Solving Tall Dense Linear Programs in Nearly Linear Time

Jan van den Brand
KTH Royal Institute of Technology, Sweden
janvdb@kth.se

Yin Tat Lee
University of Washington and MSR Redmond, U.S.A.
yintat@uw.edu

Aaron Sidford
Stanford University, U.S.A.
sidford@stanford.edu

Zhao Song
Princeton University and
Institute for Advanced Study, U.S.A.
zhaos@ias.edu

ABSTRACT

In this paper we provide an $\widetilde{O}(nd + d^3)$ time randomized algorithm for solving linear programs with d variables and n constraints with high probability. To obtain this result we provide a robust, primal-dual $\widetilde{O}(\sqrt{d})$-iteration interior point method inspired by the methods of Lee and Sidford (2014, 2019) and show how to efficiently implement this method using new data-structures based on heavy-hitters, the Johnson–Lindenstrauss lemma, and inverse maintenance. Interestingly, we obtain this running time without using fast matrix multiplication and consequently, barring a major advance in linear system solving, our running time is near optimal for solving dense linear programs among algorithms that do not use fast matrix multiplication.

CCS CONCEPTS

• **Theory of computation → Linear programming**.

KEYWORDS

linear program, interior point method, nearly linear time

ACM Reference Format:
Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. 2020. Solving Tall Dense Linear Programs in Nearly Linear Time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357713.3384309

The full version of this paper is available as [65] at https://arxiv.org/pdf/2002.02304.pdf.

STOC '20, June 22–26, 2020, Chicago, IL, USA

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384309

1 INTRODUCTION

Given $\mathbf{A} \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$, and $c \in \mathbb{R}^n$ solving a linear program (P) and its dual (D):

$$(P) = \min_{x \in \mathbb{R}^n_{\geq 0} : \mathbf{A}^\top x = b} c^\top x \text{ and } (D) = \max_{y \in \mathbb{R}^d : \mathbf{A}y \geq c} b^\top y \,. \tag{1}$$

is one of the most fundamental and well-studied problems in computer science and optimization. Developing faster algorithms for (1) has been the subject of decades of extensive research and the pursuit of faster linear programming methods has lead to numerous algorithmic advances and the advent of fundamental optimization techniques, e.g. simplex methods [18], ellipsoid methods [30], and interior-point methods (IPMs) [29].

The current fastest algorithms for solving (1) are the IPMs of Lee and Sidford [34] and Cohen, Lee, and Song [12]. These results build on a long line of work on IPMs [3, 29, 47, 52, 61, 63, 64], fast matrix multiplication [14, 15, 21, 22, 58–60, 66], and linear system solvers [9, 11, 39, 42, 44]. The first, [34] combined an $\widetilde{\mathrm{O}}(\sqrt{d})$ iteration IPM from [33, 35] with new techniques for *inverse maintenance,* i.e. maintaining an approximate inverse of a slowly changing matrix, to solve (1) in time $\widetilde{\mathrm{O}}(\mathrm{nnz}(\mathbf{A})\sqrt{d} + d^{2.5})$ with high probability. For sufficiently large values of $\mathrm{nnz}(\mathbf{A})$ or n, this is the fastest known running time for solving linear programming up to polylogarithmic factors.

The second, [12], developed a stable and robust version of the IPM of [52] (using techniques from [33, 35]) and combined it with novel randomization, data structure, and rectangular matrix multiplication [22] techniques to solve (1) in time $\widetilde{\mathrm{O}}(\max\{n, d\}^\omega)$ with high probability where $\omega < 2.373$ is the current best known matrix multiplication constant [21, 66]. When $n = \widetilde{\Theta}(d)$, this running time matches that of the best known linear system solvers for solving $n \times n$ linear systems and therefore is the best possible barring a major linear system solving advance.

Though, these results constitute substantial advances in algorithmic techniques for linear programming, the running times of [34] and [12] are incomparable and neither yield a nearly linear running time when n grows polynomially with d, i.e. when $d = n^\delta$ for any $\delta > 0$ neither [34] or [12] yields a nearly linear running time. Consequently, it has remained a fundamental open problem to determine whether or not it is possible to solve high-dimensional linear programs to high-precision in nearly linear time for any polynomial ratio of n, d, and $\mathrm{nnz}(\mathbf{A})$. Though achieving such a nearly linear runtime is known in the simpler setting of linear regression

[9–11, 39, 42, 44], achieving analogous results for solving such tall linear programs has been elusive.

In this paper we provide the first such nearly linear time algorithm for linear programming. We provide an algorithm which solves (1) with high probability in time $\widetilde{O}(nd + d^3)$. Whenever $\mathbf{A}$ is dense, i.e. $\text{nnz}(\mathbf{A}) = \Omega(nd)$, and sufficiently tall, i.e. $n = \Omega(d^2)$, this constitutes a nearly linear running time. In contrast to previous state-of-the-art IPMs for linear programming [12, 34] our algorithm doesn't need to use fast matrix multiplication and thereby matches the best known running time for regression on dense matrices which does not use fast matrix multiplication.

To achieve this result, we introduce several techniques that we believe are independent interest. First, we consider the IPM of [35] and develop an efficiently implementable robust primal-dual variant of it in the style of [5, 12, 37] which only requires solving $\widetilde{O}(\sqrt{d})$ linear systems. Prior to [33, 35], obtaining such an efficient $\widetilde{O}(\sqrt{d})$-iteration IPM was a major open problem in the theory of IPMs. We believe our primal dual method and its analysis is simpler than that of [35]; by developing a primal-dual method we eliminate the need for explicit ℓ_p-Lewis weight computation for $p \neq 2$ as in [35] and instead work with the simpler ℓ_2-Lewis weights or leverage scores. Further, we show that this primal dual method is highly stable and can be implemented efficiently given only multiplicative estimates of the variables, Hessians, and leverage scores.

With this IPM in hand, the problem of achieving our desired running time reduces to implementing this IPM efficiently. This problem is that of maintaining multiplicative approximations to vectors (the current iterates), leverage scores (a measure of importance of the rows under local rescaling), and the inverse of matrix (the system one needs to solve to take a step of the IPM) under small perturbations. While variants of vector maintenance have been considered recently [5, 12, 37] and inverse maintenance is well-studied historically [5, 12, 29, 33–35, 37, 48, 49, 61], none of these methods can be immediately applied in our setting where we cannot afford to pay too much in terms of n each iteration.

Our second contribution is to show that these data-structure problems can be solved efficiently. A key technique we use to overcome these issues is heavy-hitters sketching. We show that it is possible to apply a heavy hitters sketch (in particular [26, 50]) to the iterates of the method such that we can efficiently find changes in the coordinates. This involves carefully sketching groups of updates and dynamically modifying the induced data-structure. These sketches only work against non-adaptive adversaries, and therefore care is need to ensure that the sketch is used only to save time and not affect the progression of the overall IPM in a way that break this non-adaptive assumption. To achieve this we use the sketches to propose short-lists of possible changes which we then filter to ensure that the output of the datastructure is deterministic (up to a low probability failure event). Coupling this technique with known Johnson-Lindestrauss sketches yields our leverage score maintenance data structure and adapting and simplifying previous inverse maintenance techniques yields our inverse maintenance data structure. We believe this technique of sketching the central path is powerful and may find further applications.

1.1 Our Results

The main result of this paper is the following theorem for solving (1). This algorithm's running time is nearly linear whenever the LP is sufficiently dense and tall, i.e. $\text{nnz}(\mathbf{A}) = \widetilde{\Omega}(nd)$ and $n = \widetilde{\Omega}(d^2)$. This is the first polynomial time algorithm with a nearly linear running time for high-dimensional instances (i.e. when d can grow polynomially with n). This algorithm does not use fast matrix multiplication (FMM) and consequently its running time matches the best known running time for checking whether there even exists x such that $\mathbf{A}^\top x = b$ for dense $\mathbf{A}$ without using FMM.

Theorem 1.1 (Main Result). *There is an algorithm (Algorithm 2) which given any linear program of the form (1) for non-degenerate*[1] $\mathbf{A} \in \mathbb{R}^{n \times d}$ *and* $\delta \in [0, 1]$ *computes a point* $x \in \mathbb{R}^n_{\geq 0}$ *such that*

$$c^\top x \leq \min_{\mathbf{A}^\top x = b, x \geq 0} c^\top x + \delta \cdot \|c\|_2 \cdot R \text{ and}$$

$$\|\mathbf{A}^\top x - b\|_2 \leq \delta \cdot \left(\|\mathbf{A}\|_F \cdot R + \|b\|_2 \right)$$

where R is the diameter of the polytope in ℓ_2 norm, i.e. $\|x\|_2 \leq R$ for all $x \in \mathbb{R}^n_{\geq 0}$ with $\mathbf{A}^\top x = b$. Further, the expected running time of the method is $O((nd + d^3) \log^{O(1)} n \log \frac{n}{\delta})$.

Remark. See [33, 52] on the discussion on converting such an approximate solution to an exact solution. For integral $\mathbf{A}, b, c$, it suffices to pick $\delta = 2^{-O(L)}$ to get an exact solution where $L = \log(1 + d_{\max} + \|c\|_\infty + \|b\|_\infty)$ is the bit complexity and $d_{\max}$ is the largest absolute value of the determinant of a square sub-matrix of $\mathbf{A}$. For many combinatorial problems $L = O(\log(n + \|b\|_\infty + \|c\|_\infty))$.

Beyond this result (Section 4) we believe our robust primal-dual $\widetilde{O}(\sqrt{d})$-iteration IPM and our data structures for maintaining multiplicative approximations to vectors , leverage scores , and the inverse of matrices are of independent interest.

1.2 Previous Work

Linear programming has been the subject of extensive research for decades and it is impossible to completely cover to this impressive line of work in this short introduction. Here we cover results particularly relevant to our approach. For more detailed coverage of prior-work see, e.g. [47, 68].

IPMs: The first proof of a polynomial time IPM was due to Karmarkar in [29]. After multiple running time improvements [29, 48, 49, 61] the current fastest IPMs are the aforementioned results of [35] and [12]. Beyond these results, excitingly the work of [12] was recently extended to obtain comparable running time improvements for solving arbitrary empirical risk minimization problems (ERM) [37] and was recently simplified and de-randomized by van den Brand [5]. These works consider variants of the vector maintenance problem and our work is inspired in part by them. Our IPM leverages the barrier from [35] in a new way, that enables the application of robustness techniques from [5, 12, 35] and new techniques for handling approximately feasible points.

Heavy Hitters and Sketching: Sketching is a well-studied problem with a broad range of applications. Johnson-Lindenstrauss

[1] We assume throughout that $\mathbf{A}$ is *non-degenerate* meaning it has full-column rank and no-zero rows. This assumption can be avoided by preprocessing $\mathbf{A}$ to check for zero rows and adding a tiny amount of noise to the matrix to make it full-column rank. There are other natural ways to remove this assumption, see e.g. [35].

sketches were used extensively in previous IPMs [35], but only for the restricted application of computing leverage scores. Further sketching techniques have been used for the purpose of dimension reduction and sampling in other optimization contexts, e.g. solving linear systems [9–11, 39, 42, 44] and certain forms of ℓ_p-regression [13]. Our methods make use of ℓ_2-heavy hitters sketches [8, 16, 17, 26, 31, 45, 50], in particular the ℓ_2-sketches of [26, 50], to decrease iteration costs. We are unaware of these sketches being used previously to obtain provable improvements for solving offline (as opposed to online, streaming, or dynamic optimization problems) variants of linear programming previously. We believe this is due in part to the difficulty of using these methods with non-oblivious adversaries and consequently we hope the techniques we use to overcome these issues may be of further use. Also, note that the definition of ℓ_2-heavy hitters problem that we use is equivalent to ℓ_∞/ℓ_2-sparse recovery problem. Though the ℓ_2/ℓ_2-sparse recovery [7, 19, 23, 25, 43, 51, 55] is a more standard task in compressed sensing, we are unaware of how to efficiently use its guarantees for our applications.

Leverage Scores and Lewis Weights: Leverage scores [11, 39, 40, 57] (and more broadly Lewis weights [4, 13, 35, 38]) are fundamental notions of importance of rows of a matrix with numerous applications. In this paper we introduce a natural online problem for maintaining multiplicative approximations to leverage scores and we show how to solve this problem efficiently. Though we are unaware of this problem being studied previously, we know that in the special case of leverage scores induced by graph problems, which are known as effective resistances, there are dynamic algorithms for maintaining them, e.g [20]. However, these algorithms seem tailored to graph structure and it is unclear how to apply them in our setting. Further, there are streaming algorithms for variants of this problem [2, 27, 28], however their running time is too large for our purposes.

Inverse Maintenance: This problem has been studied extensively and are a key component in obtaining efficient linear programming running times with previous IPMs [5, 12, 29, 32–35, 37, 48, 49, 54, 55, 61] and other optimization methods [1, 36]. Outside the area of optimization, this problem is also known as Dynamic Matrix Inverse [6, 53]. Our method for solving inverse maintenance is closely related to these results with modifications needed to fully take advantage of our leverage score maintenance, obtain poly(d) (as opposed to poly(n) runtimes), ensure that randomness used to make the algorithm succeed doesn't affect the input to the data structure, and ultimately produce solvers that are correct in expectation.

2 OVERVIEW OF APPROACH

The proof of Theorem 1.1 is split into two steps. In the full version [65] we provide a new robust, primal-dual, $\widetilde{O}(\sqrt{d})$ iteration IPM inspired by the LS-barriers of [33, 35] and central path robustness techniques of [5, 12, 33, 35, 37]. Then we develop new data structures based on heavy hitters, the Johnson-Lindenstrauss lemma, and inverse maintenance. These data structure then allow us to efficiently implement this new IPM.

Here we give an outline for the new IPM and how to implement this IPM efficiently using the new data structures. Section 4 shows how to use these result to obtain the fast linear program solver.

2.1 A Robust Primal-Dual $\widetilde{O}(\sqrt{d})$ IPM

Here we provide an overview of our approach to deriving our robust, primal-dual, $\widetilde{O}(\sqrt{d})$ iteration IPM. We assume throughout this section (and the bulk of the paper) that $\mathbf{A}$ is *non-degenerate*, meaning it is full-column rank and no non-zero rows. By standard techniques, in Section 4.5 we efficiently reduce solving (1) in general to solving an instance of where this assumption holds.

We first give a quick introduction to primal-dual path IPMs, review the central path from [35] and from it derive the central path that our method is based on, and explain our new IPM.

Primal-Dual Path Following IPMs: Our algorithm for solving the linear programs given by (1) is rooted in classic primal-dual path-following IPMs. Primal-dual IPMs, maintain a *primal feasible point*, $x \in \mathbb{R}^n_{\geq 0}$ with $\mathbf{A}^\top x = b$, a *dual feasible point* $y \in \mathbb{R}^d$ with $s = \mathbf{A}y - c \geq 0$, and attempt to decrease the *duality gap*

$$\text{gap}(x, y) \stackrel{\text{def}}{=} c^\top x - b^\top y = (\mathbf{A}y + s)^\top x - (\mathbf{A}^\top x)^\top y = s^\top x\,.$$

Note that $\text{gap}(x, y)$ upper bounds the error of x and y, i.e. $\text{gap}(x, y) \leq \epsilon$ implies that x and y are each optimal up to an additive ϵ in objective function, and therefore to solve a linear program it suffices to decrease the duality gap for primal and dual feasible points.

Primal-dual path-following IPMs carefully trade-off decreasing the duality gap (which corresponds to objective function progress) with staying away from the inequality constraints, i.e. $x \geq 0$ and $s \geq 0$ (in order to ensure it is easier to make progress). Formally, they consider a (*weighted*) *central path* defined as the unique set of $(x_\mu, y_\mu, s_\mu) \in \mathbb{R}^n_{\geq 0} \times \mathbb{R}^d \times \mathbb{R}^n_{\geq 0}$ for $\mu > 0$ that satisfy

$$\begin{aligned} \mathbf{X}_\mu \mathbf{S}_\mu 1 &= \mu \cdot \tau_{\text{weight}}(x_\mu, s_\mu), \\ \mathbf{A}^\top x_\mu &= b, \\ \mathbf{A}y_\mu + s_\mu &= c, \end{aligned} \tag{2}$$

where $\mathbf{X}_\mu \stackrel{\text{def}}{=} \mathbf{Diag}(x_\mu)$, $\mathbf{S}_\mu \stackrel{\text{def}}{=} \mathbf{Diag}(s_\mu)$, and $\tau_{\text{weight}} : \mathbb{R}^n_{\geq 0} \times \mathbb{R}^n_{\geq 0} \to \mathbb{R}$ is a *weight function*. These methods maintain primal and dual feasible points and take Newton steps on the above non-linear inequalities to maintain feasible points that are more central, i.e. have (2) closer to holding, for decreasing μ. Since many properties of the central path can be defined in terms of just x_μ and s_μ we often describe methods using only these quantities and we adopt the following notation.

Definition 2.1 (Feasible Point). We say that $(x, s) \in \mathbb{R}^n_{\geq 0} \times \mathbb{R}^n_{\geq 0}$ is a *feasible point* if there exists $y \in \mathbb{R}^d$ with $\mathbf{A}^\top x = b$ and $\mathbf{A}y + s = c$.

Now, perhaps the most widely-used and simple weight function is $\tau_{\text{weight}}(x, s) \leftarrow \tau_{\text{std}}(x, s) \stackrel{\text{def}}{=} 1$, i.e. the all-ones vector. Central path points for this weight function are the solution to the following

$$x_\mu = \operatorname*{argmin}_{x \in \mathbb{R}^n_{\geq 0} : \mathbf{A}^\top x = b} c^\top x - \mu \sum_{i \in [n]} \log(x_i) \text{ and}$$

$$(y_\mu, s_\mu) = \operatorname*{argmax}_{(y,s) \in \mathbb{R}^d \times \mathbb{R}^n_{\geq 0} : \mathbf{A}^\top y + s = c} b^\top y - \mu \sum_{i \in [n]} \log(s_i)\,,$$

i.e. optimization problems trading off the objective function with logarithmic barriers on the inequality constraints. There are numerous methods for following this central path [24, 41, 47, 52, 69]. By starting with nearly-central feasible points for large μ and iteratively finding nearly-central feasible points for small μ, they

can compute ϵ-approximate solutions to (1) in $\widetilde{O}(\sqrt{n})$-iterations. For these methods, each iteration (or Newton step) consists of nearly-linear time computation and solving one linear system in the matrix $\mathbf{A}^\top \mathbf{X}\mathbf{S}^{-1}\mathbf{A} \in \mathbb{R}^{d\times d}$ for $\mathbf{X} = \mathbf{Diag}(x) \in \mathbb{R}^{n\times n}$ and $\mathbf{S} = \mathbf{Diag}(s) \in \mathbb{R}^{n\times n}$.

The first such $\widetilde{O}(\sqrt{n})$-iteration IPM with this iteration cost was established by Renegar in 1988 [52] and no faster-converging method with the same iteration cost was developed until the work of Lee and Sidford in 2014 [33, 35]. It had been known since seminal work of Nesterov and Nemirovski in 1994 [47] that there is a weight function that yields a $\widetilde{O}(\sqrt{d})$-iteration primal-dual path-following IPM, however obtaining any $\widetilde{O}(\sqrt{d})$-iteration IPM that can be implemented with iterations nearly as efficient as those of [52] was a long-standing open problem.

The Lewis Weight Barrier and Beyond The work of [33, 35] addressed this open problem in IPM theory and provided an efficient $\widetilde{O}(\sqrt{d})$-iteration IPM by providing new weight functions and new tools for following the central path they induce. In particular, [35] introduced the *Lewis weight barrier* which induces the central path in which for some $p > 0$

$$\tau_{\text{weight}}(x,s) \leftarrow \tau_{\text{ls}}(x,s) \stackrel{\text{def}}{=} \sigma^{(p)}(\mathbf{S}^{-1}\mathbf{A})$$

where for any $\mathbf{B} \in \mathbb{R}^{n\times d}$, $\sigma^{(p)}(\mathbf{B})$ are the *ℓ_p-Lewis weights of the rows of* $\mathbf{B}$ [38], a fundamental and natural measure of importance of rows with respect to the ℓ_p-norm [4, 13, 35]. In the case of non-degenerate $\mathbf{B}$, they are defined recursively as the vector $w \in \mathbb{R}^n_{>0}$ which satisfies

$$w = \text{diag}(\mathbf{W}^{(1/2)-(1/p)}\mathbf{B}(\mathbf{B}^\top \mathbf{W}^{1-(2/p)}\mathbf{B})^{-1}\mathbf{B}^\top \mathbf{W}^{(1/2)-(1/p)}),$$

where $\mathbf{W} = \mathbf{Diag}(w)$. In the special case when $p = 2$,

$$w = \sigma(\mathbf{B}) \stackrel{\text{def}}{=} \text{diag}(\mathbf{B}(\mathbf{B}^\top \mathbf{B})^\dagger \mathbf{B}^\top)$$

is known as the *leverage scores* of the rows of $\mathbf{A}$ and is a fundamental object for dimension reduction and solving linear systems [11, 39, 40, 46, 56, 57, 67].

The work of [35] formally showed that there is an $\widetilde{O}(\sqrt{d})$-iteration primal-dual IPM which uses $\tau_{\text{weight}}(x,s) \leftarrow \tau_{\text{ls}}(x,s)$ when $p = \Omega(\log n)$. This choice of p is motivated by drew geometric connections between Lewis weights and ellipsoidal approximations of polytopes [35]. Further, [35] showed how to modify this IPM to have iterations of comparable cost to the methods which use $\tau_{\text{std}}(x,s)$. This was achieved by leveraging and modifying efficient algorithms for approximately computing Lewis weights and leverage scores and developing techniques for dealing with the noise such approximate computation induces.

To obtain the results of this paper we further simplify [35] and provide more robust methods for following related central paths. Our first observation is that the central path induced by τ_{ls} can be re-written more concisely in terms of only leverage scores. Note that (2) and the definition of Lewis weights imply that there is w_μ with

$$\mathbf{X}_\mu \mathbf{S}_\mu 1 = \mu \cdot w_\mu \text{ and } w_\mu = \sigma(\mathbf{W}_\mu^{(1/2)-(1/p)}\mathbf{S}_\mu^{-1}\mathbf{A}).$$

for $\mathbf{W}_\mu \stackrel{\text{def}}{=} \mathbf{Diag}(w_\mu)$. Substituting the first equation into the second, gives more compactly that for $\alpha = 1/p$

$$\mathbf{X}_\mu \mathbf{S}_\mu 1 = \mu \cdot \mathbf{Diag}(\sigma(\mathbf{S}_\mu^{-1/2-\alpha}\mathbf{X}_\mu^{1/2-\alpha}\mathbf{A})).$$

Consequently, rather than defining centrality in terms of Lewis weights, we would get the same central path as the one induced by $\tau_{\text{ls}}(x,s)$ by letting $\tau_{\text{weight}}(x,s) \leftarrow \sigma(\mathbf{S}_\mu^{1/2-\alpha}\mathbf{X}_\mu^{1/2-\alpha}\mathbf{A})$, i.e. defining it in terms of leverage scores. In other words, the optimality of the central path conditions forces $\mathbf{X}_\mu \mathbf{S}_\mu 1$ to be a type of Lewis weight if we carefully define centrality in terms of leverage scores. Though $\tau_{\text{weight}}(x,s) \leftarrow \sigma(\mathbf{S}^{1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A})$ induces the same central path as $\tau_{\text{ls}}(x,s)$, these weight functions can be different outside the central path and thereby lead to slightly different algorithms if only approximate centrality is preserved. Further, with the current state-of-the-art theory, leverage scores are simpler to compute and approximate, as Lewis weight computation is often reduced to leverage score computation [13, 35].

Formally, in this paper we consider the following regularized-variant of this centrality measure:

Definition 2.2 (Weight Function). Throughout this paper we let $\alpha \stackrel{\text{def}}{=} 1/(4\log(4n/d))$ and for all $x, s \in \mathbb{R}^n_{>0}$ let

$$\tau_{\text{reg}}(x,s) \stackrel{\text{def}}{=} \sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$$

where $\mathbf{X} = \mathbf{Diag}(x)$ and $\mathbf{S} = \mathbf{Diag}(s)$.

This centrality measure is the same as the Lewis weight barrier except that we add a multiple of the all-ones vector, $\frac{d}{n}1$, to simplify our analysis. Further, this allows us to pick $\alpha \stackrel{\text{def}}{=} 1/(4\log(4n/d))$ as opposed to $\alpha = 1/\Omega(\log n)$ due to the extra stability it provides. Since we use this weight function throughout the paper we overload notation and let $\tau(x,s) \stackrel{\text{def}}{=} \tau_{\text{reg}}(x,s)$ and $\tau(\mathbf{B}) \stackrel{\text{def}}{=} \sigma(\mathbf{B}) + \frac{d}{n}1$.

Our Robust Primal-Dual Method: We obtain our results by proving that there is an efficient primal-dual path-following IPM based on $\tau_{\text{reg}}(x,s)$. We believe that our analysis is slightly simpler than [35] due to its specification in terms of leverage scores, rather than the more general Lewis weights, but remark that the core ingredients of its analysis are similar. Formally, we provide Newton-method type steps that allow us to control centrality with respect to this measure and increase μ fast enough that this yields an $\widetilde{O}(\sqrt{d})$-iteration method.

Beyond providing a simplified $\widetilde{O}(\sqrt{d})$-iteration IPM, we leverage this analysis to provide a robust method. Critical to the development of recent IPMs is that it is possible to design efficient primal-dual $\widetilde{O}(\sqrt{n})$-iteration IPMs that take steps using only crude, multiplicative approximations to x and s [5, 12, 37]. These papers consider the standard central path but measure centrality using potential functions introduced in [33, 35]. This robustness allows these papers to efficiently implement steps by only needing to change smaller amounts of coordinates.

Similarly, we show how to apply these approximate centrality measurement techniques to the central path induced by $\tau_{\text{reg}}(x,s)$. We show that it suffices to maintain multiplicative approximations to the current iterate ($x, s \in \mathbb{R}^n_{\geq 0}$), the regularized leverage scores ($\sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$), and the inverse of the local Hessian ($(\mathbf{A}^\top \mathbf{S}^{-1}\mathbf{X}\mathbf{A})^{-1}$) to maintain approximate centrality with respect to $\tau_{\text{reg}}(x,s)$. Interestingly, to do this we slightly modify the type of steps we take in our IPM. Rather than taking standard Newton steps, we slightly change the steps sizes on x and s to account for the effect of $\tau_{\text{reg}}(x,s)$. Further, approximation of the Hessian causes

the x iterates to be infeasible, but in the full version we discuss how to modify the steps to control this infeasibility and still prove the desired theorem.

The main guarantees of this new IPM are given by Theorem 4.1 (Section 4.1) and are proven in the full version. This theorem formalizes the above discussion and quantifies how much the iterates, leverage scores, and Hessian can change in each iteration. These bounds are key to obtaining an efficient method that can maintain multiplicative approximations to these quantities.

2.2 Heavy Hitters, Congestion Detection, and Sketching the Central Path

It has long been known that the changes to the central path are essentially sparse or low-rank on average. In Renegar's IPM [52], the multiplicative change in x and s per iteration are bounded in ℓ_2. Consequently, the matrices for which linear systems are solved in Renegar's method do not change too quickly. In fact, this phenomenon holds for IPMs more broadly, and from the earliest work on polynomial time IPMs [29], to the most recent fastest methods [5, 33–35, 37], and varied work in between [48, 49, 62] this bounded change in IPM iterates has been leveraged to obtain faster linear programming algorithms.

Our IPM also enjoys a variety of stability properties on its iterates. We show that the multiplicative change in the iterates are bounded in a norm induced by $\tau_{\text{reg}}(x,s)$ and therefore are also bounded in ℓ_2. This is known to imply that changes to $\mathbf{A}^\top\mathbf{S}^{-1}\mathbf{X}\mathbf{A} \in \mathbb{R}^{d\times d}$ can be bounded over the course of the algorithm and we further show that this implies that the changes in the weight function, $\tau_{\text{reg}}(x,s)$, can also be bounded. These facts, combined with the robustness properties of our IPM imply that to obtain an efficient linear programming algorithm it suffices to maintain multiplicative approximations to the following three quantities (1) the vectors $x, s \in \mathbb{R}^n_{\geq 0}$, (2) the regularized leverage scores $\tau_{\text{reg}}(x,s)$, and (3) the Hessian inverse $(\mathbf{A}^\top\mathbf{S}^{-1}\mathbf{X}\mathbf{A})^{-1} \in \mathbb{R}^{d\times d}$ under bounds on how quickly these quantities change.

We treat each of these problems as a self contained data-structure problem, the first we call the *vector maintenance problem*, the second we call the *leverage score maintenance problem*, and the third has been previously studied (albeit different variants) and is called the *inverse maintenance problem*. For each problem we build efficient solutions by combining techniques form the sketching literature (e.g. heavy hitters sketches and Johnson-Lindenstrauss sketches) and careful potential functions and tools for dealing with sparse and low rank approximations. (Further work is also needed to maintain the gradient of a potential used to measure proximity to the central path and this is discussed in the full version.)

In the remainder of this overview, we briefly survey how we solve each of these problems. A common issue that needs to addressed in solving each problem is that of hiding randomness and dealing with adversarial input. Each data-structure uses sampling and sketching techniques to improve running times. While these techniques are powerful and succeed with high probability, they only work against an *oblivious adversary*, i.e. one which provides input that does not depend on the randomness of the data structure. However, the output of our data-structures are used to take steps along the central path and provide the next input, so care needs to be taken to argue that the output of the data structure, as used by the method, doesn't somehow leak information about the randomness of the sketches and samples into the next input. A key contribution of our work is showing how to overcome this issue in each case.

Vector Maintenance: In the vector maintenance problem, we receive two online sequences of vectors $h^{(1)}, h^{(2)}, \ldots \in \mathbb{R}^d$ and $g^{(1)}, g^{(2)}, \ldots \in \mathbb{R}^n$ and must maintain the sum $y^{(t)} \stackrel{\text{def}}{=} \sum_{i\in[t]} \mathbf{G}^{(i)}\mathbf{A}h^{(i)}$ for a fixed matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$. The naive way of solving this would just compute $\mathbf{G}^{(t)}\mathbf{A}h^{(t)}$ in iteration t and add it to the previous result. Unfortunately, this takes $O(\text{nnz}(A))$ time per iteration, which is too slow for our purposes. Luckily we do not have to maintain this sum exactly. Motivated by the robustness of our IPM, it is enough to maintain a multiplicative approximation $\widetilde{y}^{(t)} \approx_\epsilon y^{(t)}$ of the sum. An exact definition of this problem, together with our upper bounds, can be found in Section 4.2 and the formal proof of these results can be found in the full version.

We now outline how vector maintenance problem can be solved. For simplicity assume we already have an accurate approximation $\widetilde{y}^{(t-1)} \approx_{\epsilon/2} y^{(t-1)}$. Then we only have to change the entries i of $\widetilde{y}^{(t-1)}$ where $y_i^{(t)} \not\approx_{\epsilon/2} y_i^{(t-1)}$, because for all other i we have $\widetilde{y}_i^{(t-1)} \approx_\epsilon y_i^{(t)}$. This means we must detect the large entries of $y^{(t)} - y^{(t-1)} = \mathbf{G}^{(t)}\mathbf{A}h^{(t)}$, which can be done via heavy hitter techniques. From past research on heavy hitters, we know one can construct a small, sparse, random matrix $\Phi \in \mathbb{R}^{k\times n}$ with $k \ll n$, such that known the much smaller vector $x \in \mathbb{R}^k$ with $x = \Phi y$ allows for a quick reconstruction of the large entries of y. Thus for our task, we maintain the product $\Phi\mathbf{G}^{(t)}\mathbf{A}$, which can be done quickly if $g^{(t)}$ does not change in too many entries compared to $g^{(t-1)}$. Then we can reconstruct the large entries of $\mathbf{G}^{(t)}\mathbf{A}h^{(t)}$ by computing $(\Phi\mathbf{G}^{(t)}\mathbf{A})h^{(t)}$. Note that this product can be computed much faster than $\mathbf{G}^{(t)}\mathbf{A}h^{(t)}$, because $\Phi\mathbf{G}^{(t)}\mathbf{A}$ is a $k \times d$ matrix and $k \ll n$.

One issue we must overcome, is that the output of our data-structure must not leak any information about Φ. This matrix is randomly constructed and the large entries of y can only be reconstructed from $x = \Phi y$, if the vector y is independent from the randomness in Φ. Thus if the output of the data-structure depends on Φ and the next future input $h^{(t)}$ depends on the previous output, then the required independence can no longer be guaranteed. We overcome this problem by computing any entry $y_i^{(t)}$ exactly, whenever the heavy hitter techniques detect a large change in said entry. As we now know the value of said entry exactly, we can compare it to the previous result and verify, if the entry did indeed change by some large amount. This allows us to define the output of our data-structure in a deterministic way, e.g. maintain $\widetilde{y}_i^{(t)} \approx_\epsilon y_i^{(t)}$ by setting $\widetilde{y}_i^{(t)} = y_i^{(t)}$ whenever $\widetilde{y}_i^{(t-1)} \not\approx_\epsilon y_i^{(t)}$. (The exact deterministic definition we use is slightly more complex, but this is the high-level idea.)

So far we only explained how we can detect large changes in $y^{(t)}$ that occur within a single iteration. However, it could be that some entry changes slowly over several iterations. It is easy to extend our heavy hitter technique to also detect these slower changes. The idea is that we not just detect changes within one iteration (i.e. $y_i^{(t)} \not\approx_{\epsilon/2} y_i^{(t-1)}$) but also changes within any power of two (i.e. $y_i^{(t)} \not\approx_{\epsilon/2} y_i^{(t-2^i)}$ for $i = 1, \ldots, \log t$). To make sure that this does not

become too slow, we only check every 2^i iterations if there was a large change within the past 2^i iterations. One can prove that this is enough to also detect slowly changing entries.

Leverage Score Maintenance: In the leverage score maintenance problem, we must maintain an approximation of the regularized leverage scores of a matrix of the form $\mathbf{G}^{1/2}\mathbf{A} \in \mathbb{R}^{n\times d}$, for a slowly changing diagonal matrix $\mathbf{G} \in \mathbb{R}^{n\times n}$. That means we are interested in a data-structure that maintains a vector $\widetilde{\tau} \in \mathbb{R}^n$ with $\widetilde{\tau}_i \approx_\epsilon \tau_i := (\mathbf{G}^{1/2}\mathbf{A}(\mathbf{A}^\top\mathbf{G}\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{G}^{1/2})_{i,i} + d/n$. The high-level idea for how to solve this task is identical to the previously outlined vector-maintenance: we try to detect for which i the value τ_i changed significantly, and then update $\widetilde{\tau}_i$ so that the vector stays a valid approximation. We show that detecting these indices i can be reduced to the previous vector maintenance. Here we simply outline this reduction, a formal description of our result can be found in Section 4.3 and its proof and analysis is in the full version.

Consider the case where we change from $\mathbf{G}$ to some $\mathbf{G}'$. We want to detect indices i where the ith leverage score changed significantly. Define $\mathbf{M} \stackrel{\text{def}}{=} (\mathbf{A}^\top\mathbf{G}\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{G}^{1/2} - (\mathbf{A}^\top\mathbf{G}'\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{G}'^{1/2} \in \mathbb{R}^{d\times n}$ and note that

$$(\mathbf{G}^{1/2}\mathbf{A}(\mathbf{A}^\top\mathbf{G}\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{G}^{1/2})_{i,i} = \|e_i^\top\mathbf{G}^{1/2}\mathbf{A}(\mathbf{A}^\top\mathbf{G}\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{G}^{1/2}\|_2^2,$$

so a large change in the ith leverage score, when changing $\mathbf{G} \in \mathbb{R}^{n\times n}$ to $\mathbf{G}' \in \mathbb{R}^{n\times n}$, results in a large norm of $e_i^\top\mathbf{G}^{1/2}\mathbf{A}\mathbf{M}$, if $\mathbf{G}_{i,i}$ and $\mathbf{G}'_{i,i}$ are roughly the same. (As $\mathbf{G}$ is slowly changing there are not too many i where $\mathbf{G}_{i,i}$ and $\mathbf{G}'_{i,i}$ significantly. So we can just compute the ith leverage score exactly for these entries to check if the ith score changed significantly.) By multiplying this term with a Johnson-Lindenstrauss matrix $\mathbf{J}$ the task of detecting a large leverage score change, becomes the task of detecting rows of $\mathbf{G}^{1/2}\mathbf{A}\mathbf{M}\mathbf{J}$ for which the row-norm changed significantly. Given that $\mathbf{J}$ only needs some $O(\log n)$ columns to yield a good approximation of these norms, we know that any large change in the ith leverage score, must result in some index j where $|(\mathbf{G}^{1/2}\mathbf{A}\mathbf{M}\mathbf{J})_{i,j}|$ must be large. Detecting these large entries can be done in the same way as in the vector-maintenance problem by considering each column of $\mathbf{M}\mathbf{J}$ as a vector.

To make sure that no information about the random matrix $\mathbf{J}$ is leaked, we use the same technique previously outlined in the vector-maintenance paragraph. That is, after detecting a set $I \subset [n]$ of indices i for which the leverage score might have changed significantly, we compute the ith leverage score to verify the large change and set $\widetilde{\tau}_i$ to be this computed leverage score, if the change was large enough. Unlike the vector case however, the ith leverage score is not computed in a deterministic way (as this would be prohibitively expensive). Instead we use another random Johanson-Lindenstrauss matrix $\mathbf{J}'$, so the output $\widetilde{\tau}$ is actually defined w.r.t the input and this new matrix $\mathbf{J}'$. By using a fresh independent Johnson-Lindenstrauss $\mathbf{J}'$ to verify changes to leverage scores, this data structure works against an adaptive adversary.

Inverse Maintenance: In the inverse maintenance problem, we maintain a spectral sparsifier of $\mathbf{A}^\top\mathbf{W}\mathbf{A} \in \mathbb{R}^{d\times d}$ and its inverse, for a slowly changing diagonal matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$. Using the leverage score data-structure, we can assume an approximation to the leverage scores of $\mathbf{A}^\top\mathbf{W}\mathbf{A}$ is given. Hence, we can sample $\widetilde{O}(d)$ many rows of $\mathbf{A}$ to form a spectral sparsifier. This allows us to get a spectral sparsifier and its inverse in $\widetilde{O}(d^\omega)$. To speed up the runtime, we follow the idea in [34], which resamples the row only if the leverage score changed too much. This makes sure the sampled matrix is slowly changing in ℓ_0 sense and hence we can try to apply the lazy low-rank update idea in [12] to update the inverse in $\widetilde{O}(d^2)$ time. Unfortunately, this algorithm works only against oblivious adversary and the sampled matrix is changing too fast in ℓ_2 sense, which is required for the leverage score maintenance.

Fortunately, we note that we do not need a sparsifier that satisfies both conditions at all time. Therefore, our final data structure has two ways to output the inverse of the sparsifier. The sparsifier that works only against oblivious adversary is used in implementing the Newton steps and the sparsifier that is slowly changing is used to compute the sketch $\mathbf{M}\mathbf{J}$ used in the leverage score maintenance problem mentioned above.

For the Newton steps, we only need to make sure the sparsifier does not leak the randomness since the input $\mathbf{W}$ depends on the Newton step. Since we only need to solve linear systems of the form $\mathbf{A}^\top\mathbf{W}\mathbf{A}x = b \in \mathbb{R}^d$, we can handle this problem by adding an appropriate noise to the output x. This makes sure the randomness we use in this data structure does not leak when the output x is used. This idea is also used [34], but extra care is needed to remove the nnz($\mathbf{A}$) per step cost in their algorithm.

For computing the sketch $\mathbf{M}\mathbf{J}$, we do not need to worry about leakage of randomness, but only need to make sure it is slowly changing in ℓ_2 sense. Instead of using [12] as a black-box, we show how to combing the idea of resampling in [34] and the low-rank update in [12]. This gives us an alternate smoother scheme that satisfies the requirement for slowly changing.

3 PRELIMINARIES

Here we discuss varied notation we use throughout the paper. We adopt similar notation to [33] and some of the explanations here are copied directly form this work.

Matrices: We call a matrix $\mathbf{A}$ *non-degenerate* if it has full column-rank and no zero rows. We call symmetric matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ positive semidefinite (PSD) if $x^\top\mathbf{B}x \geq 0$ for all $x \in \mathbb{R}^n$ and positive definite (PD) if $x^\top\mathbf{B}x > 0$ for all $x \in \mathbb{R}^n$.

Matrix Operations: For symmetric matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ we write $\mathbf{A} \preceq \mathbf{B}$ to indicate that $x^\top\mathbf{A}x \leq x^\top\mathbf{B}x$ for all $x \in \mathbb{R}^n$ and define $\prec$, $\preceq$, and $\succeq$ analogously. For $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times m}$, we let $\mathbf{A} \circ \mathbf{B}$ denote the Schur product, i.e. $[\mathbf{A} \circ \mathbf{B}]_{i,j} \stackrel{\text{def}}{=} \mathbf{A}_{i,j} \cdot \mathbf{B}_{i,j}$ for all $i \in [n]$ and $j \in [m]$. We use nnz($\mathbf{A}$) to denote the number of nonzero entries in $\mathbf{A}$.

Diagonals: For $\mathbf{A} \in \mathbb{R}^{n\times n}$ we define diag($\mathbf{A}$) $\in \mathbb{R}^n$ with diag($\mathbf{A}$)$_i$ = $\mathbf{A}_{ii}$ for all $i \in [n]$ and for $x \in \mathbb{R}^n$ we define $\mathbf{Diag}(x) \in \mathbb{R}^{n\times n}$ as the diagonal matrix with diag($\mathbf{Diag}(x)(x)$) = x. We often use upper case to denote a vectors associated diagonal matrix and in particular let $\mathbf{X} \stackrel{\text{def}}{=} \mathbf{Diag}(x)$, $\mathbf{S} \stackrel{\text{def}}{=} \mathbf{Diag}(s)$, $\mathbf{W} \stackrel{\text{def}}{=} \mathbf{Diag}(w)$, $\mathbf{T} \stackrel{\text{def}}{=} \mathbf{Diag}(\tau)$, $\overline{\mathbf{X}} \stackrel{\text{def}}{=} \mathbf{Diag}(\overline{x})$, $\overline{\mathbf{S}} \stackrel{\text{def}}{=} \mathbf{Diag}(\overline{s})$, $\overline{\mathbf{W}} \stackrel{\text{def}}{=} \mathbf{Diag}(\overline{w})$, $\mathbf{X}_t \stackrel{\text{def}}{=} \mathbf{Diag}(x_t)$, $\mathbf{S}_t$ = $\mathbf{Diag}(s_t)$, $\mathbf{W}_t$ = $\mathbf{Diag}(w_t)$, and $\mathbf{T}_t \stackrel{\text{def}}{=} \mathbf{Diag}(\tau_t)$.

Fundamental Matrices: For any non-degenerate matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ we let $\mathbf{P}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{A}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$ denote the orthogonal projection matrix onto $\mathbf{A}$'s image. Further, we let $\sigma(\mathbf{A}) \stackrel{\text{def}}{=}$ diag($\mathbf{P}(\mathbf{A})$)

denote $\mathbf{A}$'s *leverage scores* and we let $\tau(\mathbf{A}) \stackrel{\text{def}}{=} \sigma(\mathbf{A}) + \frac{d}{n}\mathbf{1}$ denote its regularized leverage scores. Further, we define $\Sigma(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{Diag}(\sigma(\mathbf{A}))$, $\mathbf{T}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{Diag}(\tau(\mathbf{A}))$, $\mathbf{P}^{(2)}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{P}(\mathbf{A}) \circ \mathbf{P}(\mathbf{A})$ (where $\circ$ denotes entry-wise product), and $\mathbf{\Lambda}(\mathbf{A}) \stackrel{\text{def}}{=} \Sigma(\mathbf{A}) - \mathbf{P}^{(2)}(\mathbf{A})$.

Approximations: We use $x \approx_\epsilon y$ to denote that $\exp(-\epsilon)y \leq x \leq \exp(\epsilon)y$ and $\mathbf{A} \approx_\epsilon \mathbf{B}$ to denote that $\exp(-\epsilon)\mathbf{B} \preceq \mathbf{A} \preceq \exp(\epsilon)\mathbf{B}$.

Norms: For PD $\mathbf{A} \in \mathbb{R}^{n\times n}$ we let $\|\cdot\|_\mathbf{A}$ denote the norm where $\|x\|_\mathbf{A}^2 \stackrel{\text{def}}{=} x^\top \mathbf{A} x$ for all $x \in \mathbb{R}^n$. For positive $w \in \mathbb{R}^n_{>0}$ we let $\|\cdot\|_w$ denote the norm where $\|x\|_w^2 \stackrel{\text{def}}{=} \sum_{i\in[n]} w_i x_i^2$ for all $x \in \mathbb{R}^n$. For any norm $\|\cdot\|$ and matrix $\mathbf{M}$, its induced operator norm of $\mathbf{M}$ is defined by $\|\mathbf{M}\| = \sup_{\|x\|=1} \|\mathbf{M}x\|$.

Time and Probability: We use $\widetilde{\mathrm{O}}(\cdot)$ to hide factors polylogarithmic in n and d. We say an algorithm has a property "with high probability (w.h.p.) in n" if it holds with probability at least $1 - 1/O(\mathrm{poly}(n))$ for any polynomial by choice of the constants in the runtime of the algorithm.

Misc: We let $[z] \stackrel{\text{def}}{=} \{1, 2, .., z\}$. We let $1_n, 0_n \in \mathbb{R}^n$ denote the all-one and all-zero vectors, $\mathbf{0}_n, \mathbf{I}_n \in \mathbb{R}^{n\times n}$ denote the all zero and identity matrices, and drop subscripts when the dimensions are clear. We let 1_i denote the indicator vector for coordinate i, i.e. the i-th basis vector.

4 LINEAR PROGRAMMING ALGORITHM

Here we prove the main result of this paper that there is a $\widetilde{\mathrm{O}}(nd+d^3)$ time algorithm for solving linear programs. This theorem is restated below for convenience:

Theorem 1.1 (Main Result). *There is an algorithm (Algorithm 2) which given any linear program of the form (1) for non-degenerate*[2] $\mathbf{A} \in \mathbb{R}^{n\times d}$ *and* $\delta \in [0, 1]$ *computes a point* $x \in \mathbb{R}^n_{\geq 0}$ *such that*

$$c^\top x \leq \min_{\mathbf{A}^\top x = b, x\geq 0} c^\top x + \delta \cdot \|c\|_2 \cdot R \text{ and}$$

$$\|\mathbf{A}^\top x - b\|_2 \leq \delta \cdot \left(\|\mathbf{A}\|_F \cdot R + \|b\|_2\right)$$

where R is the diameter of the polytope in ℓ_2 norm, i.e. $\|x\|_2 \leq R$ for all $x \in \mathbb{R}^n_{\geq 0}$ with $\mathbf{A}^\top x = b$. Further, the expected running time of the method is $O((nd + d^3)\log^{O(1)} n \log \frac{n}{\delta})$.

The proof for Theorem 1.1 uses four intermediate results, which we formally state in the next Sections 4.1, to 4.4. Each of these intermediate results is self-contained and analyzed in the full version. In this section we show how these results can be combined to obtain Theorem 1.1. The first result is a new, improved IPM as outlined in Section 2.1. The exact statement is given in Section 4.1.Here we give a rough summary to motivate the other three results used by our linear programming algorithm.

Our IPM is robust in the sense, that it makes progress, even if we maintain the primal dual solution pair (x, s) only approximately. Additionally, the linear system that is solved in each iteration, allows for spectral approximations. More accurately, it is enough to maintain a spectral approximation of an inverse of a matrix of the form $\mathbf{A}^\top \mathbf{W} \mathbf{A}$ for some diagonal matrix $\mathbf{W}$. For this robust IPM we also require to compute approximate leverage scores, which allows the IPM to converge in just $\widetilde{\mathrm{O}}(\sqrt{d})$ iterations. These properties of our IPM motivate three new data-structures, all of which are proven and analyzed in the full version:

(i) In Section 4.2, we present a data-structure that can maintain an approximation of the primal dual solution pair (x, s) efficiently. More formally, this data-structure maintains an approximation of the sum $\sum_{k\in[t]} \mathbf{W}^{(k)} \mathbf{A} h^{(k)}$ for diagonal matrices $\mathbf{W}^{(k)} \in \mathbb{R}^{n\times n}$ and vectors $h^{(k)} \in \mathbb{R}^d$.

(ii) In Section 4.3 we present a data-structure that can main approximate leverage scores, required by the IPM.

(iii) The last requirement of the IPM is for us to maintain a spectral approximation of $(\mathbf{A}^\top \mathbf{W} \mathbf{A})^{-1}$ for some diagonal matrix $\mathbf{W}$. We present a data-structure that can maintain this inverse approximately in $\widetilde{\mathrm{O}}(d^{\omega-\frac{1}{2}} + d^2)$ amortized time per step, when the matrix $\mathbf{W}$ changes slowly w.r.t ℓ_2-norm and if we have estimates of the leverage scores of $\mathbf{W}^{1/2}\mathbf{A}$. The exact result is stated in Section 4.4.

With this we have all tools available for proving the main result Theorem 1.1. In Section 4.5 we show how to combine all these tools to obtain the fast linear programming algorithm.

4.1 Interior Point Method

In the full version we derive and analyze the core subroutine of our primal-dual robust $\widetilde{\mathrm{O}}(\sqrt{d})$-iteration IPM (Algorithm 1). This subroutine takes an approximate central point for parameter $\mu^{(\text{init})}$ and in $\widetilde{\mathrm{O}}(\sqrt{d}\log(\mu^{(\text{target})}/\mu^{(\text{init})}))$ iterations outputs an approximate central path point for any given parameter $\mu^{(\text{target})}$. Here we simply state the routine (Algorithm 1) and the main theorem regarding its performance (Theorem 4.1). We defer the full motivation of the method and its analysis (i.e. the proof of Theorem 4.1) to the full version. In the remainder of this section we argue how with the appropriate data-structures, this theorem implies our main result.

Theorem 4.1. *There exists a constant $\zeta > 0$ such that, given $x^{(\text{init})}, s^{(\text{init})} \in \mathbb{R}^n_{>0}$, $\mu^{(\text{init})} > 0$, $\mu^{(\text{target})} > 0$, and $\epsilon \in (0, \alpha/16000)$ with $x^{(\text{init})} s^{(\text{init})} \approx_{2\epsilon} \mu^{(\text{init})} \cdot \tau(x^{(\text{init})}, s^{(\text{init})})$ and*

$$\frac{1}{\mu^{(\text{init})}} \|\mathbf{A}x^{(\text{init})} - b\|^2_{(\mathbf{A}^\top \mathbf{X}^{(\text{init})} \mathbf{S}^{(\text{init})-1} \mathbf{A})^{-1}} \leq \frac{\zeta\epsilon^2}{\log^6 n}$$

Algorithm 1 outputs $(x^{(\text{final})}, s^{(\text{final})})$ such that

$$x^{(\text{final})} s^{(\text{final})} \approx_\epsilon \mu^{(\text{target})} \cdot \tau(x^{(\text{final})}, s^{(\text{final})}) \qquad \text{and}$$

$$\frac{1}{\mu^{(\text{target})}} \|\mathbf{A}x^{(\text{final})} - b\|^2_{(\mathbf{A}^\top \mathbf{X}^{(\text{final})} \mathbf{S}^{(\text{final})-1} \mathbf{A})^{-1}} \leq \frac{\zeta\epsilon^2}{\sqrt{d}\log^6 n}$$

$$\text{in } O\left(\sqrt{d}\log(n) \cdot \left(\frac{1}{\epsilon\alpha} \cdot \log\left(\frac{\mu^{(\text{target})}}{\mu^{(\text{init})}}\right) + \frac{1}{\alpha^3}\right)\right) \text{ iterations.}$$

Furthermore, throughout Algorithm 1, we have

- *$xs \approx_{4\epsilon} \mu \cdot \tau(x, s)$ for some μ where (x, s) is immediate points in the algorithms*
- *$\|\mathbf{X}^{-1}\delta_x\|_{\tau+\infty} \leq \frac{\epsilon}{2}$, $\|\mathbf{S}^{-1}\delta_s\|_{\tau+\infty} \leq \frac{\epsilon}{2}$, $\|\mathrm{diag}(\tau)^{-1}\delta_\tau\|_{\tau+\infty} \leq 2\epsilon$ where δ_x, δ_s and δ_τ is the change of x, s and τ in one iteration and $\|x\|_{\tau+\infty} \stackrel{\text{def}}{=} \|x\|_\infty + C_{\text{norm}}\|x\|_\tau$ for $C_{\text{norm}} \stackrel{\text{def}}{=} \frac{10}{\alpha}$.*

Remark. We will take $\epsilon = 1/\mathrm{poly}\log(n)$.

[2] We assume throughout that $\mathbf{A}$ is *non-degenerate* meaning it has full-column rank and no-zero rows. This assumption can be avoided by preprocessing $\mathbf{A}$ to check for zero rows and adding a tiny amount of noise to the matrix to make it full-column rank. There are other natural ways to remove this assumption, see e.g. [35].

Algorithm 1: Path Following (Theorem 4.1)

1 **procedure** CENTERING($x^{(\text{init})} \in \mathbb{R}^n_{>0}, s^{(\text{init})} \in \mathbb{R}^n_{>0}, \mu^{(\text{init})} > 0, \mu^{(\text{target})} > 0, \epsilon > 0$)
2 $\alpha \leftarrow 1/(4\log(4n/d))$, $\lambda \leftarrow \frac{2}{\epsilon}\log(\frac{2^{16}n\sqrt{d}}{\alpha^2})$, $\gamma \leftarrow \min(\frac{\epsilon}{4}, \frac{\alpha}{50\lambda})$
3 $\mu \leftarrow \mu^{(\text{init})}$, $x \leftarrow x^{(\text{init})}$, $s \leftarrow s^{(\text{init})}$
4 **while** true **do**
5 Pick any $\overline{x}, \overline{s} \in \mathbb{R}^n_{>0}$ such that $\overline{x} \approx_\epsilon x$, $\overline{s} \approx_\epsilon s$
6 Find $\overline{v}$ such that $\|\overline{v} - v\|_\infty \le \gamma$ where $v = \mu \cdot \tau(x,s)/w$
7 Let $\Phi(v) \stackrel{\text{def}}{=} \sum_{i=1}^n \exp(\lambda(v_i - 1)) + \exp(-\lambda(v_i - 1))$ for all $v \in \mathbb{R}^n$
8 **if** $\mu = \mu^{(\text{target})}$ *and* $\Phi(\overline{v}) \le \frac{2^{16}n\sqrt{d}}{\alpha^2}$ **then break**;
9 $h = \gamma\nabla\Phi(\overline{v})^{b(\overline{\tau})}$ for $\overline{\tau} \approx_\epsilon \tau(x,s)$
10 Pick any $\mathbf{H} \in \mathbb{R}^{d\times d}$ with $\mathbf{H} \approx_{(c\epsilon)/(d^{1/4}\log^3 n)} \mathbf{A}^\top\overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}\mathbf{A}$ and $\mathbb{E}[\mathbf{H}] = \mathbf{A}^\top\overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}\mathbf{A}$ for small constant $c > 0$.
11 Let $\mathbf{Q} = \overline{\mathbf{S}}^{-1/2}\overline{\mathbf{X}}^{1/2}\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top\overline{\mathbf{X}}^{1/2}\overline{\mathbf{S}}^{-1/2}$, $\overline{\mathbf{W}} = \overline{\mathbf{X}}\overline{\mathbf{S}}$
12 $x \leftarrow x + (1+2\alpha)\overline{\mathbf{X}}\overline{\mathbf{W}}^{-1/2}(\mathbf{I} - \mathbf{Q})\overline{\mathbf{W}}^{1/2}h$
13 $s \leftarrow s + (1-2\alpha)\overline{\mathbf{S}}\overline{\mathbf{W}}^{-1/2}\mathbf{Q}\overline{\mathbf{W}}^{1/2}h$
14 Pick any $\overline{x}^{(\text{new})}, \overline{s}^{(\text{new})}, \overline{\tau}^{(\text{new})} \in \mathbb{R}^n_{>0}$ with $\overline{x}^{(\text{new})} \approx_\epsilon x$, $\overline{s}^{(\text{new})} \approx_\epsilon s$, $\overline{\tau}^{(\text{new})} \approx_1 \tau(x,s)$
15 $\delta_\lambda \leftarrow$ MAINTAINFEASIBILITY($x, s, \overline{\tau}^{(\text{new})}$)
16 $x \leftarrow x + \overline{\mathbf{X}}^{(\text{new})}\left(\overline{\mathbf{S}}^{(\text{new})}\right)^{-1}\mathbf{A}\delta_\lambda$
17 **if** $\mu > \mu^{(\text{target})}$ **then** $\mu \leftarrow \max\{\mu^{(\text{target})}, (1 - \frac{\gamma\alpha}{2^{15}\sqrt{d}})\mu\}$;
18 **else if** $\mu < \mu^{(\text{target})}$ **then** $\mu \leftarrow \min\{\mu^{(\text{target})}, (1 + \frac{\gamma\alpha}{2^{15}\sqrt{d}})\mu\}$;
19 **return** (x, s)

Note that the complexity of Theorem 4.1 depends on the cost of implementing Lines 5, 6, 12, and 13 of Algorithm 1, as well as the cost of the function MAINTAINFEASIBILITY. Here Lines 5, 12 and 13 ask us to maintain an approximation of the primal dual solution pair (x, s). A data-structure for this task is presented in Section 4.2. Additionally, to compute Lines 12 and 13, we must have access to an approximate inverse of $\mathbf{A}^\top\overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}\mathbf{A}$ (see Line 10). The task of maintaining this inverse will be performed by the data-structure presented in Section 4.4. At last, consider Line 6. To implement this line, we must have an approximation of the leverage scores $\tau(x,s)$. In Section 4.3, we present a data-structure that can efficiently maintain such an approximation.

To help us analyze the cost of function MAINTAINFEASIBILITY we prove the following in the full version.

THEOREM 4.2 (MAINTAIN FEASIBILITY). *The additional amortized cost of calling MAINTAINFEASIBILITY in Line 15 of Algorithm 1 is* $\widetilde{O}(nd^{0.5} + d^{2.5}/\epsilon^2)$ *per call, plus the cost of querying* $\widetilde{O}(n/\sqrt{d} + d^{1.5}/\epsilon^2)$ *entries of* x *and* s *(assuming* x, s *are given implicitly, e.g. via some data structure).*

4.2 Vector Data Structure

Consider an online sequence of $n\times n$ diagonal matrices $\mathbf{G}^{(1)}, ..., \mathbf{G}^{(T)} \in \mathbb{R}^{n\times n}$ and vectors $h^{(1)}, ..., h^{(T)} \in \mathbb{R}^d$, $\delta^{(1)}, ..., \delta^{(T)} \in \mathbb{R}^n$ and define $y^{(t+1)} := \sum_{k=1}^t \mathbf{G}^{(k)}\mathbf{A}h^{(k)} + \delta^{(k)}$. In this subsection we describe a data-structure that can efficiently maintain an approximation $\overline{y}^{(t)} \approx_\varepsilon y^{(t)}$, when the relative changes $\|(\mathbf{Y}^{(k)})^{-1}\mathbf{G}^{(k)}\mathbf{A}h^{(k)}\|_2$ and $\|(\mathbf{Y}^{(k)})^{-1}\delta^{(k)}\|_2$ are small. This is motivated by the following requirement of our IPM: we must maintain a multiplicative approximation of a sequence of vectors $x^{(t)}, s^{(t)} \in \mathbb{R}^n$ (see Line 5 of Algorithm 1), where $x^{(k+1)} = x^{(k)} + \delta_x^{(k)}$, $s^{(k+1)} = s^{(k)} + \delta_s^{(k)}$ and the terms $\delta_x^{(k)}$ and $\delta_s^{(k)}$ are roughly of the form (see Lines 12 and 13 of Algorithm 1):

$$\delta_x^{(k)} = (1+2\alpha)\overline{\mathbf{X}}^{(k)}\left(\overline{\mathbf{W}}^{(k)}\right)^{-1/2}(\mathbf{I} - \mathbf{Q}^{(k)})\left(\overline{\mathbf{W}}^{(k)}\right)^{1/2}v^{(k)},$$
$$\delta_s^{(k)} = (1-2\alpha)\overline{\mathbf{S}}^{(k)}\left(\overline{\mathbf{W}}^{(k)}\right)^{-1/2}\mathbf{Q}^{(k)}\left(\overline{\mathbf{W}}^{(k)}\right)^{1/2}v^{(k)}.$$

To maintain an approximation of $x^{(t)}$, we can then use the data-structure for maintaining an approximation of $y^{(t)}$ by choosing

$$\mathbf{G}^{(k)} = (1+2\alpha)\overline{\mathbf{X}}^{(k)}\left(\overline{\mathbf{W}}^{(k)}\right)^{-1/2},$$
$$h^{(k)} = -\mathbf{Q}^{(k)}\left(\overline{\mathbf{W}}^{(k)}\right)^{1/2}v^{(k)},\ \delta^{(k)} = \left(\overline{\mathbf{W}}^{(k)}\right)^{1/2}v^{(k)}.$$

Likewise, we can maintain an approximation of $s^{(t)}$ by a slightly different choice of parameters. The exact resultis the following Theorem 4.3:

THEOREM 4.3 (VECTOR MAINTENANCE). *There exists a Monte-Carlo data-structure, that works against an adaptive adversary, with the following procedures:*

- *INITIALIZE($\mathbf{A}, g, x^{(0)}, \epsilon$): Given matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, scaling $g \in \mathbb{R}^n$, initial vector $x^{(0)}$, and target accuracy $\epsilon \in (0, 1/10)$, the data-structure preprocesses in $O(\text{nnz}(\mathbf{A})\log^5 n)$ time.*
- *SCALE(i, u): Given $i \in [n]$ and $u \in \mathbb{R}$ sets $g_i = u$ in $O(d\log^5 n)$ amortized time.*
- *QUERY($h^{(t)}, \delta^{(t)}$): Let $g^{(t)} \in \mathbb{R}^n$ be the scale vector $g \in \mathbb{R}^n$ during t-th call to QUERY and let $h^{(t)} \in \mathbb{R}^d, \delta^{(t)} \in \mathbb{R}^n$ be the vectors given during that query. Define*
$$x^{(t)} = x^{(0)} + \sum_{k\in[t]}\mathbf{G}^{(k)}\mathbf{A}h^{(k)} + \sum_{k\in[t]}\delta^{(k)}.$$
Then, w.h.p. in n the data-structure outputs a vector $y \in \mathbb{R}^n$ such that $y \approx_\epsilon x^{(t)}$. Furthermore, the total cost over T steps is
$$O(T(n\log n + \sum_{k\in[T]}\left(\|(\mathbf{X}^{(k)})^{-1}\mathbf{G}^{(k)}\mathbf{A}h^{(k)}\|_2^2 + \|(\mathbf{X}^{(k)})^{-1}\delta^{(k)}\|_2^2\right)\cdot\varepsilon^{-2}\cdot d\log^6 n)).$$
- *COMPUTEEXACT(i): Output $x_i^{(t)} \in \mathbb{R}^n$ exactly in amortized time $O(d\log n)$.*

4.3 Leverage Score Maintenance

The IPM of Theorem 4.1, requires approximate leverage scores of some matrix of the form $\mathbf{GA}$, where $\mathbf{G}$ is a diagonal matrix (see Line 6 of Algorithm 1, where $\mathbf{G} = (\overline{\mathbf{X}}/\overline{\mathbf{S}})^{1/2}$) . Here the matrix $\mathbf{G}$ changes slowly from one iteration of the IPM to the next one, which

allows us to create a data-structure that can maintain the scores more efficiently than recomputing them from scratch every time $\mathbf{G}$ changes.

Theorem 4.4 (Leverage Score Maintenance). *There exists a Monte-Carlo data-structure, that works against an adaptive adversary, with the following procedures:*

- *Initialize*$(\mathbf{A}, g, \epsilon)$*: Given matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$*, scaling* $g \in \mathbb{R}^n$ *and target accuracy* $\epsilon > 0$*, the data-structure preprocesses in* $O(nd\epsilon^{-2}\log^4 n)$ *time.*
- *Scale*(i, u)*: Given* $i \in [n]$ *and* $u \in \mathbb{R}$ *sets* $g_i = u$ *in time* $O(d\epsilon^{-2}\log^5 n)$*.*
- *Query*$(\Psi^{(t)}, \Psi^{(t)}_{(\text{safe})})$*: Let* $g^{(t)}$ *be the vector* g *during* t*-th call to Query and define* $\mathbf{H}^{(t)} = \mathbf{A}^\top(\mathbf{G}^{(t)})^2\mathbf{A}$*. Given random input-matrices* $\Psi^{(t)} \in \mathbb{R}^{d\times d}$ *and* $\Psi^{(t)}_{(\text{safe})} \in \mathbb{R}^{d\times d}$ *such that*

$$\Psi^{(t)} \approx_{\epsilon/(24\log n)} (\mathbf{H}^{(t)})^{-1}, \Psi^{(t)}_{(\text{safe})} \approx_{\epsilon/(24\log n)} (\mathbf{H}^{(t)})^{-1}.$$

and any randomness used to generate $\Psi^{(t)}_{(\text{safe})}$ *is independent of the randomness used to generate* $\Psi^{(t)}$*, w.h.p. in* n *the data-structure outputs a vector* $\widetilde{\tau} \in \mathbb{R}^n$ *independent of* $\Psi^{(1)}, \ldots, \Psi^{(t)}$ *such that* $\widetilde{\tau}_i \approx_\epsilon \tau_i(\mathbf{G}^{(t)}\mathbf{A})$ *for all* $i \in [n]$*.*

The total cost of T *calls to Query is*

$$O((\Sigma_{t\in[T]}\|\mathbf{G}^{(t)}\mathbf{A}\Psi^{(t)}\mathbf{A}^\top\mathbf{G}^{(t)} - \mathbf{G}^{(t-1)}\mathbf{A}\Psi^{(t-1)}\mathbf{A}^\top\mathbf{G}^{(t-1)}\|_F)^2 \cdot\epsilon^{-4}n\log^7 n + T(T_\Psi + \epsilon^{-2}d^2\log^3 n))$$

where T_Ψ *is the time required to multiply a vector with* $\Psi^{(t)}$ *(i.e. in case it is given implicitly via a data structure).*

4.4 Inverse Maintenance

For the IPM we must approximately maintain the inverse $(\mathbf{A}^\top\mathbf{W}\mathbf{A})^{-1}$ where $\mathbf{A} \in \mathbb{R}^{n\times d}$ undergoes changes to the diagonal matrix $\mathbf{W}$ (see Line 10 of Algorithm 1 where $\mathbf{W} = \overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}$). By using estimates of the leverage scores of $\mathbf{W}^{1/2}\mathbf{A}$ (as maintained by Theorem 4.4, Section 4.3), we are able to maintain the inverse in amortized $\widetilde{O}(d^{\omega-\frac{1}{2}} + d^2)$ time per step, even for $n \gg d$. The exact result is stated as Theorem 4.5.

Theorem 4.5 (Inverse Maintenance). *Given a full rank matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *with* $n \geq d$ *and error tolerance* $\epsilon \in (0, 1/10)$*, there is a data structure that approximately solves a sequence of linear systems* $\mathbf{A}^\top\widetilde{\mathbf{W}}\mathbf{A}y = b \in \mathbb{R}^d$ *for positive diagonal matrices* $\widetilde{\mathbf{W}} \in \mathbb{R}^{n\times n}$ *through the following operation:*

- *Initialize*$(\mathbf{A}, \widetilde{w}, \widetilde{\tau}, \epsilon)$*: Given matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$*, scaling* $\widetilde{w} \in \mathbb{R}^n_{>0}$*, shifted leverage score estimates* $\widetilde{\tau} \in \mathbb{R}^n_{>0}$*, and accuracy* $\epsilon \in (0, 1/10)$*, the data-structure preprocesses in* $O(d^\omega)$ *time.*
- *Update*$(\widetilde{w}, \widetilde{\tau})$*: Output a matrix* $\Psi \in \mathbb{R}^{d\times d}$ *and vector* $\widetilde{w}^{(\text{alg})}$ *where* Ψ^{-1} *is close to* $\mathbf{A}^\top\widetilde{\mathbf{W}}\mathbf{A} \in \mathbb{R}^{d\times d}$ *and* $\widetilde{w}^{(\text{alg})}$ *is close to* $\widetilde{w}$*.*
- *Solve*$(b, \overline{w}, \delta)$*: Input* $\overline{w} \approx_1 \widetilde{w}$ *and* $\delta > 0$*, output* $y = \Psi b \in \mathbb{R}^d$ *for some random matrix* $\Psi^{-1} \in \mathbb{R}^{d\times d}$ *that is close to* $\mathbf{A}^\top\overline{\mathbf{W}}\mathbf{A} \in \mathbb{R}^{d\times d}$*.*

Let $\tau(w) \stackrel{\text{def}}{=} \tau(\mathbf{W}\mathbf{A})$*. Suppose that all estimate shifted leverage scores* $\widetilde{\tau}_i \in (1 \pm \frac{1}{16\lceil\log d\rceil})\tau(w)_i$ *for* $i \in [n]$[3] *and that there is a sequence* $w^{(0)}, w^{(1)}, \cdots, w^{(K)} \in \mathbb{R}^n$ *such that the* $w^{(k)}$ *satisfy*

$$\frac{1}{\epsilon^2}\|(\mathbf{W}^{(k)})^{-1}(w^{(k+1)} - w^{(k)})\|^2_{\tau(w^{(k)})} + \|(\mathbf{T}(w^{(k)}))^{-1}(\tau(w^{(k+1)}) - \tau(w^{(k)}))\|^2_{\tau(w^{(k)})} \leq \frac{1}{80} \quad (3)$$

for $k = 0, 1, \cdots, K-1$ *with* $K = n^{O(1)}$*. Further assume that the update sequence* $\widetilde{w}^{(0)}, \widetilde{w}^{(1)}, \cdots, \widetilde{w}^{(K)} \in \mathbb{R}^n$ *satisfies* $\widetilde{w}^{(k)} \approx_{\epsilon/(16\log d)} w^{(k)}$ *for all* k *and the* $\widetilde{w}^{(k)}$ *are independent to the output of Update and Solve. (The input* $b^{(k)}$ *can depend on the previous output of the data structure.) Then, we have the following:*

- *The amortized time per call of Update is* $O(\epsilon^{-2}\cdot(d^{\omega-\frac{1}{2}} + d^2)\cdot\log^{3/2}(n))$*.*
- *The time per call of Solve is* $O(\delta^{-2}\cdot d^2\cdot\log^2(n/\delta))$*.*
- *Update outputs some* Ψ*,* $\widetilde{w}^{(\text{alg})}$ *where* $\Psi^{-1} = \mathbf{A}^\top\widetilde{\mathbf{W}}^{(\text{alg})}\mathbf{A} \approx_\epsilon \mathbf{A}^\top\widetilde{\mathbf{W}}\mathbf{A}$ *with probability* $1 - 1/\text{poly}(n)$ *and* $\widetilde{w}^{(\text{alg})} \approx_\epsilon \widetilde{w}$*.*
- *Solve outputs some* $y = \Psi b$ *where* $\Psi^{-1} \approx_\delta \mathbf{A}^\top\overline{\mathbf{W}}\mathbf{A}$ *with probability* $1 - 1/\text{poly}(n)$ *and* $\mathbb{E}[\Psi b] = (\mathbf{A}^\top\overline{\mathbf{W}}\mathbf{A})^{-1}b$*.*

In general, Theorem 4.5 does not work against adaptive adversaries, i.e. the input w and $\widetilde{\tau}$ to the Update procedure is not allowed to depend on previous outputs. In the full version we show, that this algorithm can be improved such that the input w and $\widetilde{\tau}$ is allowed to depend on the output of Solve. However, the input is still not allowed to depend on the output of Update.

Lemma 4.6. *Theorem 4.5 holds even if the input* w *and* $\widetilde{\tau}$ *of the algorithm depends on the output of Solve. Furthermore, we have*

$$\mathbb{E}\Big[\sum_{k\in[K-1]}\Big\|\sqrt{\widetilde{\mathbf{W}}^{(\text{alg})(k+1)}}\mathbf{A}\Psi^{(k+1)}\mathbf{A}^\top\sqrt{\widetilde{\mathbf{W}}^{(\text{alg})(k+1)}} - \sqrt{\widetilde{\mathbf{W}}^{(\text{alg})(k)}}\mathbf{A}\Psi^{(k)}\mathbf{A}^\top\sqrt{\widetilde{\mathbf{W}}^{(\text{alg})(k)}}\Big\|_F\Big] \leq 16K\log^{5/2}n$$

where $\Psi^{(k)} \in \mathbb{R}^{d\times d}$*,* $\widetilde{w}^{(\text{alg})(k)}$ *is the output of the* k*-th step of Update(* $\widetilde{w}^{(k)}$*,* $\widetilde{\tau}^{(k)}$*).*

4.5 Linear Programming Algorithm

Here we show how to combine the tools from Section 4.1 to 4.4 to obtain a linear program solver that runs in $\widetilde{O}(nd + d^3)$ time. First, we give a brief summary of our linear programming algorithm, Algorithm 2. The algorithm consists of two phases. In the first phase we construct a good initial feasible solution, and in the second we move along the central path towards the optimal solution.

The construction of the initial point works as follows: via a simple transformation (stated below as Theorem 4.7), we obtain a feasible solution pair (x, s) where both x and s are close to the all 1 vector and hence good enough as a point close to the central path of the standard log barrier function. However, we need to find a point such that $xs \approx_\epsilon \mu\cdot(\sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1)$. By picking $x = 1$ and $\mu = 1$, the initial slack s needs to satisfy $s \approx_\epsilon \sigma(\mathbf{S}^{-\frac{1}{2}-\alpha}\mathbf{A}) + \frac{d}{n}1$, which (up to the additive $\frac{d}{n}1$) is exactly the condition for ℓ_p Lewis weight with $p = \frac{1}{1+\alpha}$. Cohen and Peng showed that such a vector s can be found efficiently as long as $p \in (0, 4)$ [13]. We note that such s might not satisfy $\mathbf{A}y + s = c$. That is why we define $c^{(\text{tmp})} := \mathbf{A}y + s$ for $y = 0$, so that (x, s) is a feasible solution pair for the cost vector $c^{(\text{tmp})}$. In the first phase of Algorithm 2, we move the point (x, s)

[3] Recall that $\tau(w)_i = (\sqrt{\mathbf{W}}\mathbf{A}(\mathbf{A}^\top\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^\top\sqrt{\mathbf{W}})_{i,i} + \frac{d}{n}, \forall i \in [n]$

along the central path of the temporary cost vector $c^{(\text{tmp})}$ and bring the points to a location where we can switch the cost $c^{(\text{tmp})}$ to c without violating the centrality conditions. This is how we obtain our feasible starting point for the cost vector c. In the subsequent second phase, we move along the path for the cost c until it is close to the optimal solution.

Moving along these two paths of the first and second phase is performed via the IPM of Algorithm 1 (Section 4.1, Theorem 4.1). Note that Algorithm 1 does not specify in Line 5 how to obtain the approximate solution pair $(\overline{x}, \overline{s})$, so we must implement this step on our own. Likewise, we must specify how to efficiently compute the steps in Lines 12 and 13. These implementations can be found in the second part of Algorithm 2. The high-level idea is to use the data-structures presented in Section 4.2 to 4.4.

To illustrate, consider Line 13 of Algorithm 1, which computes

$$s \leftarrow s + (1-2\alpha)\overline{\mathbf{SW}}^{-1/2}\mathbf{Q}\overline{\mathbf{W}}^{1/2}h$$

for $\overline{\mathbf{W}} = \overline{\mathbf{X}}\overline{\mathbf{S}}$ and $\mathbf{Q} = \overline{\mathbf{S}}^{-1/2}\overline{\mathbf{X}}^{1/2}\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top\overline{\mathbf{X}}^{1/2}\overline{\mathbf{S}}^{-1/2}$ for $\mathbf{H} \approx_\epsilon \mathbf{A}^\top\overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}\mathbf{A}$. We split this task into three parts: (i) compute $r := \mathbf{A}^\top\overline{\mathbf{X}}^{1/2}\overline{\mathbf{S}}^{-1/2}\overline{\mathbf{W}}^{1/2}h$, (ii) compute $v := \mathbf{H}^{-1}r$, and (iii) compute $(1-2\alpha)\overline{\mathbf{SW}}^{-1/2}\overline{\mathbf{S}}^{-1/2}\overline{\mathbf{X}}^{1/2}\mathbf{A}v = (1-2\alpha)\mathbf{A}v$. Part (i), the vector r, can be maintained efficiently, because we maintain the approximate solutions $\overline{x}, \overline{s}$ (thus also $\overline{w}$) and vector h, (via an Algorithm in the full version) in such a way, that per iteration only few entries change on average. Part (ii) is solved by the inverse maintenance data-structure of Section 4.4 (Theorem 4.5). The last part (iii) is solved implicitly by the data-structure of Section 4.2 (Theorem 4.3), which is also used to obtain the approximate solutions $\overline{x}, \overline{s}$ in Line 5 of Algorithm 1. We additionally run the data-structure of Section 4.3 (Theorem 4.4) in parallel, to maintain an approximation of the leverage scores, which allows us to find the approximation $\overline{v}$ required in Line 6. These modifications to Algorithm 1 are given in the second part of Algorithm 2.

The following theorem shows how to reduce solving any bounded linear program to solving a linear program with a non-degenerate constrain matrix and an explicit initial primal and dual interior point.

Theorem 4.7 (Initial Point). *Consider some linear program* $\min_{\mathbf{A}^\top x = b, x \geq 0} c^\top x$ *with* n *variables and* d *constraints. Assume that*
1. Diameter of the polytope: $\|x\|_2 \leq R$ *for all* $x \geq 0$ *with* $\mathbf{A}^\top x = b$
2. Lipschitz constant of the linear program: $\|c\|_2 \leq L$.
3. The constraint matrix $\mathbf{A}$ *is non-degenerate.*
For any $\delta \in (0, 1]$, *the modified linear program* $\min_{\overline{\mathbf{A}}^\top \overline{x} = \overline{b}, \overline{x} \geq 0} \overline{c}^\top \overline{x}$ *with*

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & 1_n \|\mathbf{A}\|_F \\ 0 & 1\|\mathbf{A}\|_F \\ \frac{1}{R}b^\top - 1_n^\top \mathbf{A} & 0 \end{bmatrix} \in \mathbb{R}^{(n+2)\times(d+1)},$$

$$\overline{b} = \begin{bmatrix} \frac{1}{R}b \\ (n+1)\|\mathbf{A}\|_F \end{bmatrix} \in \mathbb{R}^{d+1} \text{ and } \overline{c} = \begin{bmatrix} \frac{\delta}{L}\cdot c \\ 0 \\ 1 \end{bmatrix} \in \mathbb{R}^{n+2}$$

satisfies the following:

1. $\overline{x} = \begin{bmatrix} 1_n \\ 1 \\ 1 \end{bmatrix}, \overline{y} = \begin{bmatrix} 0_d \\ -1 \end{bmatrix}$ *and* $\overline{s} = \begin{bmatrix} 1_n + \frac{\delta}{L}\cdot c \\ 1 \\ 1 \end{bmatrix}$ *are feasible.*

Algorithm 2: LP Algorithm (based on Algorithm 1)

1 **global variables**
2 $\quad \mathbf{A} \in \mathbb{R}^{n\times d}$, $\mu > 0$
3 $\quad D^{-1}$; // Theorem 4.5
4 $\quad D^{(x)}, D^{(s)}$; // Theorem 4.3
5 $\quad D_{\text{Leverage}}$; // Theorem 4.4
6 $\quad D_{\text{Gradient}}$; // See appendix of full version
7 $\quad \overline{\tau} \in \mathbb{R}^n$; // $\overline{\tau}_{\gamma/8} \approx (\sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1)$
$\quad$ /* Same parameters as in Theorem 4.1 */
8 $\quad \alpha \stackrel{\text{def}}{=} 1/(4\log(4n/d))$, $\epsilon \stackrel{\text{def}}{=} \frac{\alpha}{16000}$
9 $\quad \lambda \stackrel{\text{def}}{=} \frac{2}{\epsilon}\log(\frac{2^{16}n\sqrt{d}}{\alpha^2})$, $\gamma \stackrel{\text{def}}{=} \min(\epsilon/4, \frac{\alpha}{50\lambda})$
10 **procedure** *Solve*$(\mathbf{A} \in \mathbb{R}^{n\times d}, b \in \mathbb{R}^n, c \in \mathbb{R}^d, \delta > 0)$
11 $\quad$ Modify the LP and obtain an initial x, y and s by Lemma 4.7 to accuracy $\delta/8n^2$
$\quad$ /* For notational simplicity, we use $\mathbf{A}, b, c, n, d$ for the modified LP induced by Lemma 4.7 in the remainder of the code. */
12 $\quad$ Find s with $s \approx_\epsilon \sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$ via Theorem 4.8
13 $\quad \tau \leftarrow s, \mu \leftarrow 1$ // Since $x = 1$ (Lemma 4.7), $xs \approx_\epsilon \mu\tau$
14 $\quad D^{-1}$.Initialize$(\mathbf{A}, \mathbf{S}^{-1-2\alpha}x^{1-2\alpha}, \tau, \gamma/512\log n)$
15 $\quad D_{\text{Leverage}}$.Initialize$(\mathbf{A}, \mathbf{S}^{-1-2\alpha}x^{1-2\alpha}, \gamma/8)$
16 $\quad D^{(x)}$.Initialize$(\mathbf{A}, \mathbf{S}^{-1}x, x, \gamma/8)$
17 $\quad D^{(s)}$.Initialize$(\mathbf{A}, 1, s, \gamma/8)$
18 $\quad D_{\text{Gradient}}$.Initialize$(\mathbf{A}, \mu\tau/(xs), \tau, x, \gamma)$
19 $\quad c^{(\text{tmp})} \leftarrow s$; // $c^{(\text{tmp})} = \mathbf{A}y + s$ for $y = 0$
20 $\quad x, s, \overline{\tau}, \mu \leftarrow$ Centering$(x, s, \tau, \mu, \Theta(n^2\sqrt{d}/(\gamma\alpha^2)))$
21 $\quad s^{(\text{new})} \leftarrow s + c - c^{(\text{tmp})}$
22 $\quad D^{(s)}$.Initialize$(\mathbf{A}, 1, s^{(\text{new})}, \gamma/8)$
23 $\quad x, s, \overline{\tau}, \mu \leftarrow$ Centering$(x, s, \overline{\tau}, \mu, \delta^2/(8^3n^4d))$
24 $\quad$ Reduce $\frac{1}{\mu}\|\mathbf{A}^\top x - b\|^2_{(\mathbf{A}^\top\mathbf{X}\mathbf{S}^{-1}\mathbf{A})^{-1}}$ to some small $O(\delta/n^2)$
25 $\quad$ Return an approximate solution of the original linear program according to Lemma 4.7

2. Let $(\overline{x}, \overline{y}, \overline{s})$ *be primal dual vectors of the modified LP and* $\Phi_b \stackrel{\text{def}}{=} \frac{1}{\mu}\cdot\|\overline{\mathbf{A}}^\top\overline{x} - \overline{b}\|^2_{(\overline{\mathbf{A}}^\top\overline{\mathbf{X}\mathbf{S}}^{-1}\overline{\mathbf{A}})^{-1}}$, *then* $\|\overline{x}\|_\infty \leq (1 + O(\Phi_b))\cdot O(n)$.
3. Let $(\overline{x}, \overline{y}, \overline{s})$ *be primal dual vectors of the modified LP with* $\overline{x}\cdot\overline{s} \approx_{0.5} \mu\cdot\tau(\overline{x}, \overline{s})$ *for* $\mu < \delta^2/(8d)$ *and small enough* $\Phi_b := \frac{1}{\mu}\cdot\|\overline{\mathbf{A}}^\top\overline{x} - \overline{b}\|^2_{(\overline{\mathbf{A}}^\top\overline{\mathbf{X}\mathbf{S}}^{-1}\overline{\mathbf{A}})^{-1}} = O(1)$ *(i.e.* $\overline{x}$ *does not have to be feasible). The vector* $\widehat{x} \stackrel{\text{def}}{=} R\cdot\overline{x}_{1:n}$ *where* $\overline{x}_{1:n}$ *is the first* n *coordinates of* $\overline{x}$ *is an approximate solution to the original linear program in the following sense*

$$c^\top\widehat{x} \leq \min_{\mathbf{A}^\top x = b, x\geq 0} c^\top x + O(nLR)\cdot(\sqrt{\Phi_b} + \delta),$$

$$\|\mathbf{A}^\top\widehat{x} - b\|_2 \leq O(n^2)\cdot(\|\mathbf{A}\|_F R + \|b\|_2)\cdot(\sqrt{\Phi_b} + \delta)\Big),$$

$$\widehat{x} \geq 0.$$

As outlined before, the initial points given in Theorem 4.7 do not satisfy $xs \approx_\epsilon \mu\cdot(\sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + (d/n)1)$. To satisfy this condition we pick $x = 1$ and $\mu = 1$, and pick the initial slack vector s a to satisfy $s \approx_\epsilon \sigma(\mathbf{S}^{-\frac{1}{2}-\alpha}\mathbf{A}) + \frac{d}{n}1$, which is exactly the condition

Algorithm 3: LP Algorithm (based on Algorithm 1), Continuation of Algorithm 2.

procedure CENTERING($x^{(\text{init})} \in \mathbb{R}^n_{>0}, s^{(\text{init})} \in \mathbb{R}^n_{>0}, \tau^{(\text{init})} > 0, \mu^{(\text{init})} > 0, \mu^{(\text{target})} > 0$)
  $\mu \leftarrow \mu^{(\text{init})}$, $\bar{\tau} \leftarrow \tau^{(\text{init})}$, $\tau^{(\text{tmp})} \leftarrow \tau^{(\text{init})}$, $\bar{x} \leftarrow x^{(\text{init})}$, $x^{(\text{tmp})} \leftarrow x^{(\text{init})}$, $\bar{s} \leftarrow s^{(\text{init})}$, $s^{(\text{tmp})} \leftarrow s^{(\text{init})}$
  while TRUE **do**
    $\bar{x}_i = x_i^{(\text{tmp})}$ for all i such that $\bar{x}_i \not\approx_{\gamma/8} x_i^{(\text{tmp})}$
    $\bar{s}_i = s_i^{(\text{tmp})}$ for all i such that $\bar{s}_i \not\approx_{\gamma/8} s_i^{(\text{tmp})}$
    $\bar{\tau}_i = \tau_i^{(\text{tmp})}$ for all i such that $\bar{\tau}_i \not\approx_{\gamma/8} \tau_i^{(\text{tmp})}$
    $\overline{w} \leftarrow \overline{\mathbf{X}}\overline{s}$, $\overline{v} \leftarrow \mu \cdot \overline{\mathbf{W}}^{-1}\overline{\tau}$
    if $\mu = \mu^{(\text{target})}$ *and* $\Phi(\overline{v}) \leq \frac{2^{16}n\sqrt{d}}{\alpha^2}$ **then break**;
    D_{GRADIENT}.UPDATE($i, \overline{v}_i, \bar{\tau}_i, \bar{x}_i$) for i where $\overline{v}_i$, $\bar{\tau}_i$ or $\bar{x}_i$ changed
    $h, r \leftarrow D_{\text{GRADIENT}}$.QUERY()
    $\Psi^{(\alpha)}, g \leftarrow D^{-1}$.UPDATE($\overline{\mathbf{S}}^{-1-2\alpha}\overline{x}^{1-2\alpha}, \overline{\tau}$)
    D_{LEVERAGE}.UPDATE($i, \sqrt{g}$) for i where g_i changed
    $\Psi^{(\alpha)}_{(\text{safe})}(b) \stackrel{\text{def}}{=} D^{-1}$.SOLVE($b, \overline{\mathbf{S}}^{-1-2\alpha}\overline{x}^{1-2\alpha}, \frac{\gamma}{512\log n}$)
    $\Psi_{(\text{safe})}(b) \stackrel{\text{def}}{=} D^{-1}$.SOLVE($b, \overline{\mathbf{S}}^{-1}\overline{x}, (c\epsilon)/(d^{1/4}\log^3 n)$)
    $D^{(x)}$.SCALE($\overline{x}/\bar{s}$) // Only scale coordinates where $\bar{x}$ or $\bar{s}$ changed.
    $x^{(\text{tmp})} \leftarrow D^{(x)}$.QUERY($(1+2\alpha)\Psi_{(\text{safe})}r, (1+2\alpha)\overline{\mathbf{X}}h$)
    $s^{(\text{tmp})} \leftarrow D^{(s)}$.QUERY($(1-2\alpha)\Psi_{(\text{safe})}r, 0_n$)
    $\tau^{(\text{tmp})} \leftarrow D_{\text{LEVERAGE}}$.QUERY($\Psi^{(\alpha)}, \Psi^{(\alpha)}_{(\text{safe})}$)
    $\delta_\lambda \leftarrow$ MAINTAINFEASIBILITY($D^{(x)}, D^{(s)}, \mu$)
    $x^{(\text{tmp})} \leftarrow D^{(x)}$.QUERY($\delta_\lambda, 0$)
    if $\mu > \mu^{(\text{target})}$ **then**
      $\mu \leftarrow \max(\mu^{(\text{target})}, (1 - \frac{\gamma\alpha}{2^{15}\sqrt{d}})\mu)$;
    else if $\mu < \mu^{(\text{target})}$ **then**
      $\mu \leftarrow \min(\mu^{(\text{target})}, (1 + \frac{\gamma\alpha}{2^{15}\sqrt{d}})\mu)$;
  $x \leftarrow D^{(x)}$.COMPUTEEXACT($1, \ldots, n$)
  $s \leftarrow D^{(s)}$.COMPUTEEXACT($1, \ldots, n$)
  return $(x, s, \tau^{(\text{tmp})}, \mu)$

for ℓ_p Lewis weight with $p = \frac{1}{1+\alpha}$. The following theorem shows that such a vector s can be found efficiently as long as $p \in (0,4)$. This vector s might not be a valid slack vector, so as outlined before, the algorithm runs in two phases: first using a cost vector $c^{(\text{tmp})}$ for which the initial s is feasible, and then switching to the correct c.

THEOREM 4.8 ([13]). *Given $p \in (0,4)$, $\eta > 0$ and non-degenerate $\mathbf{A} \in \mathbb{R}^{n\times d}$ w.h.p. in n, we can compute $w \in \mathbb{R}^n_{>0}$ with $w \approx_\epsilon \sigma(\mathbf{W}^{\frac{1}{2}-\frac{1}{p}}\mathbf{A}) + \eta 1$ in $\widetilde{\mathrm{O}}((\mathrm{nnz}(\mathbf{A}) + d^\omega)\mathrm{poly}(1/\epsilon))$ time.*

PROOF. Our proof is similar to [13], which proved a variant when $\eta = 0$ Consider the map $T(w) \stackrel{\text{def}}{=} (\mathbf{W}^{\frac{2}{p}-1}(\sigma(\mathbf{W}^{\frac{1}{2}-\frac{1}{p}}\mathbf{A}) + \eta 1))^{p/2}$. Further, fix any positive vectors $v, w \in \mathbb{R}^n$ such that $v \approx_\alpha w$. We have that $\mathbf{A}^\top\mathbf{V}^{1-\frac{1}{p}}\mathbf{A} \approx_{|1-\frac{2}{p}|\alpha} \mathbf{A}^\top\mathbf{W}^{1-\frac{2}{p}}\mathbf{A}$ and hence

$$a_i^\top(\mathbf{A}^\top\mathbf{V}^{1-\frac{2}{p}}\mathbf{A})^{-1}a_i \approx_{|1-\frac{2}{p}|\alpha} a_i^\top(\mathbf{A}^\top\mathbf{W}^{1-\frac{2}{p}}\mathbf{A})^{-1}a_i. \tag{4}$$

Note that

$$T(v)_i^{2/p} = \frac{v_i^{1-\frac{2}{p}}a_i^\top(\mathbf{A}^\top\mathbf{V}^{1-\frac{2}{p}}\mathbf{A})^{-1}a_i + \eta}{v_i^{1-\frac{2}{p}}} = a_i^\top(\mathbf{A}^\top\mathbf{V}^{1-\frac{2}{p}}\mathbf{A})^{-1}a_i + \eta v_i^{\frac{2}{p}-1}.$$

Using (4), we have

$$\begin{aligned} T(v)_i^{2/p} &\leq e^{|1-\frac{2}{p}|\alpha}a_i^\top(\mathbf{A}^\top\mathbf{W}^{1-\frac{2}{p}}\mathbf{A})^{-1}a_i + \eta v_i^{\frac{2}{p}-1} \\ &\leq e^{|1-\frac{2}{p}|\alpha}(a_i^\top(\mathbf{A}^\top\mathbf{W}^{1-\frac{2}{p}}\mathbf{A})^{-1}a_i + \eta w_i^{\frac{2}{p}-1}) \\ &= e^{|1-\frac{2}{p}|\alpha}T(w)_i^{2/p}. \end{aligned}$$

Similarly, we have $T(v)_i^{2/p} \geq e^{-|1-\frac{2}{p}|\alpha}T(w)_i^{2/p}$. Taking $p/2$ power of both sides, we have

$$e^{-|\frac{p}{2}-1|\alpha}T(w)_i \leq T(v)_i \leq e^{|\frac{p}{2}-1|\alpha}T(w)_i.$$

Hence, $T(v) \approx_{|p/2-1|\alpha} T(w)$.

Consequently, let $w_0 = \eta 1$ and consider the algorithm $w_{k+1} = T(w_k)$. Since $\eta > 0$ we have that $w_0 = w_0\eta^{-p/2}\eta^{p/2} \leq T(w_0)$ and $T(w_0) \leq w_0\eta^{-p/2}(1+\eta)^{p/2} \leq w_0\exp(p\eta^{-1}/2)$. Consequently, $T(w_0) \approx_{p\eta^{-1}/2} w_0$ and after k steps we have that

$$T(w_k) \approx_{\exp(|p/2-1|^k p\eta^{-1}/2)} w_k.$$

Since for $p \in (0,4)$, we have that $|p/2-1| < 1$ we see that after $O(\log(\eta^{-1}/\epsilon))$ steps we have $w_k \approx_\epsilon T(w_k)$. Further, since $w_k \geq \eta 1$ implies that $T(w_k) \geq \eta 1$ we have that $w_k \in \mathbb{R}^n_{>0}$ as desired. To implement the steps, one can check, by the same proof, that it suffices to get a $\approx_{O(\epsilon)}$ multiplicative approximation to $T(w)$ in each step, which can be done in $\widetilde{\mathrm{O}}((\mathrm{nnz}(\mathbf{A})+d^\omega)/\epsilon^2)$ per step by standard leverage score estimation techniques. □

Now, we first prove the correctness of the Algorithm 2.

LEMMA 4.9. *Algorithm 2 outputs x such that w.h.p. in n*

$$c^\top x \leq \min_{\mathbf{A}^\top x = b, x \geq 0} c^\top x + LR\cdot\delta,$$
$$\|\mathbf{A}^\top x - b\|_2 \leq \delta\cdot\left(\|\mathbf{A}\|_F\cdot R + \|b\|_2\right),$$
$$x \geq 0.$$

PROOF. We define the primal dual point (x, s) via the formula of lines 12 and 13. We start by showing that our implementation of Algorithm 1 in Algorithm 2 satisfies all required conditions, i.e. that we can apply Theorem 4.1. Throughout this proof, states hold only w.h.p. and therefore the restatement of this is often omitted for brevity.

Invariant: $\overline{x} \approx_{\gamma/4} x$, $\bar{s} \approx_{\gamma/4} s$, $\overline{\tau} \approx_{\gamma/4} \sigma(\overline{\mathbf{S}}^{-1/2-\alpha}\overline{\mathbf{X}}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$ and $\Psi_{(\text{safe})} \approx_\epsilon (\mathbf{A}^\top\overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}\mathbf{A})^{-1}$: We first show that throughout CENTERING, we have the invariant above, assuming the input parameter $\tau_i^{(\text{init})}$ satisfied $\tau_i^{(\text{init})} \approx_{\gamma/4} \sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$. Theorem 4.3 shows that $x^{(\text{tmp})} \approx_{\gamma/8} x$ and $s^{(\text{tmp})} \approx_{\gamma/8} s$ (Line 41 and 42).

Then, by the update rule (Line 29 and 30), we have that $x^{(\text{tmp})} \approx_{\gamma/8} \overline{x}$ and $s^{(\text{tmp})} \approx_{\gamma/8} \overline{s}$. Hence, we have the desired approximation for $\overline{x}$ and $\overline{s}$. Next, Theorem 4.5 shows that

- $\Psi^{(\alpha)} \approx_{\gamma/(512\log n)} (\mathbf{A}^\top \mathbf{S}^{-1-2\alpha}\mathbf{X}^{1-2\alpha}\mathbf{A})^{-1}$ (Line 14) and
- $\Psi^{(\alpha)}_{(\text{safe})} \approx_{\gamma/(512\log n)} (\mathbf{A}^\top \mathbf{S}^{-1-2\alpha}\mathbf{X}^{1-2\alpha}\mathbf{A})^{-1}$ (Line 38).

Hence, we can apply Theorem 4.4 and get

$$\tau^{(\text{tmp})} \approx_{\gamma/8} \sigma(\overline{\mathbf{S}}^{-1/2-\alpha}\overline{\mathbf{X}}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$$

(Line 15 and Line 43). Again, by update rule (Line 31), we have the desired approximation for $\overline{\tau}$.

Finally, $\Psi_{(\text{safe})} \approx_\epsilon (\mathbf{A}^\top \overline{\mathbf{S}}^{-1}\overline{\mathbf{X}}\mathbf{A})^{-1}$ follows from Theorem 4.5 (Line 39). These invariants also imply $\|\overline{v} - v\|_\infty \le \gamma$ and that D_{Gradient} maintains $\nabla\Phi(\overline{v}')^{\flat(\overline{\tau})}$ for some $\|\overline{v}' - v\|_\infty \le \gamma$ (see full version). Thus, in summary, Centering of Algorithm 2 behaves like Centering of Theorem 4.1.

Invariant: $xs \approx \mu \cdot \tau(x,s)$: Initially, $x = 1$ due to the reduction (Lemma 4.7), $\mu = 1$ and $s \approx_\epsilon \sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$ (Line 12). Hence, $xs \approx_\epsilon \mu \cdot (\sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1)$ initially.

We change x, s, μ in Line 20 by calling Centering. By Theorem 4.1 we then have $xs \approx_\epsilon \mu\tau$ after this call to Centering. Next, consider the step where we switch the cost vector on Line 21. Let $s^{(\text{new})}$ be the vector s after Line 21 and s is before Line 21. Since $\mathbf{A}y + s = c^{(\text{tmp})}$ we have that $\mathbf{A}y + s^{(\text{new})} = c^{(\text{tmp})} - s + s^{(\text{new})} = c$, i.e. s is a valid slack vector for cost c. Further, we have that, by design

$$\frac{s^{(\text{new})} - s}{s} = \frac{c - c^{(\text{tmp})}}{s}.$$

First, we bound the denominator. We have that $sx \approx_{1/2} \mu\tau$, so for all $i \in [n]$ we have

$$s_i \ge \frac{\mu}{2x_i}\frac{d}{n} \ge \frac{\mu}{\Omega(n^2)}$$

where we used $\|x\|_\infty \le O(n)$ by Lemma 4.7 as we ensure x is sufficiently close to feasible for the modified linear program by Theorem 4.1. For the numerator, we note that $\|c\|_\infty \le 1$ for the modified linear program and $\|c^{(\text{tmp})}\|_\infty = \|s\|_\infty \le 3$ for the s computed by Theorem 4.8 in Line 12.again by the definition of s (Line 12) and the modified $\mathbf{A}$ and y. Hence, we have that

$$\left\|\frac{s^{(\text{new})} - s}{s}\right\|_\infty \le \frac{16n^2}{\mu} \le \frac{\gamma\alpha^2}{c\cdot\sqrt{d}}.$$

for any constant c by choosing the constant in the $O(\cdot)$ in Line 20 appropriately. Thus $xs \approx_{2\epsilon} \mu\cdot\tau(x,s)$ and $\overline{\tau} \approx_{\gamma/4} \sigma(\mathbf{S}^{-1/2-\alpha}\mathbf{X}^{1/2-\alpha}\mathbf{A}) + \frac{d}{n}1$, so when we call Centering in Line 23, we again obtain $xs \approx_\epsilon \mu \cdot \tau(x,s)$.

Conclusion: Before the algorithm ends, we have $xs \approx_{1/4} \mu\tau$. In Line 24, we reduce $\Phi_b := \|\mathbf{A}^\top x - b\|^2_{(\mathbf{A}^\top\mathbf{X}\mathbf{S}^{-1}\mathbf{A})^{-1}}$ to some small enough $\Phi_b = O(\delta/n^2)$. As argued in the full version, this does not move the vector x too much, i.e. we still have $x \cdot s \approx_{1/2} \mu \cdot \tau(x,s)$. Hence, by the choice of the new μ in Line 23, Lemma 4.7 shows that we can output a point $\widehat{x}$ with the desired properties. □

Finally, we analyze the cost of the Algorithm 2.

Lemma 4.10. *Algorithm 2 takes $O((nd + d^3)\log^{O(1)} n \log(n/\delta))$ time with high probability in n.*

Proof. For simplicity, we use $\widetilde{O}(\cdot)$ to suppress all terms that are $\log^{O(1)} n$. We first note that all parameters $\alpha, \epsilon, \lambda, \gamma$ are either $\log^{O(1)} n$ or $1/\log^{O(1)} n$. The cost of Algorithm 2 is dominated by the repeated calls to Centering.

Number of iterations: By Theorem 4.1 the calls to Centering in Lines 20 and 23 perform $\widetilde{O}(\sqrt{d}\log(1/\delta))$ many iterations in total.

Cost of D^{-1}: Note that $\|\tau^{-1}\delta_\tau\|_{\tau+\infty} = 2\epsilon$, $\|\mathbf{S}^{-1}\delta_s\|_{\tau+\infty} \le \epsilon/2$, and $\|\mathbf{X}^{-1}\delta_x\|_{\tau+\infty} \le \epsilon/2$ by Theorem 4.1. Hence, the weight of the vector $w = \mathbf{S}^{-1-2\alpha}x^{1-2\alpha}$ satisfies $\|\mathbf{W}^{-1}\delta_w\|_\tau = O(\epsilon)$. Note however, that we need D^{-1}.Update to compute the inverse with accuracy $O(\gamma/\log n)$ and hence Theorem 4.5 requires the relative movement of w to be less than $\gamma/\log n$. This can be fixed by splitting the step into $\widetilde{O}(1)$ pieces. Now, Theorem 4.5 shows that the total cost is $\widetilde{O}(d^\omega + \sqrt{d}\log(1/\delta)(d^{\omega-\frac{1}{2}} + d^2)) = \widetilde{O}((d^\omega + d^{2.5})\log(1/\delta))$ where the term d^ω is the initial cost and $d^{\omega-\frac{1}{2}} + d^2$ is the amortized cost per step. Note that the solver runs with accuracy $\widetilde{O}(d^{-1/4})$ so the total time for all calls to D^{-1}.Solve will be $\widetilde{O}(d^3)$.

Cost of D_{Leverage}: By Lemma 4.6, the total movement of the projection matrix is

$$\sum_{k\in[K-1]} \left\| \sqrt{\mathbf{W}^{(k+1)}}\mathbf{A}\Psi^{(k+1)}\mathbf{A}^\top\sqrt{\mathbf{W}^{(k+1)}} - \sqrt{\mathbf{W}^{(k)}}\mathbf{A}\Psi^{(k)}\mathbf{A}^\top\sqrt{\mathbf{W}^{(k)}}\right\|_F = \widetilde{O}(K)$$

where K is the number of steps and $w = \mathbf{S}^{-1-2\alpha}x^{1-2\alpha}$. Then, Theorem 4.4 shows that the total cost is $O(\frac{n}{d}K^2 \cdot d) = O(nK^2) = O(nd\log^2(1/\delta))$.

Cost of $D^{(x)}$ and $D^{(s)}$: By Theorem 4.3, the total cost for $D^{(x)}$ is bounded by $\widetilde{O}(K^2 \max\|\mathbf{X}^{-1}\delta_x\|_2^2 \cdot d + Kn)$ where $\max\|\mathbf{X}^{-1}\delta_x\|_2^2$ is the maximum movement in one step in ℓ_2-norm. Note that $\tau \ge \frac{d}{n}$ and hence $\|\mathbf{X}^{-1}\delta_x\|_2^2 \le \frac{n}{d}\|\mathbf{X}^{-1}\delta_x\|_\tau^2 = \widetilde{O}(\frac{n}{d})$. Hence, we have that the cost is $\widetilde{O}(d\log^2(1/\delta)\cdot\frac{n}{d}\cdot d) = \widetilde{O}(nd)$. The bound for $D^{(s)}$ is the same.

Cost of maintaining $\mathbf{A}^\top\overline{\mathbf{X}}h$ (D_{Gradient}): The cost of maintaining $\mathbf{A}^\top\overline{\mathbf{X}}h$ is exactly equals to d times the number of coordinates changes in $\overline{x}, \overline{s}, \overline{\tau}$. Note that we change the exact x, s, τ by at most around $(1 \pm \gamma/8)$ multiplicative factor in every step. So, the total number of entry changes performed to $\bar{x}, \bar{s}, \bar{\tau}$ is bounded by

$$\widetilde{O}\left(K^2\left(\max\|\mathbf{X}^{-1}\delta_x\|_2^2 + \max\|\mathbf{S}^{-1}\delta_s\|_2^2 + \max\|\tau^{-1}\delta_\tau\|_2^2\right)\right),$$

where max refers to the maximum movement in any step in ℓ_2-norm. As we showed before, all the movement terms are bounded by $\widetilde{O}(\frac{n}{d})$ (see the paragraphs regarding $D^{(x)}$, $D^{(s)}$, and D^{-1}). So in total there are $\widetilde{O}(n\log^2(1/\delta))$ many entry changes we perform to $\overline{x}, \overline{s}, \overline{\tau}$. Hence, the total cost of maintenance is again $\widetilde{O}(nd\log^2(1/\delta))$.

Cost of implementing MaintainFeasibility: By Theorem 4.2 we pay $\widetilde{O}(nd^{0.5} + d^{2.5})$ amortized time per call to MaintainFeasibility. Additionally, we must compute $\widetilde{O}(n/\sqrt{d} + d^{1.5})$ entries of x in each iteration (i.e. call $D^{(x)}$.ComputeExact(i)), which costs $\widetilde{O}(d)$ per call, so we have total cost of $\widetilde{O}(nd + d^3)$ after $\widetilde{O}(\sqrt{d})$ iterations.

Removing the extra $\log(1/\delta)$ term: We note that all the extra $\log(1/\delta)$ terms are due to running the data structures for $\sqrt{d}\log(1/\delta)$ steps. However, we can we reinitialize the data structures every $\sqrt{d}$ iterations. This decreases the K dependence from K^2 to $K\sqrt{d}$.

Independence and Adaptive Adversaries: Randomized data structures often can not handle inputs that depend on outputs of the previous iteration. For example Theorem 4.5 is such a case, where the input to UPDATE is not allowed to depend on the output of any previous calls to UPDATE. In our Algorithm 2 the input to the data-structures inherently depends on their previous output, so here we want to verify that this does not cause any issues.

The data-structures $D^{(x)}$ and $D^{(s)}$ are given by Theorem 4.3, which explicitly states that the data-structure works against adaptive adversaries. This means, that the input is allowed to depend on the output of previous iterations. Likewise, D_{LEVERAGE} works against adaptive adversaries by Theorem 4.4 (provided the input $\Psi^{(\alpha)}_{(\text{safe})}$ are chosen by randomness independent of the randomness chosen for $\Psi^{(\alpha)}$ which is the case by Lemma 4.6). The only issue is with D^{-1} (given by Theorem 4.5), where the input $w, \widetilde{\tau}$ to UPDATE is not allowed to depend on the output of previous calls to UPDATE (however, w and $\widetilde{\tau}$ are allowed to depend on the output of SOLVE by Lemma 4.6). Also note, that the inputs $\bar{w}, b, \delta$ to SOLVE are allowed to depend on any previous output of UPDATE and SOLVE, as Theorem 4.5 only has issues with the inputs w and $\widetilde{\tau}$ to UPDATE. So to show that our algorithm works, we are only left with verifying that the input $w, \widetilde{\tau}$ to UPDATE does not depend on previous results of UPDATE. Let us prove this by induction.

The result of UPDATE is $\Psi^{(\alpha)} \leftarrow D^{-1}.\text{UPDATE}(\overline{S}^{-1-2\alpha}\overline{x}^{1-2\alpha}, \overline{\tau})$, and when executing this line for the very first time, it can obviously not depend on a previous output yet. The matrix $\Psi^{(\alpha)}$ is only used as input to $D_{\text{LEVERAGE}}.\text{QUERY}(\Psi^{(\alpha)}, \Psi^{(\alpha)}_{(\text{safe})})$, but by Theorem 4.4 the output of this procedure does not depend on $\Psi^{(\alpha)}$. Hence $\Psi^{(\alpha)}$ does not affect anything else, which also means the input $\overline{S}^{-1-2\alpha}\overline{x}^{1-2\alpha}$ and $\overline{\tau}$ to the next call of D^{-1}.UPDATE do not depend on previous $\Psi^{(\alpha)}$. □

5 OPEN PROBLEMS

Our main result is a linear program solver that runs in expected $\widetilde{O}(nd + d^3)$ time. For dense constraint matrices $\mathbf{A} \in \mathbb{R}^{n\times d}$ this is optimal among algorithms that do not use fast matrix multiplication, barring a major improvement in solving linear systems. The fastest linear system solvers run in $\widetilde{O}(\text{nnz}(A) + d^\omega)$ time, where $\text{nnz}(A)$ is the number of nonzero entries in $\mathbf{A}$ and ω is the matrix exponent. This leads to two open questions: (i) Can the nd term in our complexity be improved to $\text{nnz}(A)$? (ii) Can the d^3 term be improved to d^ω by exploiting fast matrix multiplication?

A major bottleneck for question (i) is how to detect large entries of the product $\mathbf{A}h$ for $\mathbf{A} \in \mathbb{R}^{n\times d}$, $h \in \mathbb{R}^d$. Currently the complexity of our data structure for that problem (see full version) is $\widetilde{O}(\|\mathbf{G}\mathbf{A}h\|^2 \cdot \varepsilon^{-2} \cdot d)$, which can be interpreted as "d times the number of entries larger than ε". Note that verifying the answer (that is, for a list of indices $I \subset [n]$, check that $(\mathbf{A}h)_i$ is indeed larger than ε) requires the same complexity for dense matrices $\mathbf{A}$. However, for matrices with $z \le d$ entries per row, the verification complexity is just $z \cdot |I|$. This suggests that it might be possible to improve the data structure to run in $\widetilde{O}(\|\mathbf{G}\mathbf{A}h\|^2 \cdot \varepsilon^{-2} \cdot z)$ for matrices $\mathbf{A}$ with z nonzero entries per row. If such a data structure could be found, it would result in a faster linear programming algorithm.

So far question (ii) has only been answered for the case $d = \Omega(n)$ [5, 12] and no progress has been made for the more general case $d = o(n)$, even when allowing for a worse dependency on n than our $\widetilde{O}(nd + d^3)$ bound. So far the best dependency on d is $d^{2.5} \gg d^\omega$ [33, 34], so a first step could be to improve our complexity to $\widetilde{O}(nd + d^{2.5})$ time. Currently the d^3 bottleneck comes from maintaining the feasibility of the primal solution x (Theorem 4.2) so a first step would be to improve the complexity of maintaining the feasibility.

ACKNOWLEDGEMENTS

We thank Sébastien Bubeck, Ofer Dekel, Jerry Li, Ilya Razenshteyn, and Microsoft Research for facilitating conversations between and hosting researchers involved in this collaboration. We thank Vasileios Nakos, Jelani Nelson, and Zhengyu Wang for very helpful discussions about sparse recovery literature. We thank Jonathan Kelner, Richard Peng, and Sam Chiu-wai Wong for helpful conversations. This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 715672. This project was supported in part by NSF awards CCF-1749609, CCF-1740551, DMS-1839116, CCF-1844855, 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

[1] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. 2019. Iterative Refinement for lp-norm Regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, https://arxiv.org/pdf/1901.06764.pdf, 1405–1424.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2013. Spectral Sparsification in Dynamic Graph Streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*. 1–10.

[3] Kurt M. Anstreicher. 1996. Volumetric path following algorithms for linear programming. *Math. Program.* 76 (1996), 245–263.

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

[5] Jan van den Brand. 2020. A Deterministic Linear Program Solver in Current Matrix Multiplication Time. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. https://arxiv.org/pdf/1910.11957.pdf.

[6] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. 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.

[7] Emmanuel J Candes, Justin K Romberg, and Terence Tao. 2006. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics* 59, 8 (2006), 1207–1223.

[8] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming (ICALP)*. Springer, 693–703.

[9] Kenneth L Clarkson and David P Woodruff. 2013. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (STOC)*. ACM, https://arxiv.org/pdf/1207.6365.pdf, 81–90.

[10] Michael B. Cohen. 2016. Nearly Tight Oblivious Subspace Embeddings by Trace Inequalities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 278–287.

[11] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. 2015. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS)*. ACM, https://arxiv.org/pdf/1408.5099.pdf, 181–190.

[12] Michael B Cohen, Yin Tat Lee, and Zhao Song. 2019. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. ACM, https://arxiv.org/pdf/1810.07896.pdf, 938–942.

[13] Michael B. Cohen and Richard Peng. 2015. ℓ_p Row Sampling by Lewis Weights. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, (STOC)*. https://arxiv.org/pdf/1412.0588.pdf, 183–192.

[14] Don Coppersmith and Shmuel Winograd. 1982. On the asymptotic complexity of matrix multiplication. *SIAM J. Comput.* 11, 3 (1982), 472–492.

[15] Don Coppersmith and Shmuel Winograd. 1990. Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.* 9, 3 (1990), 251–280.

[16] Graham Cormode and Marios Hadjieleftheriou. 2009. Finding the frequent items in streams of data. *Commun. ACM* 52, 10 (2009), 97–105.

[17] Graham Cormode and Shan Muthukrishnan. 2004. An improved data stream summary: the count-min sketch and its applications. In *Proceedings of the Annual Conference on Foundations of Software Technology and Theoretical Computer Science*.

[18] George B Dantzig. 1951. Maximization of a linear function of variables subject to linear inequalities. *New York* (1951).

[19] David L. Donoho. 2006. Compressed sensing. *IEEE Trans. Information Theory* 52, 4 (2006), 1289–1306.

[20] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. 2019. Fully dynamic spectral vertex sparsifiers and applications. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. https://arxiv.org/pdf/1906.10530.pdf, 914–925.

[21] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *ISSAC*.

[22] Francois Le Gall and Florent Urrutia. 2018. 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)*. https://arxiv.org/pdf/1708.05622.pdf, 1029–1046.

[23] Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. 2010. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing 2012 (A preliminary version of this paper appears in STOC 2010)* 41, 2 (2010), 436–453.

[24] Clovis C Gonzaga. 1992. Path-following methods for linear programming. *SIAM review* 34, 2 (1992), 167–224.

[25] Haitham Al Hassanieh. 2016. *Sparse recovery and Fourier sampling*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[26] Daniel M Kane, Jelani Nelson, Ely Porat, and David P Woodruff. 2011. Fast moment estimation in data streams in optimal space. In *Proceedings of the forty-third annual ACM symposium on Theory of computing (STOC)*. ACM, https://arxiv.org/pdf/1007.4191.pdf, 745–754.

[27] Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. 2017. Single Pass Spectral Sparsification in Dynamic Streams. In *SIAM J. Comput.*, Vol. 46(1). https://arxiv.org/pdf/1407.1289.pdf, 456–477.

[28] Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. 2020. Dynamic Streaming Spectral Sparsification in Nearly Linear Time and Space. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. https://arxiv.org/pdf/1903.12150.pdf.

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

[30] Leonid G Khachiyan. 1980. Polynomial algorithms in linear programming. *U. S. S. R. Comput. Math. and Math. Phys.* 20, 1 (1980), 53–72.

[31] Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. 2016. Heavy hitters via cluster-preserving clustering. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, https://arxiv.org/pdf/1604.01357, 61–70.

[32] Yin Tat Lee. 2016. *Faster algorithms for convex and combinatorial optimization*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[33] Yin Tat Lee and Aaron Sidford. 2014. Path-Finding Methods for Linear Programming : Solving Linear Programs in $\widetilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. https://arxiv.org/pdf/1312.6677.pdf, https://arxiv.org/pdf/1312.6713.pdf, 424–433.

[34] Yin Tat Lee and Aaron Sidford. 2015. Efficient Inverse Maintenance and Faster Algorithms for Linear Programming. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. https://arxiv.org/pdf/1503.01752.pdf, 230–249.

[35] Yin Tat Lee and Aaron Sidford. 2019. Solving Linear Programs with $\sqrt{rank}$ Linear System Solves. In *arXiv preprint*. http://arxiv.org/pdf/1910.08033.pdf. Journal submission.

[36] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. 2015. A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*.

[37] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. 2019. Solving Empirical Risk Minimization in the Current Matrix Multiplication Time. In *Conference on Learning Theory (COLT)*. https://arxiv.org/pdf/1905.04447.pdf, 2140–2157.

[38] D. Lewis. 1978. Finite dimensional subspaces of L_p. *Studia Mathematica* 63, 2 (1978), 207–212. http://eudml.org/doc/218208

[39] Mu Li, Gary L. Miller, and Richard Peng. 2013. Iterative Row Sampling. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. https://arxiv.org/pdf/1211.2713.pdf, 127–136.

[40] Michael W. Mahoney. 2011. Randomized Algorithms for Matrices and Data. *Foundations and Trends in Machine Learning* 3, 2 (2011), 123–224.

[41] Sanjay. Mehrotra. 1992. On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization* 2, 4 (1992), 575–601.

[42] Xiangrui Meng and Michael W Mahoney. 2013. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (STOC)*. ACM, https://arxiv.org/pdf/1210.3135.pdf, 91–100.

[43] Vasileios Nakos and Zhao Song. 2019. Stronger L2/L2 Compressed Sensing; Without Iterating. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*. https://arxiv.org/pdf/1903.02742.pdf.

[44] Jelani Nelson and Huy L Nguyên. 2013. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. https://arxiv.org/pdf/1211.1002.pdf, 117–126.

[45] Jelani Nelson, Huy L Nguyên, and David P Woodruff. 2014. On deterministic sketching and streaming for sparse recovery and norm estimation. In *Linear Algebra and its Applications*, Vol. 441. https://arxiv.org/pdf/1206.5725.pdf, 152–167.

[46] Jelani Nelson and David P. Woodruff. 2014. Personal communication. . (2014).

[47] Yurii Nesterov and Arkadii Semenovich Nemirovskii. 1994. *Interior-point polynomial algorithms in convex programming*. Vol. 13. Society for Industrial and Applied Mathematics.

[48] Yu Nesterov and Arkadi Nemirovskiy. 1989. *Self-concordant functions and polynomial-time methods in convex programming*. USSR Academy of Sciences, Central Economic & Mathematic Institute.

[49] Yu Nesterov and A Nemirovsky. 1991. Acceleration and parallelization of the path-following interior point method for a linearly constrained convex quadratic problem. *SIAM Journal on Optimization* 1, 4 (1991), 548–564.

[50] Rasmus Pagh. 2013. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)* 5, 3 (2013), 9.

[51] Eric C Price. 2013. *Sparse recovery and Fourier sampling*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[52] James Renegar. 1988. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming* 40, 1-3 (1988), 59–93.

[53] Piotr Sankowski. 2004. Dynamic Transitive Closure via Dynamic Matrix Inverse (Extended Abstract). In *45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 509–517.

[54] Aaron Daniel Sidford. 2015. *Iterative methods, combinatorial optimization, and linear programming beyond the universal barrier*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[55] Zhao Song. 2019. *Matrix Theory: Optimization, Concentration and Algorithms*. Ph.D. Dissertation. The University of Texas at Austin.

[56] Zhao Song, David P Woodruff, and Peilin Zhong. 2019. Relative Error Tensor Low Rank Approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. https://arxiv.org/pdf/1704.08246.pdf.

[57] Daniel A Spielman and Nikhil Srivastava. 2011. Graph sparsification by effective resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926.

[58] Andrew James Stothers. 2010. On the complexity of matrix multiplication. (2010).

[59] Volker Strassen. 1969. Gaussian elimination is not optimal. *Numerische mathematik* 13, 4 (1969), 354–356.

[60] Volker Strassen. 1986. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In *27th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 49–54.

[61] Pravin M. Vaidya. 1989. A New Algorithm for Minimizing Convex Functions over Convex Sets (Extended Abstract). In *30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 338–343.

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

[63] Pravin M. Vaidya. 1990. Reducing the Parallel Complexity of Certain Linear Programming Problems (Extended Abstract). In *31st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 583–589.

[64] Pravin M Vaidya and David S Atkinson. 1993. A technique for bounding the number of iterations in path following algorithms. *Complexity in Numerical Optimization* (1993), 462–489.

[65] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. 2020. Solving Tall Dense Linear Programs in Nearly Linear Time. *CoRR* abs/2002.02304 (2020).

[66] Virginia Vassilevska Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*. ACM, 887–898.

[67] David P. Woodruff. 2014. Sketching as a Tool for Numerical Linear Algebra. *Foundations and Trends in Theoretical Computer Science* 10, 1-2 (2014), 1–157.

[68] Yinyu Ye. 2011. *Interior point algorithms: theory and analysis*. Vol. 44. John Wiley & Sons.

[69] Yinyu Ye, Michael J Todd, and Shinji Mizuno. 1994. An $O(\sqrt{nL})$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research* 19, 1 (1994), 53–67.