lists the running times for solving the feasibility problem in the literature.

We focus our discussion on the trade-off between the oracle complexity and the runtime per iteration. The oracle complexity has a lower bound $\Omega(n \log(n))$ [NY83]. While the ellipsoid method achieves a suboptimal $O(n^2 \log(n))$ in oracle complexity, the runtime per iteration is $O(n^5)$ which is faster than all subsequent methods. The good runtime follows from the simple calculations needed to update the ellipsoid, whereas the suboptimal oracle complexity can be attributed to the "looseness" of maintaining only an ellipsoid as a proxy to the intersection of past separating half-spaces.

Indeed, one can attain the optimal oracle complexity $O(n^2 \log(n))$ by maintaining all previous separating half-spaces. This is the random walk method [BV02] where the query point $x$ is chosen to be its (approximate) center of gravity. Updating $x$ involves performing a random walk in this polytope and is computationally expensive.

Other cutting methods improve oracle complexity by maintaining more fine-grained information about past separating hyperplanes in a way that is computationally friendly. Of particular relevance is Vaidya’s volumetric center method [Vai89a]. Vaidya’s $\Omega$, similar to the random walk method, is also a polytope $\Omega = \{x \in \mathbb{R}^n : Ax \geq b\}$. As the name suggests, the query point is chosen to be the volumetric center, which is the minimizer of the following convex function defined by the feasible region $\Omega$:

$$
\frac{1}{2} \log \det \left( A^T S^{-2} A \right)
$$

where $S_x := \text{diag}(Ax - b)$ is the diagonal matrix of the slacks.

Nevertheless, by judiciously including only a representative subset of previous half-spaces $\alpha x \geq b$, Vaidya showed that the volumetric center and $\Omega$ can be updated by basic matrix operations which run in $O(n^2 \log n)$ time.

We defer the discussion of LSW to the next subsection, which explains how Vaidya’s method can be sped up using machineries from numerical linear algebra.

1.3 Lee-Sidford-Wong Method (LSW)

LSW’s key observation is that Vaidya’s method relies heavily on leverage scores, which measure the relative importance of each separating hyperplane. In Vaidya’s method, naively updating leverage scores requires $O(n^3)$ time and is a bottleneck. Inspired by the work of Spielman and Srivastava [SS11], LSW attempted to address this by using random Johnson-Lindenstrauss [JL84] (JL) projection to approximate changes in leverage scores. Leverage scores can then be updated by summing over the differences.

Approximating leverage scores changes via JL projection however still requires solving a linear system which would still take $O(n^3)$ time. LSW further overcame this barrier by resorting to a recent work that efficiently solves “slowly-changing” linear system in amortized $O(n^2)$ time. Thus after paying $O(n^3)$ initially, they can solve such linear systems in $O(n^2)$ time per iteration.

Error accumulation. Nevertheless, as an approximate method JL introduces errors which would accumulate across iterations. While Vaidya’s method tolerates small errors in leverage scores, the total errors incurred in JL projection (as accumulated across iterations) would eventually become too big and destroy the performance guarantee of Vaidya’s method.

LSW handled this by modifying Vaidya’s framework to take into account of the error in the convex function to be minimized. This approach gives rise to a “hybrid” center algorithm, which involves a complicated interplay of optimization and linear algebra. In particular, to reduce the error accumulated the error parameter $\epsilon_0$ in JL projection has to be as small as $\epsilon_0 = 1/(\log(n))$. As the runtime of JL depends on $1/\epsilon_0^2$, this unfortunately introduces the $\log^2 n$ overhead in LSW runtime when compared to Vaidya’s volumetric center method.

1.4 Overview of Our Approach

To achieve the desired $O(nS\log(n) + n^3 \log(n))$ runtime, we build a data structure that approximates changes in leverage scores to within $c$ in $\ell_2$ norm for small enough constant $c$. This would suffice for Vaidya’s cutting plane method. The desired runtime then directly follows from the performance guarantee of the data structure,
which handles each iteration in amortized $O(n^2)$ time. Designing such a data structure requires several new ideas to control the error accumulation issue.

Our data structure employs a layered approach where different layers are associated with different error tolerances, and achieve different accuracy-efficiency tradeoffs in approximating the changes in leverage scores. The more inner a layer is, the more error it can tolerate and the faster is the runtime. Whenever the error accumulated in a layer becomes too much, the layer above would take over and produce a finer error estimate which would, of course, be more time costly. But because of our layered approach, the higher layer is called on less often and afford to spend more time.

Such a layered approach further leads to the following issue. In the middle and outer layers, we batch the updates of multiple steps into one which allows us to make use of fast rectangular matrix multiplication. However, our algorithm needs to handle possibly exponential weight changes. We show there are not too many such weights and can be handled separately in groups of $\log n$ size using low-rank update formula.

Our data structure also draws on various numerical tools such as fast rectangular matrix multiplication, "tall" JL projection, preconditioning, inverse maintenance, and polynomial interpolation for approximating integrals.

A more in-depth discussion of our techniques can be found in Section 2.

1.5 Discussion of Optimality

Similar to previous methods our cutting plane method achieves the optimal oracle complexity $n \log(n)$ [NY83]. We present some evidence that our running time of $O(n^3 \log(\kappa))$ is also tight.

A bottleneck of Vaidya’s method is to solve the inverse maintenance problem. Formally, given a sequence of positive vectors $w^1, w^2, \cdots w^T$, let $P(w)$ be defined as

$$P(w) = \sqrt{W} A (A^T W A)^{-1} A^T \sqrt{W},$$

where $W$ is the diagonal matrix such that $W_{ii} = w_i$. The goal is to output a sequence of vectors $v^1, v^2, \cdots, v^T$ such that $v^t \approx w^t \text{ and } P(v^t) \approx P(w^t), \forall t \in [T]$.

There is a long line of research on inverse maintenance and dynamic matrix data-structure problems [Kha80, Vai89b, San04, LS15, HKNS15, CLS19, LSZ19, Son19, BNS19, Bra20]. This task can be done naively by spending $n^2$ time so the goal is to achieve $O(n^2)$ amortized cost per iteration. For example, in the LP setting the number of iterations is $O(\sqrt{n})$ and Vaidya [Vai89b] combined fast matrix multiplication with inverse maintenance to achieve $O(n^2)$ amortized cost per iteration, which gives an $O(n^{3.5})$ time algorithm. This remained a barrier until recent works [CLS19, LSZ19] combined sampling and sketching techniques with fast rectangular matrix multiplication and inverse matrix maintenance to give an $O(n^{2.5})$ time algorithm.

One of the major computation required in each step is matrix-vector multiplication, e.g., $P(w) \cdot h$. Naively, this step takes $O(n^2)$ time per iteration. To achieve $O(n^2)$ amortized cost per iteration, previous works [CLS19, LSZ19] used an idea called “iterating and sketching” which was formally described in [Son19]. This idea is very different from the classical “sketch and solve” [CW13] and “guess a sketch” [RSW16]. The classical idea usually applying sketching matrices only once without modifying the solver itself. However, the “iterating and sketching” idea has to modify the solver and applying sketching/sampling matrices over each iteration.

In [CLS19], they use the diagonal sampling matrix $D \in \mathbb{R}^{n \times n}$ which has roughly $\sqrt{n}$ nonzeros. They use that sampling matrix to sample on the right hand side:

$$\sqrt{W} A (A^T W A)^{-1} A^T \sqrt{W} D h.$$

In [LSZ19], they use the subsampled randomized Hadamard/Fourier matrix $R \in \mathbb{R}^{n \times n}$. They use the sketching matrix to sample on the left hand side:

$$R^T R \sqrt{W} A (A^T W A)^{-1} A^T \sqrt{W} h.$$

Compared to the cutting plane method, LP is an easier maintenance task as the matrix $A$ is fixed throughout. In the cutting plane method, however, rows get inserted into or deleted from $A$ from continuously. One critical idea used in all previous works on LP [Vai89b, CLS19, LSZ19] is to delay low-rank updates on $(A^T W A)^{-1}$. However, in the cutting plane method, the low rank updates to $A$ cannot be delayed. Thus it appears that previous techniques are inapplicable. Moreover, $n^2$ lower bounds have recently been established for natural matrix maintenance tasks (e.g. determinant, inverse) with row/column insertions/deletions under the Online Matrix-Vector conjecture (e.g. [HKNS15, BNS19]). Therefore, we believe our algorithm is tight and conjecture the following:

**Conjecture 1.7.** Solving the feasibility problem requires time $O(n^2 \log(n))$. Hence our cutting plane method achieves the optimal runtime.

1.6 Related Works

Leverage scores. Leverage scores are a fundamental concept in graph problems and numerical linear algebra. There are many works about how to approximate leverage scores [SS11, DMIMW12, CW13, NN13] or more general version of leverages, e.g. Lewis weights [Lew78, BLM89, CP15] and ridge leverage scores [CM17].

From graph perspective, it was applied to solve max-flow [Mad13], generate random spanning trees [Sch18], and sparsify graphs [SS11]. From matrix perspective, it was used to give matrix CUR decomposition [BW14, SWZ17, SWZ19] and tensor CUR decomposition [SWZ19]. From optimization perspective, it was used for approximating the John Ellipsoid [CCLY19], accelerating the kernel methods [AKM17, AKM19], showing the convergence of the deep neural network [LSS20], cutting plane methods, e.g. [Vai89a, LS15] and this paper.

**Linear Program.** Linear Program is a fundamental problem in convex optimization and can be treated as an special case where one can apply the cutting plane method. There is a super long list of work focused on fast algorithms for linear program [Dan47, Kha80, Kar84, Vai87, Vai89b, LS14, LS15, Sid15, Lee16, CLS19, LSZ19, Son19, Bra20, BLS20].
Membership oracle. Besides the separation oracle considered in this paper, there is another line of work on using the membership oracle to solve the feasibility problem [Pro96, KV06, LV06, GLS12, LSV18]. For a query point \( x \), this oracle outputs \( x \in K \) or \( x \notin K \).

## 2 OUR TECHNIQUES

Section 1.4 provided a quick overview. Here we take a deeper dive into our techniques.

### 2.1 Efficient Approximation of Changes in Leverage Scores

The key to fast maintenance of leverage scores is an efficient way to approximate their changes between consecutive steps. While a fine-grained approximation leads to an accurate approximation, the time to compute such an approximation might be unaffordable. On the other hand, a coarse-grained approximation can be efficiently computed, but might lead to accumulating errors that blow up after a small number of steps. This leads to a tradeoff between accuracy and efficiency.

Central to our data structure are a coarse-grained formula and a fine-grained formula for the change in leverage scores. While the coarse-grained formula approximates the leverage score’s change via a single integral, the fine-grained formula is a cocktail involving cutting plane method and handles each iteration in amortized \( \ell \)-structure and interpolates between accuracy and efficiency. Specifically, our layered data structure contains three layers. The inner and middle layers both employ the simple data structure to achieve computational efficiency while the outer layer uses the complicated data structure to ensure a low error.

Our layered approach builds on several fundamental results on matrix multiplication which we summarize in Theorem 2.1. We remark that Vaidya [Vai89b] used the first result, a recent LP solver [CLS19] used the first two, while our cutting plane method crucially depends on all the results in the table.

**Theorem 2.1 (Fast rectangular matrix multiplication results).** For any \( n, r > 0 \), denote \( T_{\text{mat}}(n, r, r) \) the time to compute the multiplication of an \( n \times n \) matrix and an \( n \times r \) matrix\(^{1}\). Then we have:

| Reference | \( r \) | \( T_{\text{mat}}(n, n, r) \) | Layer |
|-----------|-------|-----------------|-------|
| [LG14]    | \( n \) | \( O(n^{\omega+o(1)}) \) | Restart |
| [GU18]    | \( n^{\omega-31} \) | \( O(n^{2+o(1)}) \) | Outer layer |
| [Cop82]   | \( n^{1/2-\epsilon} \) | \( O(n^{\omega} \log^2 n) \) | Middle layer |
| [BD76]    | \( \log^2 O(1) n \) | \( O(n^2) \) | Inner layer |

Our data structure also draws on various numerical tools such as fast rectangular matrix multiplication, “tall” JL projection, preconditioning, inverse maintenance, and polynomial interpolation for approximating integrals, which we discuss in more details below.

### 2.2 Layered Data Structure

To achieve the best of both worlds of the simple and complicated data structures, we propose a data structure that combines both data structures and interpolates between accuracy and efficiency. Specifically, our data structure approximates changes in leverage scores to \( \ell_2 \)-error within \( 1/\log O(1/n) \) which would suffice for Vaidya’s cutting plane method and handles each iteration in amortized \( O(n^2) \) time.

Our data structure employs a layered approach (see Figure 1) where different layers are associated with different error tolerances. The more inner a layer is, the more error it can tolerate and the faster is the runtime. Whenever the error accumulated in a layer becomes too high, the layer above would take over and produce a finer error estimate which would, of course, be more time costly. But because of our layered approach, the higher layer is called on less often and can afford to spend more time. More specifically, our layered data structure contains three layers. The inner and middle layers both employ the simple data structure to achieve computational efficiency while the outer layer uses the complicated data structure to ensure a low error.

To rescue this preconditioner idea, we transform and split the sequence into pieces of size \( \text{poly} \log(n) \). We ensure the weight changes by only a quasi-polynomial factor and this decreases the cost of

\[^{1}\text{Note that } T_{\text{mat}}(n, n, r) = T_{\text{mat}}(n, r, n)\]
Figure 1: Illustration of our three-level data structure with $T_{\text{inn}} = 2$, $T_{\text{mid}} = 3$ and $T_{\text{out}} = 4$ approximating the leverage scores of the sequence \{w, w, ..., w\}. The three levels maintain three approximate sequences \{v_{\text{inn}}^1, v_{\text{inn}}^2, ..., v_{\text{inn}}^4\}, \{v_{\text{mid}}^1, v_{\text{mid}}^2, ..., v_{\text{mid}}^3\} and \{v_{\text{out}}^1, v_{\text{out}}^2, ..., v_{\text{out}}^4\}, with errors $\epsilon_{\text{inn}} = ||\log(w) - \log(v_{\text{inn}})||_\infty$, $\epsilon_{\text{mid}} = ||\log(w) - \log(v_{\text{mid}})||_\infty$ and $\epsilon_{\text{out}} = ||\log(w) - \log(v_{\text{out}})||_\infty$ that satisfy $1 \gg \epsilon_{\text{inn}} \gg \epsilon_{\text{mid}} \gg \epsilon_{\text{out}} > 0$. The inner level takes a step for every $w$-update, the middle step takes a step in every $T_{\text{inn}}$ inner steps, and the outer step takes a step in every $T_{\text{out}}$ outer steps. Each middle step refines the inner approximations of leverage scores, and each outer step refines the approximation of both the inner and middle approximations of the leverage scores. For the actual choice of the parameters $T_{\text{inn}}, T_{\text{mid}}, T_{\text{out}}$ and $\epsilon_{\text{inn}}, \epsilon_{\text{mid}}, \epsilon_{\text{out}}$ in our data structure.

2.4 Illustration of Our Analysis

We describe the numerical tools used in our analysis, and provide simple illustrations of our applications of these tools.

Discrete sampling for multiple variable integrals. The most standard way to approximate an integral is by discretization, which takes a weighted sum of the integrand over a set of points in the domain. Unlike common discretization tools like the trapezoidal method, for our purpose we need to interpolate multiple variable integral using a polynomial for higher accuracy. We give a simple example to illustrate our application of polynomial interpolation for multiple variable integrals as follows. In our fine-grained formula of the leverage score change from $w$ to $w_{\text{new}}$, one of the integral terms is

$$
\sigma_{i,\text{cts}} = \int_0^1 \int_0^1 \int_0^1 Y_{i,s,t}^{-1} Y_{i,s',t} d\omega d\omega' dt,
$$

where

$$
Y_{i,s,t} = \sqrt{W_{\text{mid}} - W_{\text{new}}} \cdot Q(y_{s,t}) \cdot (Z_t - X_t) \cdot Q(y_{s,t}) \cdot \sqrt{W_{\text{new}}} \cdot \epsilon_i.
$$

In order to approximate such an integral, we take a set $T \subseteq [0, 1]$ of $N = \log^{O(1)}(n)$ points along the integration together with weights $(\omega_i)_{i \in T}$, and approximate the integral by

$$
\sigma_{i,\text{cts}} = \sum_{i \in T} \sum_{s \in S} \sum_{s' \in S} \omega_i \omega_s \omega_{s'} Y_{i,s,t}^{-1} Y_{i,s',t}.
$$

We show that the $\ell_2$-error can be bounded as $||\sigma_{\text{cts}} - \sigma_{\text{cts}}||_2 \leq \text{poly}(n)/22N$, which is negligible by our choice of $N = \log^{O(1)}(n)$.

Projection maintenance and preconditioning. Inverse maintenance was first proposed in [Kha80] as a method for solving "slowly-changing" linear system.

Given a sequence of positive vectors $w, w, ..., w^T \in \mathbb{R}^n$, let $P(w)$ be defined as $P(w) = \sqrt{W}A^{-1}W^{-1/2} \sqrt{W}$, where $W \in \mathbb{R}^{m \times m}$ is the diagonal matrix with $W_{i,i} = w_i$. The goal is to output a sequence of vectors $v^1, v^2, ..., v^T$ such that $(1 - \epsilon)v^t \leq w^t \leq (1 + \epsilon)v^t$, $\forall t \in [T]$, and efficiently computes $P(v^t)_u$, for query vector $u \in \mathbb{R}^n$. The recent work of [CLS19] gave an efficient way to perform such a task. For our purpose, however, rather than just computing $P(v^t)_u$, we also need explicit approximations to the matrices $Q(w) = A^{-1}W^{-1/2}A^{-1}$ and $M(w)^{-1} = (A^{-1}W^{-1/2}A)^{-1}$. The matrices $Q(w)$ and $M(w)^{-1}$ appear frequently in our formula for the changes in leverage scores, and we need their approximations as pre-conditioners for accelerating the computation of certain matrix rectangular multiplication involving $Q(w)$ and $M(w)^{-1}$.

"Tall" JL & fast rectangular matrix multiplication. From the previous paragraph on discrete sampling, it suffices to compute $Y_{i,s,t}$ for all $i$, where

$$
Y_{i,s,t} = \sqrt{W_{\text{mid}} - W_{\text{new}}} \cdot Q(y_{s,t}) \cdot (Z_t - X_t) \cdot Q(y_{s,t}) \cdot \sqrt{W_{\text{new}}} \cdot \epsilon_i.
$$

Notice that computing $Y_{i,s,t}$ for all $i$ is essentially computing the matrix products

$$
\sqrt{W_{\text{mid}} - W_{\text{new}}} \cdot Q(y_{s,t}) \cdot (Z_t - X_t) \cdot Q(y_{s,t}) \cdot \sqrt{W_{\text{new}}},
$$

which would take $O(n^{\omega(1)} \log(n))$ time if computed exactly. To improve the time while ensuring keeping the error small, we invoke JL with dimension $n^c$ for small constant $c$ by computing

$$
\sigma_{i,j \beta} = \sum_{t \in T} \sum_{s \in S} \sum_{s' \in S} \omega_i \omega_s \omega_{s'} Y_{i,s,t}^{-1} Y_{j,s',t}^{-1} R_{Y,s',t} R_{Y,s',t} \cdot Y_{i,s',t}.
$$
where $R_{t,s,t'} \in \mathbb{R}^{n \times n}$ is a random matrix. It is essential that $c$ is picked such that the rectangular matrix multiplication can be done in time roughly $n^2$. To obtain a small error, $\sigma_{\text{cut}}$ should be a good estimate with a small variance. We note that $\sigma_{\text{cut}}$ is indeed an unbiased estimator of $\sigma_{\text{cut}}$, and its variance can be bounded as $\sum_{i=1}^n \text{Var}[\sigma_{i,j}] \leq O(e^2/n^2)$, where $\epsilon$ is the error for the projection maintenance used by the data structure. Leveraging fast rectangular matrix multiplication, we pick $c = 0.31$ and $\epsilon = n^{-0.1}$. The variance would then be bounded by $n^{-0.51}$ which is sufficiently small after $n^{\alpha-2+\omega(1)}$ steps before the data structure restarts.

2.5 Much Faster Rectangular Matrix Multiplication Implies Deterministic Cutting Plane Method

Our algorithm crucially relies on different kinds of fast rectangular matrix multiplication results. We also show that if these results are improved, then we are able to get a deterministic cutting plane method immediately.

**Corollary 2.2.** If $T_{\text{mat}}(n,n,r) = O(n^2 \log^{O(1)}(n))$ for $r = n^\theta$ with $\theta > 2/3$, then there is a deterministic cutting plane method which runs in time

$$O(n \cdot \text{SO log}(n) + n^3 \log(n)),$$

where SO is the time complexity of the separation oracle.

Let $\alpha$ denote the dual exponent of matrix multiplication, which is the largest number $\alpha > 0$ such that $T_{\text{mat}}(n,n,n^\alpha) = n^{3+\omega(1)}$. Let $\beta$ denote the largest number such that $T_{\text{mat}}(n,n,n^{\beta}) = n^2 \log^{O(1)}(n)$. A very recent result by Christandl, Le Gall, Lysikov and Zuiddam [CGLZ20] showed the limitations of several tensor techniques: they proved that $\alpha < 0.625$ for certain tensors. We believe our work initiated two interesting open questions in the area of fast matrix multiplication: (1) whether one can prove a better upper bound on $\beta$ (compared to $\alpha$) for certain tensor techniques, and (2) if there is a non-trivial inequality between $\beta$ and $\alpha$.

**ACKNOWLEDGMENTS**

The full version of this paper [JLSW20] is available on arXiv, https://arxiv.org/pdf/2004.04250.pdf.

The authors would like to thank the anonymous referees of STOC 2020 for their helpful comments and suggestions. The authors would like to express their sincere gratitude to matrix multiplicationer Josh Alman for patiently answering our exponential number of questions about fast matrix multiplication.

The authors would like to thank Swati Padmanabhan for very useful discussions at the early stage of this project. The authors would like to thank Lijie Chen, Nikhil Devanur, Irit Dinur, Simon S. Du, Wei Hu, Jason Lee, Jerry Li, Ruqi Shen, Aaron Schild, Aaron Sidford, Santosh Vempala, Xin Yang, Petin Zhong, and Danyang Zhuo for helpful discussions. The authors would like to thank Josh Alman, Irit Dinur, Avi Wigderson, and Jeroen Zuiddam for useful discussion about limitation of fast matrix multiplication.

The authors would like to thank Sanjeev Arora, Alexander Andoni, Ainesh Bakshi, Yangsibo Huang, Rajesh Jayaram, Ravindran Kannan, Michael Kapralov, Adam Klivans, Kai Li, Christos Papadimitriou, Eric Price, Daniel Roy, Clifford Stein, Omri Weinstein, David P. Woodruff, and Hengjie Zhang for asking interesting questions at the end of the talk of this paper.

This project was supported in part by NSF awards CCF-1749609, CCF-1740551, DMS-1839116, and Microsoft Research Faculty Fellowship. This project was supported in part by Special Year on Optimization, Statistics, and Theoretical Machine Learning (being led by Sanjeev Arora) at Institute for Advanced Study.

**REFERENCES**

[AD54] Kenneth J Arrow and Gerard Debreu. Existence of an equilibrium for a competitive economy. Econometrica: Journal of the Econometric Society, pages 265–290, 1954.

[AKM*17] Haim Avron, Michael Kapralov, Cameron Musco, Christopher Musco, Ameya Velingker, and Amir Zandieh. Random Fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In International Conference on Machine Learning (ICML). https://arxiv.org/pdf/1804.09893.pdf, 2017.

[AKM*19] Haim Avron, Michael Kapralov, Cameron Musco, Christopher Musco, Ameya Velingker, and Amir Zandieh. A universal sampling method for reconstructing signals with simple Fourier transforms. In 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC). https://arxiv.org/pdf/1812.08723.pdf, 2019.

[AM02] Lawrence M Ausubel and Paul R Milgrom. Ascending auctions with package bidding. Advances in Theoretical Economics, 1(1), 2002.

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

[BD76] Roger W Brockett and David Dobkin. On the number of multiplications required for matrix multiplication. SIAM Journal on Computing, 3(4):624–628, 1976.

[BDX10] Benjamin Birnbaum, N. Devanur, and Lin Xiao. New convex programs and distributed algorithms for fisher markets with linear and spending constraint utilities. Unpublished manuscript, 2010.

[BLLM89] Jean Bourgain, Joram Lindenstrauss, and V. Milman. Approximation of zonoids by zonotopes. Acta mathematica, 162(1):73–141, 1989.

[BLS20] 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.

[BN59] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS). https://arxiv.org/pdf/1905.05067.pdf, 2019.

[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 thrity-fourth annual ACM symposium on Theory of computing (STOC). pages 109–115. ACM, 2002.

[BW14] Christos Boutsidis and David P Woodruff. Optimal lower bounds for decompositions. In Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC), pages 353–362. ACM, https://arxiv.org/pdf/1405.7910, 2014.

[CCLY19] Michael B Cohen, Ben Cousins, Yin Tat Lee, and Xin Yang. A near-optimal algorithm for approximating the John ellipsoid. In Annual Conference on Learning Theory (COLT). https://arxiv.org/pdf/1905.11580.pdf, 2019.

[CGLZ20] Matthias Christandl, François Le Gall, Vladimir Lysikov, and Jeroen Zuiddam. Barriers for rectangular matrix multiplication. In arXiv preprint. https://arxiv.org/2003.03019.pdf, 2020.

[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/1810.07696.pdf, 2019.

[CMM17] Michael B Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1758–1777. SIAM, https://arxiv.org/pdf/1511.07263.pdf, 2017.

[Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. SIAM Journal on Computing, 11(3):467–471, 1982.

[Cor89] Bernard Cornet. Linear exchange economies. Cahier Eco-Math, Université de Paris, 1, 1989.
[PU02] David C Parkes and Lyle H Ungar. An ascending-price generalized vickrey auction. 2002.
[RSW16] Ilya Razenshteyn, Zhao Song, and David P Woodruff. Weighted low rank approximations with provable guarantees. In Proceedings of the 48th Annual Symposium on the Theory of Computing (STOC), 2016.
[San04] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 509–517. IEEE, 2004.
[Sch18] Aaron Sidford. An almost-linear time algorithm for uniform random spanning tree generation. In Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC), pages 214–227. ACM, https://arxiv.org/pdf/1711.06455.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.
[SS11] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In SIAM Journal on Computing, volume 40(6), pages 1913–1926. https://arxiv.org/pdf/0803.0929.pdf, 2011.
[SWZ17] Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise \( \ell_1 \)-norm error. In Proceedings of the 49th Annual Symposium on the Theory of Computing (STOC). ACM, https://arxiv.org/pdf/1611.00898.pdf, 2017.
[SWZ19] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In ACM-SIAM Symposium on Discrete Algorithms (SODA). https://arxiv.org/pdf/1704.08246, 2019.
[Vas87] Pravin M Vaidya. An algorithm for linear programming which requires\( \alpha((m+n)n^2 + (m+n)^{1.5}n) \) 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.
[Vaz10] Vijay V Vazirani. Spending constraint utilities with applications to the adwords market. Mathematics of Operations Research, 35(2):458–478, 2010.
[Vég16] László A Végh. A strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives. volume 45(5), pages 1729–1761. https://arxiv.org/pdf/1110.4882.pdf, 2016.
[Wan16] Zi Wang. A New Strongly Polynomial Algorithm for Computing Fisher Market Equilibria with Spending Constraint Utilities. PhD thesis, New York University, 2016.
[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.
[Yeo08] Yinhuo Ye. A path to the arrow-debreu competitive market equilibrium. Mathematical Programming, 111(1-2):315–348, 2008.
[YN76] David B Yudin and Arkadii S Nemirovski. Evaluation of the information complexity of mathematical programming problems. Ekonomika i Matematicheskie Metody, 12:128–142, 1976.