Minimum Cost Flows, MDPs, and ℓ_1 -Regression in Nearly Linear Time for Dense Instances

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

Yin Tat Lee
University of Washington and
MSR Redmond
USA
yintat@uw.edu

Yang P. Liu
Stanford University
USA
yangpliu@stanford.edu

Thatchaphol Saranurak
University of Michigan
USA
thsa@umich.edu

Aaron Sidford
Stanford University
USA
sidford@stanford.edu

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

Di Wang
Google Research
USA
wadi@google.com

ABSTRACT

In this paper we provide new randomized algorithms with improved runtimes for solving linear programs with two-sided constraints. In the special case of the minimum cost flow problem on n -vertex m -edge graphs with integer polynomially-bounded costs and capacities we obtain a randomized method which solves the problem in $\tilde{O}(m + n^{1.5})$ time. This improves upon the previous best runtime of $\tilde{O}(m\sqrt{n})$ [Lee-Sidford'14] and, in the special case of unit-capacity maximum flow, improves upon the previous best runtimes of $m^{4/3+o(1)}$ [Liu-Sidford'20, Kathuria'20] and $\tilde{O}(m\sqrt{n})$ [Lee-Sidford'14] for sufficiently dense graphs.

In the case of ℓ_1 -regression in a matrix with n -columns and m -rows we obtain a randomized method which computes an ϵ -approximate solution in $\tilde{O}(mn + n^{2.5})$ time. This yields a randomized method which computes an ϵ -optimal policy of a discounted Markov Decision Process with S states and A actions per state in time $\tilde{O}(S^2A + S^{2.5})$. These methods improve upon the previous best runtimes of methods which depend polylogarithmically on problem parameters, which were $\tilde{O}(mn^{1.5})$ [Lee-Sidford'15] and $\tilde{O}(S^{2.5}A)$ [Lee-Sidford'14, Sidford-Wang-Wu-Ye'18] respectively.

To obtain this result we introduce two new algorithmic tools of possible independent interest. First, we design a new general interior point method for solving linear programs with two sided constraints which combines techniques from [Lee-Song-Zhang'19, Brand et al.'20] to obtain a robust stochastic method with iteration count nearly the square root of the smaller dimension. Second,

to implement this method we provide dynamic data structures for efficiently maintaining approximations to variants of Lewis-weights, a fundamental importance measure for matrices which generalize leverage scores and effective resistances.

CCS CONCEPTS

• Theory of computation → Linear programming; Network flows.

KEYWORDS

Linear Program, Data Structure, Interior Point Method

ACM Reference Format:

Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2021. Minimum Cost Flows, MDPs, and ℓ_1 -Regression in Nearly Linear Time for Dense Instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21)*, June 21–25, 2021, Virtual, Italy. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3406325.3451108>

1 INTRODUCTION

We consider solving linear programs expressed in the following primal/dual form:

$$(P) \quad \min_{\substack{x \in \mathbb{R}^m: A^T x = b \\ \ell_i \leq x_i \leq u_i \ \forall i \in [m]}} c^T x \quad \text{and} \quad (1)$$

$$(D) \quad \max_{\substack{y \in \mathbb{R}^n, s \in \mathbb{R}^m \\ Ay + s = c}} b^T y + \sum_{i \in [n]} \min(\ell_i s_i, u_i s_i).$$

where $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ and each $\ell_i \leq u_i \in \mathbb{R}$.

Equation (1) naturally encompasses prominent continuous and combinatorial optimization problems. When $\ell_i = 0$ and $u_i = \infty$,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
STOC '21, June 21–25, 2021, Virtual, Italy
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8053-9/21/06.
<https://doi.org/10.1145/3406325.3451108>

The full version of this paper is available as [10] at <https://arxiv.org/abs/2101.05719>.

(1) corresponds to the standard primal-dual formulation of linear program. In the case when A is the incidence matrix of a graph, i.e. $b = \vec{0}$, and $\ell_i = 0$ for all $i \in [m]$, (1) corresponds to the problem of computing a minimum cost circulation in a directed graph with linear costs c and edge capacities given by u . Further, in the case when each $u_i = 1$, $\ell_i = -1$, and $b = 0$, (1) corresponds to solving ℓ_1 regression and can be used to solve Markov Decision Processes (MDPs) [82].

Recent advances in interior point methods (IPMs), a prominent class of continuous optimization methods, and data structures have led to nearly linear runtimes for solving fundamental classes of (1) to high precision with only a polylogarithmic dependence on problem parameters. In [12] an $\tilde{O}(mn + n^{2.5})$ time randomized method was obtained for solving (1) when $\ell_i = 0$ and $u_i = \infty$ for all $i \in [m]$, i.e., when the problem is in standard form. Further, in [11] a randomized method was obtained for solving minimum cost perfect matching in bipartite graphs in time $\tilde{O}(m + n^{1.5})$, i.e. when the problem is in standard form and A is the incidence matrix of a bipartite graph.

Unfortunately, though it is well known that all linear programs can be written in standard form, naively transforming (1) to standard form can increase the dimension of the problem, i.e. turn n to $\Omega(m)$. This issue prevents the application of the recent advances in [11, 12] to (1) when both u_i and ℓ_i are bounded. Consequently, despite extensive study, obtaining nearly linear running times for solving (1), maximum flow, minimum cost flow, ℓ_1 regression, and Markov Decision Processes to high-precision with a polylogarithmic dependence on problem parameters in nearly linear time in high-dimensional dense instances has been elusive.

The issue of losing density when reformulating (1) in standard form is a known difficulty in obtaining improved runtimes from continuous optimization methods. It arose in [51, 53] and was addressed and leveraged to obtain improved randomized runtimes for minimum cost flow, ℓ_1 -regression [52], and MDPs [51, 52, 82]. These methods provide IPMs which work directly with (1). The IPMs reweight constraints based on variations of Lewis weights [23, 53], a natural notion of row importance for matrices which generalizes leverage scores, and implement and apply the corresponding methods efficiently.

Lewis weight reweighting schemes were also key to the aforementioned nearly linear time algorithms for solving linear programs in standard form [12] and computing minimum-cost bipartite perfect matchings [11] on dense instances. Unfortunately, a key technique applied by these results is that when (1) is in standard form (i.e. $\ell_i = 0$ and $u_i = \infty$ for all $i \in [n]$) it is possible to leverage the primal-dual structure of the problem to rewrite the optimality conditions of [53] in terms of leverage scores, a simple special case of Lewis weights. Leveraging this structure, these papers design IPMs and data-structures for efficiently leveraging and manipulating leverage scores towards achieving their runtimes. Unfortunately, in the case that both ℓ_i and u_i are bounded, this same technique doesn't directly apply (see Section 3.1).

In this paper, we show how to overcome this difficulty and directly obtain nearly linear time algorithms for solving (1), minimum cost flow, ℓ_1 -regression [52], and MDPs, in on moderately dense instances. First, we provide a new IPM directly tailored to solving (1)

which directly works with a variant of Lewis weights. Our method applies techniques in robustifying IPMs [8, 11, 12, 21, 37, 54, 84] and in particular techniques from [54] on robust IPMs for solving empirical risk minimization problems, to a variant of the Lewis-weight based optimality conditions suggested by [53]. Further, the method applies and generalizes sampling and analysis techniques from [11]. The combination of these techniques and interaction with Lewis weights induces a number of technical challenges. Interestingly, ultimately, our analysis leverages higher order smoothness properties of barriers for the intervals $\{x : \ell_i \leq x \leq u_i\}$.

As with many recent results [8, 11, 12, 21, 37, 54, 84], this new IPM reduces the challenge of solving (1) to solving a sequence of data structure problems and appropriately initializing and rounding the iterates of the IPM. One particularly challenging aspect is that our IPM requires that regularized ℓ_p -Lewis weights be maintained efficiently throughout the algorithm. Though previous work [11, 12] provided various results on dynamically maintaining leverage scores, i.e. the special case when $p = 2$, and there are known algorithms for computing Lewis weights efficiently using leverage scores, the complexity of such a naïve approach is unclear. We overcome this issue by providing both a more careful reduction from Lewis weight maintenance to leverage score maintenance and a direct reduction from leverage score maintenance to detecting large rows in a dynamically changing matrix, what we call a heavy hitter data structure [11, 12].

By considering specialized heavy hitter data structures for the various problems we consider, providing additional data structure as needed, and carefully applying the resulting IPM we obtain the main runtime results of this paper. In the special case of solving linear programs in standard form we provide new sampling data structure for matrices that allow us to improve upon the runtime of [12] in certain settings.

1.1 Our Results

Here we provide the main results of this paper, including new nearly linear time algorithms for solving (1) in different settings.

Linear Programming with Two-sided Constraints. Our main contributions are efficient algorithms for solving primal/dual LPs with two-sided constraints (1) via IPMs. In the general case, where there is no additional structure on A we obtain the following result.

THEOREM 1.1 (PRIMAL SOLUTION FOR GENERAL LPs). *Let $A \in \mathbb{R}^{m \times n}$, $c, \ell, u \in \mathbb{R}^m$, and $b \in \mathbb{R}^n$. Assume that there is a point x satisfying $A^\top x = b$ and $\ell_i \leq x_i \leq u_i$ for all $i \in [m]$. Let $W \stackrel{\text{def}}{=} \max(\|c\|_\infty, \|A\|_\infty, \|b\|_\infty, \|u\|_\infty, \|\ell\|_\infty, \frac{\max_i(u_i - \ell_i)}{\min_i(u_i - \ell_i)})$. For any $\delta > 0$ there is an algorithm running in time $\tilde{O}((mn + n^{2.5}) \log(W/\delta))$ that with high probability (w.h.p.) which computes a vector $x^{(\text{final})}$ satisfying*

$$\|A^\top x^{(\text{final})} - b\|_\infty \leq \delta \quad \text{and} \quad \ell_i \leq x_i^{(\text{final})} \leq u_i \quad \forall i \quad \text{and} \\ c^\top x^{(\text{final})} \leq \min_{\substack{A^\top x = b \\ \ell_i \leq x_i \leq u_i \forall i}} c^\top x + \delta.$$

The prior best result runtimes for achieving guarantees comparable to Theorem 1.1 were $\tilde{O}(m^{\max\{\omega, 2+1/18\}})$ [37] ($\omega \approx 2.37286$ [3, 32, 95] is the exponent of current matrix multiplication) and

$\tilde{O}((\text{nnz}(\mathbf{A}) + n^2)\sqrt{n})$ [52]. Whenever $\mathbf{A}$ is tall and dense, i.e. $m \geq n^{1.5}$ and $\text{nnz}(\mathbf{A}) = \Omega(mn)$, this corresponds to a nearly linear time algorithm for solving (1) to high precision.

Interestingly, even in the special case when $\ell_i = 0$ and $u_i = \infty$ for all $i \in [n]$ this improves upon the previous best runtimes of $\tilde{O}(m^\omega)$ and $\tilde{O}((\text{nnz}(\mathbf{A}) + \sqrt{n})\sqrt{n})$ mentioned above, as well as $\tilde{O}(mn + n^3)$ [12]. In particular, we improve upon [12] by improving the additive $\tilde{O}(n^3)$ term to a $\tilde{O}(n^{2.5})$. This improvement stems from IPM sampling techniques of [11] (as refined in this paper) and new data structures introduced in this paper.

ℓ_1 -Regression and MDPs. Theorem 1.1 immediately yields improved runtimes for additional prominent optimization problems. For instance, when $b = \vec{0}$ and $\ell_i = -1$ and $u_i = 1$ for all $i \in [n]$, the dual formulation in (1) encodes to ℓ_1 -regression. Consequently, we obtain the following result.

THEOREM 1.2 (DUAL SOLUTION FOR GENERAL LPs, ℓ_1 -REGRESSION). *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{c} \in \mathbb{R}^m$, and $\delta > 0$. There is an algorithm running in time $\tilde{O}((mn + n^{2.5}) \log(W/\delta))$ for $W \stackrel{\text{def}}{=} \max(\|\mathbf{c}\|_\infty, \|\mathbf{A}\|_\infty)$ which w.h.p. computes a vector $\mathbf{z} \in \mathbb{R}^n$ such that*

$$\|\mathbf{A}\mathbf{z} + \mathbf{c}\|_1 \leq \min_{\mathbf{z} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{z} + \mathbf{c}\|_1 + \delta.$$

As with Theorem 1.1 this runtime is nearly linear whenever $\mathbf{A}$ is tall and dense, i.e. $\text{nnz}(\mathbf{A}) = \Omega(mn)$ and $m \geq n^{1.5}$. Further, as with Theorem 1.1, in the regime of high accuracy algorithms, it improves upon the results of [8, 21, 37, 52, 54, 84] mentioned above.

Further, leveraging a reduction from [82], in certain settings this yields to improved running times for solving MDPs, a fundamental mathematical model for reasoning about uncertainty. An instance of the discounted Markov Decision Process (DMDP) is specified by a tuple $(S, \mathbf{A}, P, r, \gamma)$ where S is the state space, $\mathbf{A}$ is the action space, P describes state-action-state transition probabilities, r describes state-action rewards in range $[-M, M]$, and $\gamma \in (0, 1)$ is a discount factor. The goal is to compute a policy that maps each state to an action that (approximately) maximizes the reward in a certain sense. (See the full version [10] for a more precise definition of the problem) We obtain the following result.

THEOREM 1.3 (DISCOUNTED MDP). *Given a DMDP $(S, \mathbf{A}, P, r, \gamma)$, there is an algorithm that, with high probability, computes a ϵ -optimal policy π in $\tilde{O}((|S|^2|\mathbf{A}| + |S|^{2.5}) \log(\frac{M}{(1-\gamma)\epsilon}))$ time.*

Since the input size of the state-action-state transition is at least $\Omega(|S|^2|\mathbf{A}|)$, this algorithm runs in nearly-linear time whenever $|\mathbf{A}| \geq \sqrt{|S|}$. Also, this result directly improves upon the previous algorithm with running time $\tilde{O}((|S|^{2.5}|\mathbf{A}|) \log(\frac{M}{(1-\gamma)\epsilon}))$ [51, 82].

Minimum Cost Flow. In the minimum cost flow problem, we are given a connected directed graph $G = (V, E, u, c)$ with edges capacities $u \in \mathbb{R}_{\geq 0}^E$ and costs $c \in \mathbb{R}^E$. We call $x \in \mathbb{R}^E$ an s - t flow for $s, t \in V$ if $x_e \in [0, u_e]$ for all $e \in E$ and for each vertex $v \notin \{s, t\}$ the amount of flow entering v , i.e. $\sum_{e=(a,v) \in E} x_e$ equals the amount of flow leaving v , i.e. $\sum_{e=(v,b) \in E} x_e$. The value of s - t flow is the amount of flow leaving s (or equivalently, entering t). The maximum flow problem is to compute an s - t flow of maximum value. In the minimum cost maximum flow problem, the goal is to compute a maximum s - t flow of minimum cost, $\sum_{e \in E} c_e x_e = c^\top x$.

Such problems can be expressed in the form of (1) by taking $\mathbf{A}$ as the graph incidence matrix, letting ℓ_i and u_i denote edge capacities, and choosing b appropriately. Unfortunately, directly applying Theorem 1.1 would yield a large $\tilde{O}((mn + n^{2.5}) \log W)$ runtime which does not improve upon previous results. However, applying the techniques developed in this paper along with Laplacian system solvers [20, 44–48, 50, 75, 86] and data structure ideas from [11] specific to graphs we prove the following theorem.

THEOREM 1.4 (MIN COST FLOW). *There is an algorithm that, given a n -vertex, m -edge, directed graph $G = (V, E, u, c)$ integral edge capacities $u \in \mathbb{Z}_{\geq 0}^E$ and costs $c \in \mathbb{Z}^E$, with high probability, computes a minimum cost maximum flow in time*

$$\tilde{O}(m \log(\|u\|_\infty \|c\|_\infty) + n^{1.5} \log^2(\|u\|_\infty \|c\|_\infty)).$$

Efficiently solving the minimum cost flow problem to high accuracy gives an algorithm for maximum flow in the same runtime on weighted graphs, and we give the reduction formally in the full version. Using standard capacity scaling methods [2], we can improve the $\log^2 W$ dependence to $\log W$ on the $n^{1.5}$ term in this case, where $W = \max(\|u\|_\infty)$. The previous best runtimes for min-cost flow were $m^{4/3+o(1)} \log W$ in the case of unit capacity graphs [5] and $\tilde{O}(m\sqrt{n} \log^{O(1)} W)$ [53]. Our algorithm improves on these for dense graphs, and in particular runs in nearly linear time for $m \geq n^{1.5}$.

1.2 Related Work

The problems we consider in this paper, e.g. linear programming, dynamic data structures, minimum cost flow, ℓ_1 -regression, MDPs, are all incredibly well-studied. Each has an extensive history and numerous results. Here, we provide just a brief summary of the results and tools most directly related to this paper.

Linear Programming IPMs. There has been significant work towards the design of IPMs for linear programming, starting from [41, 77]. More recently, there have been IPM-based runtime improvements to linear programming by decreasing the number of iterations [51], proving that maintaining approximate primal/dual solutions suffice [8, 11, 12, 21, 37, 54, 84], and using data structures to decrease iteration costs [8, 11, 21, 37, 52, 54, 84]. There is a related line of work on strongly polynomial linear programming [24, 25, 64, 65, 90], where the goal is to achieve exact solutions with improved parameter dependencies, instead of high accuracy solutions as we do here. In [57] it was shown how to implement IPMs in the semi-streaming model.

ℓ_1 -Regression. There are several algorithms for ℓ_1 -regression in both the low accuracy ($\text{poly}(\epsilon^{-1})$ dependence) [14–17, 29, 70, 83, 94, 96, 97] and high accuracy ($\log(1/\epsilon)$ dependence) [21, 52] regimes. The state of the art results in the high-accuracy regime are [52] which achieves a $\tilde{O}((\text{nnz}(\mathbf{A}) + n^2)\sqrt{n} \log(1/\epsilon))$ runtime, and [37] which achieves a $\tilde{O}(n^{\max\{\omega, 2+1/18\}})$ runtime, improving on $\tilde{O}(n^{\max\{\omega, 2+1/6\}})$ [21].

Minimum Cost Flow. There has been significant work towards combinatorial algorithms for the minimum cost flow problem [30, 31, 34, 72, 73, 87] in both the strongly polynomial and weakly polynomial regimes. Daitch and Spielman [26] showed that one can

Table 1: The summary of the results for the maximum flow problem. W denotes the maximum capacity. Subpolynomial ($n^{o(1)}$) terms are hidden. For simplicity, we only list exact algorithms which yielded polynomial improvements.

Year	Authors	References	Time	
			Sparse	Dense
1970	Dinitz	[28]	mn	
1997	Goldberg, Rao	[33]	$m^{3/2} \log W$	$mn^{2/3} \log W$
2013	Madry	[61, 62]	$m^{10/7} W^{1/7}$	
2013	Lee and Sidford	[51]		$m\sqrt{n} \log^{O(1)} W$
2020	Liu and Sidford	[59]	$m^{11/8} W^{1/4}$	
2020	Liu and Sidford; Kathuria	[42, 58]	$m^{4/3} W^{1/3}$	
2020		This paper		$(m + n^{1.5}) \log W$

Table 2: The summary of the results for the minimum-cost flow problem. Subpolynomial ($n^{o(1)}$) terms are hidden. W denotes the maximum absolute value of capacities and costs. For simplicity, we only list exact algorithms which yielded polynomial improvements. Results marked with an asterisk work only on unit-capacity graph.

Year	Authors	References	Time	
			Sparse	Dense
1972	Edmonds and Karp	[30]	$m^2 \log W$	
1987	Goldberg and Tarjan	[34]		$mn \log W$
2008	Daitch and Spielman	[26]	$m^{3/2} \log^2 W$	
2013	Lee and Sidford	[51]		$m\sqrt{n} \log^{O(1)} W$
2017	* Cohen, Madry, Sankowski, Vladu	[22]	$m^{10/7} \log W$	
2020	* Axiotis, Madry, Vladu	[5]	$m^{4/3} \log W$	
2020		This paper		$m \log W + n^{1.5} \log^2 W$

use a Laplacian system solver to implement steps of an IPM to achieve a more efficient mincost flow algorithm with logarithmic capacity dependence. Since then, runtime improvements have been achieved by reducing the number of iterations of a general IPM to $\tilde{O}(\sqrt{n})$ [51] and to $O(m^{1/3+o(1)})$ [5] for graphs with unit-capacity, improving on $\tilde{O}(m^{10/7} \log W)$ [22].

Markov Decision Process (MDP). We focus on solving *discounted* MDPs.¹ Previous works in the high precision regime (i.e. logarithmic dependency on error) includes [56, 82, 88] with the best running time of $\tilde{O}((|S|^2|A| + \frac{|S||A|}{(1-\gamma)^3}) \log(\frac{M}{\epsilon}))$. Strongly polynomial time exact algorithms are also known [80, 98, 99].

For algorithms that depend logarithmically on one minus the discount factor γ , the algorithm by [51] implies a running time of $\tilde{O}((|S|^{2.5}|A|) \log(\frac{M}{(1-\gamma)\epsilon}))$ (shown in [82]). Our result in Theorem 1.3 directly improves this algorithm.

There is another line of work focusing on fast algorithms in the low precision regime with polynomial dependency on the error parameter [1, 6, 43, 55, 81, 82, 91–93]. This setting is not directly comparable to our result.

Dynamic Data Structures. IPMs reduce the task of solving linear programs to the task of solving linear systems. Instead of solving these linear systems from scratch in each iteration, these iterative algorithms can be sped up by using data structures that efficiently

maintain the matrix inverse corresponding to the linear system [12, 21, 37, 41, 52, 54, 71, 84, 89]. There also exist data structures to efficiently maintain the solution of the linear system instead of (or in addition to) the corresponding matrix [8, 9, 13, 37, 78]. Recently there have also been data structures developed that are able to efficiently maintain an approximation of the primal/dual solution [11, 12, 37, 54]. For graph applications these algorithms are based on the dynamic expander decomposition technique [7, 35, 68, 79]. For general LPs, the data structure for maintaining approximate primal/dual solutions are based on heavy hitters and sketching [19, 38–40, 49, 66, 67, 69, 74, 76].

1.3 Organization

We give the preliminaries in Section 2. Our overview is in Section 3, split into overview of the IPM in Section 3.1 and overview of the data structures in Section 3.2. A detailed presentation and proof of our IPM and our regularized Lewis weight maintenance data structure are available in the full version [10]. There we also show in detail how to use our data structure to implement the IPM.

2 PRELIMINARIES

We follow similar notation as in [11]. We let $[n] \stackrel{\text{def}}{=} \{1, 2, \dots, n\}$ and $\vec{e}_i$ denote the i -th standard unit vector. We use $\tilde{O}(\cdot)$ notation to hide $(\log \log W)^{O(1)}$, $\log \epsilon^{-1}$, and $(\log n)^{O(1)}$ factors, where W typically denotes the largest absolute value used for specifying any value in the problem (e.g. demands and edge weights) and n denotes the number of nodes. When we write *with high probability* (or w.h.p),

¹Other variants of this problems includes deterministic MDPs (equivalent to the min-mean cycle problem) [11, 18, 27, 60] and average-reward MDPs [4, 36, 63].

we mean with probability $1 - n^c$ for any constant $c > 0$. We write $\mathbf{1}_{\text{condition}}$ for the indicator variable, which is 1 if the condition is true and 0 otherwise.

Diagonal Matrices. Given a vector $v \in \mathbb{R}^d$ for some d , we write $\text{Diag}(v)$ for the $d \times d$ diagonal matrix with $\text{Diag}(v)_{i,i} = v_i$. For a vector v we also write $\mathbf{V}$ for the diagonal matrix $\text{Diag}(v)$ when clear from context.

Matrix and Vector operations. Given vectors $u, v \in \mathbb{R}^d$ for some d , we perform arithmetic operations $\cdot, +, -, /$, $\sqrt{\cdot}$ element-wise. For example $(u \cdot v)_i = u_i \cdot v_i$ or $(\sqrt{v})_i = \sqrt{v_i}$. For the inner product we write $\langle u, v \rangle$ and $u^\top v$ instead. For a vector $v \in \mathbb{R}^d$ and a scalar $\alpha \in \mathbb{R}$ we let $(\alpha v)_i = \alpha v_i$ and $(v + \alpha)_i = v_i + \alpha$.

For symmetric matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ we write $\mathbf{A} \leq \mathbf{B}$ to indicate that $x^\top \mathbf{A} x \leq x^\top \mathbf{B} x$ for all $x \in \mathbb{R}^n$ and define $>, <$, and $\geq$ analogously. We call any matrix (not necessarily symmetric) *non-degenerate* if its rows are all non-zero and it has full column rank.

We use $u \approx_\epsilon v$ to denote that $\exp(-\epsilon)v \leq u \leq \exp(\epsilon)v$ entrywise and $\mathbf{A} \approx_\epsilon \mathbf{B}$ to denote that $\exp(-\epsilon)\mathbf{B} \leq \mathbf{A} \leq \exp(\epsilon)\mathbf{B}$. Note that this notation implies $u \approx_\epsilon v \approx_\delta w \Rightarrow u \approx_{\epsilon+\delta} w$, and $u \approx_\epsilon v \Rightarrow u^\alpha \approx_{\epsilon \cdot |\alpha|} v^\alpha$ for any $\alpha \in \mathbb{R}$.

For any matrix $\mathbf{A}$ with real entries, let $\text{nnz}(\mathbf{A})$ denote the number of non-zero entries in $\mathbf{A}$ and $\text{nnz}(a_i)$ be the number of non-zero entries in the i -th row of $\mathbf{A}$.

Leverage Scores and Lewis-Weights. For any non-degenerate matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ we let $\sigma(\mathbf{A}) \in \mathbb{R}^m$ with $\sigma(\mathbf{A})_i \stackrel{\text{def}}{=} (\mathbf{A}(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top)_{i,i}$ denote $\mathbf{A}$'s *leverage scores*. For $p \in (0, \infty)$ and non-degenerate matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ we define the ℓ_p *Lewis weight* as the solution $w \in \mathbb{R}_{>0}^m$ to the equation $w = \sigma(\mathbf{W}^{\frac{1}{2} - \frac{1}{p}} \mathbf{A})$, where $\mathbf{W} = \text{diag}(w)$. We use a regularized Lewis weight in our algorithms:

Definition 2.1 (Regularized Lewis weights for c). For $p = 1 - \frac{1}{4 \log(4m/n)}$, and $v \in \mathbb{R}_{>0}^m$ define the (v -regularized) ℓ_p -Lewis weights as the solution to $w = \sigma(\mathbf{W}^{\frac{1}{2} - \frac{1}{p}} \mathbf{A}) + v$.

Norms. We write $\|\cdot\|_p$ for the ℓ_p -norm, i.e. $\|v\|_p := (\sum_i |v_i|^p)^{1/p}$, $\|v\|_\infty = \max_i |v_i|$ and $\|v\|_0$ being the number of non-zero entries of v . For a positive definite matrix $\mathbf{M}$ we define $\|v\|_{\mathbf{M}} = \sqrt{v^\top \mathbf{M} v}$. For a vector τ we define $\|v\|_\tau := (\sum_i \tau_i v_i^2)^{1/2}$ and $\|v\|_{\tau+\infty} := \|v\|_\infty + C \log(4m/n) \|v\|_\tau$ for a large constant C , where $m \geq n$ are the dimensions of the constraint matrix of the linear program.

3 OVERVIEW OF APPROACH

In this section, we give an overview of the major aspects of our algorithm, including the path following IPMs (Section 3.1), the data structures necessary for its implementation (Section 3.2), and how to combine them for our applications (Section 3.3).

3.1 IPM

As context and motivation for our method, we start by discussing the IPM of [53]. For matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$, vectors $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, and lower and upper bounds $\ell, u \in \mathbb{R}^m$, this IPM solves linear programs of the form

$$\min_{\substack{\mathbf{A}^\top x = b \\ \ell \leq x_i \leq u_i \forall i \in [m]}} c^\top x$$

to high accuracy. The runtime of this method is dominated by the runtime needed for solving $\tilde{O}(\sqrt{n})$ linear systems of the form $\mathbf{A}^\top \mathbf{D} \mathbf{A}$ for non-negative diagonal matrices $\mathbf{D}$. To achieve this result, for all $i \in [n]$ [53] considers 1-self-concordant barriers $\phi_i : (\ell_i, u_i) \rightarrow \mathbb{R}$ for the intervals for i , e.g. $\phi_i(x) = -\log(u_i - x) - \log(x - \ell_i)$, and for a path parameter μ , considers the central path of points

$$x_\mu \stackrel{\text{def}}{=} \arg\min_{\mathbf{A}^\top x = b} c^\top x + \mu \sum_{i=1}^m w_i \phi_i(x_i), \quad (2)$$

for a well-chosen *weight function* $w \in \mathbb{R}^m$. For standard IPMs such as that of Renegar [77], $w_i = 1$ for all i , and in [53] and our algorithm w is a function of x (though it is often convenient to think of w as fixed during a step). Optimality conditions tell us that for fixed w , (2) holds if $\mathbf{A}^\top x_\mu = b$ and $c + \mathbf{A}y + \mu \mathbf{W} \nabla \Phi(x) = 0$ for some vector $y \in \mathbb{R}^n$, where $\mathbf{W} = \text{diag}(w)$ is the diagonal matrix of the weights w . This optimality condition can be re-written as $s + \mu \mathbf{W} \nabla \Phi(x) = 0$ where $s = c + \mathbf{A}y$ denotes the dual slack variables.

In both [53] and this work we allow w to depend on x and define a *weight function* $w(x) : \mathbb{R}^m \rightarrow \mathbb{R}_{>0}^m$. We say that a triple (x, s, μ) is *central* for weight function $w(x)$ if

$$\mathbf{A}^\top x = b, \quad s = \mathbf{A}y + c, \quad \text{and} \quad s + \mu \mathbf{W}(x) \nabla \Phi(x) = 0. \quad (3)$$

A choice of $\mathbf{W}(x)$ made in [53], and which we used a regularized version of in this paper (Definition 2.1), is a ℓ_p Lewis weight for $p = 1 - \frac{1}{4 \log(4m/n)}$. The ℓ_p Lewis weight function $w(x)$ is defined as the solution to

$$w(x) = \sigma(\mathbf{W}(x)^{\frac{1}{2} - \frac{1}{p}} (\nabla^2 \Phi(x))^{-\frac{1}{2}} \mathbf{A}), \quad (4)$$

where $\sigma(\cdot)$ denotes the leverage scores of a matrix. Using this weight function, [53] is able to argue that $\tilde{O}(\sqrt{n})$ steps of an IPM suffice to solve the linear program, as opposed to $\tilde{O}(\sqrt{m})$ from Renegar's method. Carefully implementing methods based on variants of the ℓ_p -Lewis weight function solves the maximum flow problem in time $\tilde{O}(m\sqrt{n})$ and general LPs in time $\tilde{O}(\sqrt{n}(\text{nnz}(\mathbf{A}) + n^\omega))$, accounting for the runtime needed to solve linear systems.

To obtain further runtime improvements, there has been significant work towards performing less work per iteration by speeding up the linear system solve times via inverse maintenance [52], as well as more recent work showing that such methods can in fact be implemented even with only approximate values for the primal and dual variables x, s [8, 11, 12, 21, 37, 54, 84]. To illustrate these *robust IPMs*, consider the simple case where $\ell_i = 0$ and $u_i = \infty$ for all i , i.e. the condition on x in the linear program is simply $x \geq 0$, and $\phi_i(x_i) = -\log x_i$. In this case the centrality condition, (3), reduces to $xs = w(x)\mu$ and this motivates the following *centrality potential*

$$\Psi(x, s, \mu) \stackrel{\text{def}}{=} \sum_{i=1}^m \cosh \left(\lambda \left(\frac{x_i s_i}{w(x)_i \mu} - 1 \right) \right) \quad (5)$$

for $\lambda = \Theta(\log m/\epsilon)$. Maintaining $\Psi(x, s, \mu) \leq \text{poly}(m)$ at all times ensures that $xs \approx_\epsilon w(x)\mu$. Consequently, these robust IPMs take projected Newton steps that induce gradient descent steps on the potential Ψ to guarantee that it stays small in expectation throughout the algorithm.

The analysis in [11, 12] critically relied on the fact that only one-sided constraints, $x \geq 0$, were imposed, instead of a two-sided constraint, $\ell \leq x \leq u$. These works leveraged that the centrality

condition $xs = w(x)\mu$ for ℓ_p Lewis weights can be written in terms of leverage scores of a slightly different diagonal weighting. Specifically, if $xs = w(x)\mu$, where $w(x)$ is the ℓ_p Lewis weight, then for $\alpha = 1 - p$ we have that $xs = \sigma(S^{-1/2-\alpha}X^{1/2-\alpha}A)$. The reliance on this fact impairs extending it to the setting of two-sided constraints.

We bypass this issue by working directly with ℓ_p Lewis weights and the centrality condition (3) for general 1-self-concordant functions ϕ . Interestingly, our analysis requires a fourth derivative condition, beyond the standard third derivative condition of self-concordance (for details, see full version). Formally, we consider the centrality potential

$$\Psi(x, s, \mu) \stackrel{\text{def}}{=} \sum_{i=1}^m \cosh \left(\lambda \left(\frac{s_i + \mu \tau(x)_i \phi'_i(x_i)}{\mu \tau(x)_i \sqrt{\phi''_i(x_i)}} \right) \right), \quad (6)$$

where intuitively, the denominator arises from normalizing by the Hessian of the current point x . To analyze the progress of the Newton steps towards decreasing the potential, analysis is required to understand and bound derivatives of the ℓ_p Lewis weights. Additionally, there are several other technical challenges, including working with a regularized version of the ℓ_p Lewis weight to ensure that the Lewis weights are all $\geq n/m$, carefully maintaining approximate feasibility of the primal variable x , and using spectral sparsifiers of $A^\top DA$ instead of the true matrix for efficient inverse maintenance. This loss of feasibility requires us to use a sampling procedure on the primal variable, and we build a general theory for valid sampling distributions that allow our algorithm to work (Definition 4.13 in the full version) At a high level, we show that any sampling scheme that satisfies various properties, such as bounded variance, maximum, and mean preservation suffices to implement our IPM. The goal of showing that these generalized sampling schemes work is to handle both sampling each coordinate independently and sampling coordinates proportional to weights, so that we can handle the graphical case (as in [11]) and linear programs.

Overall, we show that we can take steps of size $\Omega(n^{-1/2})$ while maintaining that the expected potential is polynomially bounded, and that all points x we maintain are approximately feasible. In this way, we can implement an IPM for two-sided linear programs that requires $\tilde{O}(\sqrt{n})$ steps that only approximately maintains the primal variable x and dual slack s .

3.2 Data Structures

As outlined in Section 3.1, in contrast to [11, 12], our IPM maintains approximate regularized ℓ_p Lewis weights for $p \in [1/2, 2)$. To efficiently implement the IPM we do not want to recompute the Lewis weights from scratch in every iteration. Instead we seek a data structure that maintains approximate Lewis weights. Here we describe how such a data structure can be obtained by reducing to the HEAVYHITTER data structure problem defined below:

Definition 3.1 (Heavy hitter). For $c \in \mathbb{R}^m$ and $P, Q \in \mathbb{R}_{>0}$ with $nP \geq \|c\|_1 \geq P$, we call a data structure with the following procedures a (P, c, Q) -HEAVYHITTER data structure:

- **INITIALIZE**($A \in \mathbb{R}^{m \times n}, g \in \mathbb{R}_{>0}^m$) Let A be a matrix with $c_i \geq \text{nnz}(a_i)$, $\forall i \in [m]$ and $P \geq \text{nnz}(A)$. The data structure initializes in $O(P)$ time.

- **SCALE**($i \in [m], b \in \mathbb{R}$): Sets $g_i \leftarrow b$ in $O(c_i)$ time.
- **QUERYHEAVY**($h \in \mathbb{R}^n, \epsilon \in (0, 1)$): Returns $I \subset [m]$ containing exactly those i with $|(\text{GA}h)_i| \geq \epsilon$ in $O(\epsilon^{-2} \|\text{GA}h\|_c^2 + Q)$ time.

A contribution of our work is to show that, if we have such a data structure for a matrix A , then we are able to efficiently maintain the Lewis weights of VA for a diagonal matrix V that changes over time (i.e. $V = (\nabla^2 \Phi(x))^{-1/2}$ when used inside our IPM, see (4)).

Constructing HEAVYHITTER-data structures was key to advances in [11, 12]. For the special case where A is an edge-vertex incidence matrix, [11] constructed a HEAVYHITTER-data structure with complexities $P = \tilde{O}(m)$, $c_i = \tilde{O}(1)$ for all $i \in [m]$, and $Q = \tilde{O}(n \log W)$, where W is a bound on the ratio of the largest to smallest non-zero entry in G . This data structure will be useful for our min-cost flow application. In [12] a HEAVYHITTER-data structure was given for general $m \times n$ matrices, where $P = \tilde{O}(\text{nnz}(A))$, $c_i = \tilde{O}(n)$ for all $i \in [m]$, and $Q = \tilde{O}(n)$. This data structure can be used for general linear programs. Our algorithm for solving general LPs use this data structure from [12] and the algorithms for graph problems such as min-cost flow use the data structure from [11].

We now outline how to reduce the task of maintaining the Lewis weights $\tau(\text{GA})$ under updates to G , to the HEAVYHITTER problem. This reduction is done via the intermediate data structure problem of maintaining the leverage scores $\sigma(\text{GA})$ under updates to G . We show that Lewis weight maintenance can be reduced to leverage score maintenance and show that leverage score maintenance can be reduced to the HEAVYHITTER problem.

3.2.1 Regularized Lewis Weights. We are interested in the regularized ℓ_p -Lewis weight of a matrix M which is defined as the value $\tau(M)$ that satisfies the recursive equation $\tau = \sigma(T^{1/2-1/p}M) + z$ for a given vector $z \in \mathbb{R}_{>0}^m$, and $T = \text{diag}(\tau)$. We want a data structure that maintains an approximation $\tilde{\tau} \approx_\epsilon \tau(VA)$ for any $p \in [1/2, 2)$ and $\epsilon > 0$ under updates to V . Note that for the IPM we use $p = 1 - 1/(4 \log(4m/n))$; consequently, $p \in [1/2, 2)$ but intuitively, may be thought of as an approximate ℓ_1 -Lewis weight.

To outline our data structure, we first want to outline the algorithm of Cohen and Peng [23] that can be adapted to compute an approximation the regularized ℓ_p -Lewis weight $\tau(M)$ in the static setting (i.e. when the input matrix does not change over time). Given some matrix M we initialize with $w = \vec{1}_m$ and repeatedly set

$$w \leftarrow (w^{2/p-1}(\sigma(W^{1/2-1/p}M) + z))^{p/2}. \quad (7)$$

One can prove (see Lemma 5.3 in the full version) that each iteration reduces the approximation error by a $1 - p/2$ factor, i.e. if we had $w \approx_\gamma \sigma(W^{1/2-1/p}M) + z$ before (7), then we have $w \approx_{\gamma(1-p/2)} \sigma(W^{1/2-1/p}M) + z$ after (7). Since $\vec{1}_m \approx_{O(\log(m))} \sigma(M) + z$ (by $n/m \leq z \leq \text{poly}(m)$), after $\Theta(\log((\log m)/\epsilon))$ iterations of (7) we have $w \approx_\epsilon \tau(M)$.

We provide an efficient extension of this analysis to the dynamic setting. One natural idea for doing this would be to initialize $K = \Omega(\log((\log m)/\epsilon))$ data structures $D_1, \dots, D_K$ that maintain the following approximate leverage scores: Let $w^{(1)} = \vec{1}_m$ and define recursively

$$\bar{\sigma}^{(i)} \approx \sigma((W^{(i)})^{1/2-1/p}VA) + z \quad (8)$$

$$w^{(i+1)} \leftarrow ((w^{(i)})^{2/p-1} \bar{\sigma}^{(i)})^{p/2} \quad (9)$$

where $\bar{\sigma}^{(i)}$ is maintained a the leverage score data structure D_i discussed in Section 3.2.2

If the leverage score data structures are accurate enough (i.e. the approximation in (8) is good enough), then $w^{(K)}$ for some $K = \Omega(\log((\log m)/\epsilon))$ would be a good approximation of the Lewis weight. Further, this $w^{(K)}$ can be maintained under updates to $\mathbf{V}$: When $\mathbf{V}$ changes, we update all the leverage score data structures $D_1, \dots, D_K$. Likewise, if some $\bar{\sigma}^{(i)}$ changes, then we update $w^{(i+1)}$ and the data structure D_{i+1} that maintains

$$\bar{\sigma}^{(i+1)} \approx \sigma((\mathbf{W}^{(i+1)})^{1/2-1/p} \mathbf{V} \mathbf{A}) + z.$$

Problems with this approach. While one can show that the previously outlined approach would indeed allow us to maintain approximate regularized Lewis weights (assuming the approximate leverage scores $\bar{\sigma}^{(i)}$ are accurate enough), we are not able to analyze the time complexity of this process. This is because an update to some D_i (i.e. when $w^{(i-1)}$ changes) causes the output $\bar{\sigma}^{(i)}$ to change as well, thus changing the input to D_{i+1} . This means an update to D_i might propagate through all other $D_{i'}$ for $i' > i$. The computational cost of this propagation of the updates is difficult to analyze because updating the j -th entry of the input of some data structure D_i requires time proportional to $\bar{\sigma}_j^{(i)}$. Now, only for large i do we know that $w^{(i)} \approx \bar{\sigma}^{(i)}$ and can show that, $w^{(i)}$ is an approximation to the regularized Lewis weights. When this happens, we have bounds on how $w^{(i)}$ changes from guarantees of the IPM and this implies a small time complexity for D_i . However, for small i , $w^{(i)} \not\approx \bar{\sigma}^{(i)}$ and the same bounds and complexity analysis does not immediately apply.

One attempt to fix this issue would be to start from a moderately good approximation, i.e.

$$w^{(1)} \approx_{\gamma} \sigma((\mathbf{W}^{(1)})^{1/2-1/p} \mathbf{V} \mathbf{A}) + z \quad (10)$$

for $0 < \gamma = O(\epsilon)$. Then after only $K = O(1)$ recursions we have that $w^{(K)}$ is an ϵ -approximation of the regularized Lewis weights. Since here we only have $O(1)$ data structures D_i and each $\bar{\sigma}^{(i)}$ is at least an $O(\epsilon)$ -approximation of the regularized Lewis weight, we are able to bound the time for propagating the updates through all D_i .

The assumption (10) on $w^{(1)}$ can be satisfied as follows. Assume the input $\mathbf{V}$ changes to some $\mathbf{V}'$. We know by guarantees of the IPM that $\mathbf{V} \approx_{O(\epsilon)} \mathbf{V}'$. Let $\bar{w}^{(K)}$ be the value $w^{(K)}$ we previously returned as ϵ -approximation of the regularized Lewis weight, then we have

$$\bar{w}^{(K)} \approx_{\epsilon} \sigma((\mathbf{W}^{(K)})^{1/2-1/p} \mathbf{V} \mathbf{A}) + z \approx_{O(\epsilon)} \sigma((\mathbf{W}^{(K)})^{1/2-1/p} \mathbf{V}' \mathbf{A}) + z.$$

Thus we can define $w^{(1)} := \bar{w}^{(K)}$ as the required moderately good approximation.

The problem with this approach is that the vector $w^{(K)}$ might change in many entries when switching from $\mathbf{V}$ to $\mathbf{V}'$. Thus we might have to perform many updates to the data structure D_1 at the start of each iteration, resulting in a larger than desired complexity.

The final algorithm. Our regularized Lewis weight data structure combines these ideas with one more trick to bound the number of updates to D_1 ; we delay the updates a bit. We know that we have $w^{(K)} \approx_{\epsilon} \tau$ where τ is the exact regularized Lewis-weight.

We additionally maintain some $\bar{w}^{(K)}$ where we set $\bar{w}_j^{(K)} \leftarrow w_j^{(K)}$ whenever the entry $w_j^{(K)}$ changes and $\bar{w}_j^{(K)} \not\approx_{3\epsilon} w_j^{(K)}$. This way we know $\bar{w}_j^{(K)}$ only changes, whenever τ_j must have changed by at least an $\exp(\epsilon)$ factor. By guarantees of the IPM (Lemma 4.35 in the full version) we can bound how often entries of τ change by an $\exp(\epsilon)$ factor, which then in turn bounds how often entries of $\bar{w}^{(K)}$ are changed. As we set $w^{(1)} := \bar{w}^{(K)}$ whenever the input $\mathbf{V}$ changes, we can now bound the number of updates to D_1 . Note that here we still satisfy the requirement (10) because $\bar{w}^{(K)} \approx_{3\epsilon} w^{(K)}$, so we can still bound the time spent on propagating the updates to D_1 through all other D_i . This way we maintain a good approximation of the Lewis-weight with a low overall complexity bound.

3.2.2 Leverage Scores. In Section 3.2.1 we outlined how to maintain approximate Lewis-weights, if we have access to a data structure that can maintain approximate leverage scores. Here we outline how to efficiently maintain an approximation $\bar{\sigma} \approx_{\epsilon} \sigma(\mathbf{G}\mathbf{A}) + z$ for $\mathbf{G} = \text{Diag}(g)$, $g \in \mathbb{R}_{>0}^m$, $z \in \mathbb{R}_{>0}^m$. Here the vector g is allowed to change over time, while matrix $\mathbf{A}$ and vector z are fixed, and our task is to create a data structure to maintain $\bar{\sigma}$. We obtain such a data structure by reducing to the HEAVYHITTER problem (Definition 3.1) for the same matrix $\mathbf{A}$. Since variants of this have been considered in prior work, we first compare their results and explain why these data structures are not sufficient for our reduction to maintain regularized Lewis weights. We then describe how we obtain our data structure for leverage scores, but we do not yet optimize the complexity to highlight the general idea. At last, we outline how to speed-up the resulting data structure.

Comparison to previous work. The general idea of our leverage score data structure is the same as in [12] and [11], the main difference here is how we improve the complexity. Specifically, the leverage score data structures from [12] and [11] were able to maintain an ϵ -approximation of the leverage scores $\bar{\sigma} \approx_{\epsilon} \sigma(\mathbf{G}\mathbf{A})$, if the input $\mathbf{G}$ was an $O(\epsilon/\log n)$ -approximation of some other $\tilde{\mathbf{G}}$ that satisfied some stability properties, i.e. the diagonal matrix $\tilde{\mathbf{G}}$ must change very slowly over time. For previous IPMs $\mathbf{G} = \bar{\mathbf{X}}^{1/2} \mathbf{S}^{-1/2}$, where $\bar{x} \approx_{O(\epsilon/\log n)} x$, $\bar{s} \approx_{O(\epsilon/\log n)} s$, so $\mathbf{G}$ was an $O(\epsilon/\log n)$ -approximation of $\tilde{\mathbf{G}} := \mathbf{X}^{1/2} \mathbf{S}^{-1/2}$ where both x, s are stable (i.e. they change slowly over the runtime of the algorithm) by guarantees of the IPM. Thus data structures in [12] and [11] were able to maintain leverage scores efficiently.

Unfortunately, these data structures are not usable for our Lewis-weight reduction. This is because in order for the recursion in (8) and (9) to yield an ϵ -approximation of the Lewis-weight, the leverage scores $\bar{\sigma}^{(i)}$ must be at least an ϵ -approximation as well. However, for the old leverage score data structure to return an ϵ -approximation, the input must be $O(\epsilon/\log n)$ -close to some other stable sequence. We use as input $\mathbf{G} = (\mathbf{W}^{(i)})^{1/2-1/p} \mathbf{V}$ (see (9)), which does not satisfy the required property. This is because, while the exact Lewis-weight τ would satisfy the required stability properties by guarantees of the IPM, the input vector $w^{(i)}$ is at best an ϵ -approximation of the exact Lewis-weight τ . In summary, the complexity bounds of the previous leverage score data structures do not apply when we use them for our Lewis-weight reduction due to the additional precision we require.

So while the idea of our leverage score data structure is the same as in [12] and [11], we must analyze and optimize the complexity in a different way.

High-level Idea for maintaining Leverage Scores. Note that the output size (i.e. dimension of $\bar{\sigma}$) is m and we want to maintain the leverage scores in $o(m)$ time, so we can not afford to recompute all entries of $\bar{\sigma}$ in each iteration. Instead, the high-level idea is that in each iteration we (i) detect a set $I \subset [m]$ of indices i where $\bar{\sigma}_i \neq \sigma(\mathbf{V}\mathbf{A})_i + z_i$, and (ii) compute a new approximation of $\sigma(\mathbf{V}\mathbf{A})_i + z_i$ for all $i \in I$ and update $\bar{\sigma}_i$ accordingly. Thus we split the outline of the data structure into these two parts:

Computing Few Leverage Scores: We start by outlining task (ii) as that one is easier. Computing few leverage scores is standard (see e.g. Spielman-Srivastava [85]) and we explain it briefly as we build on it. For a matrix $\mathbf{X}$ let $\mathbf{P}(\mathbf{X}) \stackrel{\text{def}}{=} \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ denote an orthogonal projection matrix. By $\mathbf{P}(\mathbf{V}\mathbf{A}) = \mathbf{P}(\mathbf{V}\mathbf{A})\mathbf{P}(\mathbf{V}\mathbf{A})$ we have that $\mathbf{P}(\mathbf{V}\mathbf{A})_{i,i} = \|\bar{\mathbf{e}}_i^\top \mathbf{P}(\mathbf{V}\mathbf{A})\|_2^2$, so we can reformulate maintaining approximate leverage scores as maintaining an approximation of these norms for $i = 1, \dots, m$. Given some set $I \subset [m]$ we can compute this norm for $i \in I$ by using a JL-matrix² $\mathbf{J}$ and computing the matrix $\mathbf{M} := (\mathbf{A}^\top \mathbf{V}^2 \mathbf{A})^{-1} \mathbf{A} \mathbf{V} \mathbf{J}^\top$. Next, we obtain an approximation of the i -th leverage score by computing

$$\|\bar{\mathbf{e}}_i^\top \mathbf{V} \mathbf{A} \mathbf{M}\|_2^2 \approx \|\bar{\mathbf{e}}_i^\top \mathbf{V} \mathbf{A} (\mathbf{A}^\top \mathbf{V}^2 \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{V}\|_2^2 = \|\bar{\mathbf{e}}_i^\top \mathbf{P}(\mathbf{V}\mathbf{A})\|_2^2. \quad (11)$$

Here the complexity will be dominated by computing $\mathbf{M}$ and computing (11) for all $i \in I$. Given that $\mathbf{J}$ needs only some $\tilde{O}(1)$ rows to yield a good approximation of the norm, we only need to solve very few linear systems in $\mathbf{A}^\top \mathbf{V}^2 \mathbf{A}$ to compute $\mathbf{M}$ and $\mathbf{M}$ has very few columns, so the norm (11) can be computed quickly.

Detecting Leverage Score Changes: We now outline how to solve task (i), i.e. how to detect when $\bar{\sigma}_i \neq \sigma(\mathbf{V}\mathbf{A})_i + z_i$. For that assume that $\mathbf{V}'$ changes to $\mathbf{V}$ and we already had $\bar{\sigma}_i \approx \sigma(\mathbf{V}'\mathbf{A})_i + z_i$ from the previous iteration. Then we must detect indices i where $\sigma(\mathbf{V}'\mathbf{A})_i + z_i \neq \sigma(\mathbf{V}\mathbf{A})_i + z_i$, because for those i the previous $\bar{\sigma}_i$ can no longer be a good approximation. As the vector z is fixed, it suffices to find indices i with $|\sigma(\mathbf{V}'\mathbf{A})_i - \sigma(\mathbf{V}\mathbf{A})_i| > \epsilon z_i$ for some small enough $\epsilon > 0$. Using the interpretation of the leverage scores being the norm of the rows of $\mathbf{P}$, we can find such indices i by searching for indices where

$$\|\bar{\mathbf{e}}_i^\top (\mathbf{P}(\mathbf{V}'\mathbf{A}) - \mathbf{P}(\mathbf{V}\mathbf{A}))\|_2 \geq \|\bar{\mathbf{e}}_i^\top \mathbf{P}(\mathbf{V}'\mathbf{A})\|_2 - \|\bar{\mathbf{e}}_i^\top \mathbf{P}(\mathbf{V}\mathbf{A})\|_2 > \epsilon \sqrt{z_i}$$

If we simply return all i where $v_i \neq v'_i$, then the only remaining i we must detect are those with

$$\|\bar{\mathbf{e}}_i^\top \mathbf{V} \mathbf{Z}^{-1/2} \mathbf{A} \underbrace{((\mathbf{A}^\top \mathbf{V}^2 \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{V} - (\mathbf{A}^\top \mathbf{V}'^2 \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{V}')}_{=: \mathbf{M}'} \mathbf{J}^\top\|_2 > \epsilon$$

where $\mathbf{J}$ is again a JL-matrix and $\mathbf{Z} = \text{Diag}(z)$. Note that because $\mathbf{J}^\top$ has few columns, it suffices to look for large entries of the matrix vector products $\mathbf{V} \mathbf{Z}^{-1/2} \mathbf{A} \mathbf{M}' \bar{\mathbf{e}}_k$ for $k = 1, \dots, \tilde{O}(1)$, which in turn can be solved by the HeavyHitter data structure.

So far we only discussed how to detect indices i where the leverage score changed a lot within a single iteration. However, it could

also happen that a leverage score changes only a little in each iteration such that after many iteration we have $\bar{\sigma}_i \neq \sigma(\mathbf{V}\mathbf{A})_i + z_i$. To detect these slowly changing indices, we follow the approach appeared in [11, 12]. That is, we perform the same trick used for a single iteration, but instead for each $j = 0, \dots, \log \sqrt{n}$, we check whether the leverage score has changed significantly in the past 2^j iterations, i.e., the matrix $\mathbf{V}'$ now refers to the state of $\mathbf{V}$ some 2^j iterations in the past.

Improving the complexity. For the data structure we outlined so far, the main bottleneck is solving linear systems in $\mathbf{A}^\top \mathbf{V}^2 \mathbf{A}$ and computing the product $\mathbf{A}^\top \mathbf{V} \mathbf{J}^\top$. Both of these require $\text{nnz}(\mathbf{A}) = \Omega(m)$ time, which is too slow for our purposes.³ To speed this up, we use leverage score sampling [85] to construct a random sparse diagonal matrix $\mathbf{R}$ with $\tilde{O}(n)$ nonzero entries and $\mathbf{A}^\top \mathbf{V}^2 \mathbf{R} \mathbf{A} \approx \mathbf{A}^\top \mathbf{V} \mathbf{A}$. Careful analysis shows that the algorithm outlined above still works, when solving systems in $\mathbf{A}^\top \mathbf{V}^2 \mathbf{R} \mathbf{A}$ and when using $\mathbf{A}^\top \mathbf{V} \mathbf{R}^{1/2} \mathbf{J}^\top$ instead of $\mathbf{A}^\top \mathbf{V} \mathbf{J}^\top$. Because of the sparsity of $\mathbf{R}$, the $\text{nnz}(\mathbf{A})$ cost decreases to $\tilde{O}(n \cdot \max_i \text{nnz}(a_i))$.

However, this speed-up yields a new problem. If we use two different random matrix $\mathbf{R}$ and $\mathbf{R}'$ with $\mathbf{A}^\top \mathbf{V}^2 \mathbf{R} \mathbf{A} \approx \mathbf{A}^\top \mathbf{V} \mathbf{A}$ and $\mathbf{A}^\top \mathbf{V}'^2 \mathbf{R}' \mathbf{A} \approx \mathbf{A}^\top \mathbf{V}' \mathbf{A}$, then the runtime of the HEAVYHITTER data structure can become very large. This is because the HEAVYHITTER data structures must find large entries of $\mathbf{V} \mathbf{Z}^{-1/2} \mathbf{A} \mathbf{M}' \bar{\mathbf{e}}_k$ and by Definition 3.1 the complexity of that task scales in $\|\mathbf{V} \mathbf{Z}^{-1/2} \mathbf{A} \mathbf{M}' \bar{\mathbf{e}}_k\|_2^2$, so the total cost for all k scales in $\|\mathbf{V} \mathbf{Z}^{-1/2} \mathbf{A} \mathbf{M}'\|_F^2$. Without random sampling, this Frobenius-norm can be bounded by stability properties of the IPM. However, the Frobenius norm is very sensitive to spectral changes which causes the norm to blow-up when using two different $\mathbf{R}$ and $\mathbf{R}'$ (i.e. two different spectral approximations) for $\mathbf{A}^\top \mathbf{V} \mathbf{A}$ and $\mathbf{A}^\top \mathbf{V}' \mathbf{A}$. Thus we wish to use a single random $\mathbf{R}$ that yields a valid approximation for both.

To see how to construct such $\mathbf{R}$, consider classic leverage score sampling first. If one sets $\mathbf{R}_{i,i} = 1/p_i$ independently for each $i \in [m]$ with probability p_i (where $p_i \geq \min(1, \sigma(\mathbf{V}\mathbf{A})_i \log n / \epsilon^2)$) and $\mathbf{R}_{i,i} = 0$ otherwise, then $\mathbf{A}^\top \mathbf{V}^2 \mathbf{R} \mathbf{A} \approx \mathbf{A}^\top \mathbf{V}^2 \mathbf{A}$. To make sure that $\mathbf{R}$ also has the property $\mathbf{A}^\top \mathbf{V}'^2 \mathbf{R} \mathbf{A}$, we use $p_i = 1$ for all i where $\sigma(\mathbf{V}\mathbf{A})_i \neq \sigma(\mathbf{V}'\mathbf{A})_i$ and $p_i = 2\bar{\sigma}_i$ otherwise. The indices i for which we have to set $p_i = 1$ are simply those where we recently had to change $\bar{\sigma}_i$. Further, we need $p_i = 1$ for all i with $v_i \neq v'_i$. This is because we can then bound $\mathbf{A}^\top \mathbf{V}^2 \mathbf{R} \mathbf{A} - \mathbf{A}^\top \mathbf{V}'^2 \mathbf{R} \mathbf{A} = \mathbf{A}^\top \mathbf{R} (\mathbf{V}^2 - \mathbf{V}'^2) \mathbf{A}$ more easily. If we did not choose $p_i = 1$, and $\mathbf{R}_{i,i}$ happens to be non-zero because of the sampling, then $\mathbf{R}_{i,i} (\mathbf{V}^2 - \mathbf{V}'^2)$ would blow-up the difference $(v_i - v'_i)$ by an $1/p_i$ factor. So by choosing $p_i = 1$ for all i with $v_i \neq v'_i$, we are able to prove better complexity bounds.

For comparison, in [12] the leverage score data structure did not use a single $\mathbf{R}$ and instead they used two different $\mathbf{R}$ and $\mathbf{R}'$. They fixed the issue of the Frobenius-norm being large by carefully updating $\mathbf{R}$ to $\mathbf{R}'$ and performing an amortized analysis on the sum of Frobenius-norms over several iterations. However, this analysis required the random $\mathbf{R}$ to be updated over several iterations, which meant the same randomness had to be re-used in all those iterations, thus resulting in difficulties with handling adaptive adversaries. We instead use only one random $\mathbf{R}$ per iteration and this random $\mathbf{R}$ is

²A JL-matrix $\mathbf{J}$ satisfies $\|\mathbf{J}v\|_2 \approx \|v\|_2$ for any fixed vector v . For example a random Gaussian matrix with $O(\epsilon^{-2} \log n)$ rows yields w.h.p a $(1 \pm \epsilon)$ -approximation of the norm.

³This is true, even if we assume access to some preconditioner of $\mathbf{A}^\top \mathbf{V} \mathbf{J}^\top$.

resampled in every iteration, thus no randomness is re-used and adaptive adversaries are not an issue.

3.2.3 Further Data Structures. We now outline all the other data structures used by our algorithms. Some of these data structures were developed in [11, 12] though we perform small modifications to them in the appendix of the full version. Here we give a brief description of these data structures and how they are used to efficiently implement our IPM. A more detailed overview for how to implement our IPM can be found in the full version where we provide the exact statements of the involved data structures.

As mentioned in Section 3.1, our IPM only requires access to approximations $\bar{x}, \bar{s}$ of the iterates x, s . The updates to these vectors are roughly of the following form

$$s^{(\text{new})} \leftarrow s + \mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top \Phi''(\bar{x})^{-1/2}g \quad (12)$$

for some gradient-vector g , matrix $\mathbf{H} \approx \mathbf{A}^\top \bar{\mathbf{T}}^{-1} \Phi''(\bar{x})^{-1} \mathbf{A}$ for diagonal matrix $\bar{\mathbf{T}} = \text{Diag}(\bar{\tau})$ with the approximate Lewis weight $\bar{\tau}$ on the diagonal. Note that naive computation of (12) would require $O(\text{nnz}(\mathbf{A}) + n^\omega)$ time per iteration. This can be sped up via data structures that efficiently maintain partial solutions of this expression. The task of computing (12) can be split into three subtasks: (i) Compute $\mathbf{A}^\top \Phi''(\bar{x})^{-1/2}g$, solved by a data structure from [11] which we modify in the full version. (ii) Multiply the result by $\mathbf{H}^{-1}$, solved either via Laplacian solver (e.g. when considering min-cost flow) or a data structure from [12]. We can essentially use the previous data structure, except we sample $\mathbf{H}$ (the spectral sparsifier) by the ℓ_p Lewis weights that we are already maintaining instead of by leverage score. (iii) Let $v^{(i)}$ be the vector $\mathbf{H}^{-1}\mathbf{A}^\top \Phi''(\bar{x})^{-1/2}g$ we computed with the previous data structures during iteration number i of the IPM. Then after the t -th iteration of the IPM the vector s is given by $s^{(\text{init})} + \sum_{i=1}^t \mathbf{A}v^{(i)}$ according to (12). Maintaining an approximation of such matrix vector products is done via a data structure from [11] which we modify in the full version. At last, Section 3.1 mentioned that the update to the primal solution must be sampled. For graph applications such as max-flow we use a data structure from [11] to perform this sampling efficiently. For general linear programs we construct a new data structure in the full version.

3.3 Putting Everything Together

Given our new IPM for two-sided constraints (Section 3.1) and data structures for implementing this IPM (Section 3.2), we apply them to obtain our results in a standard way.

First of all, our IPM needs to start with a *centered* initial point (i.e. an initial point with small centrality potential (6)). Given an LP instance, we modify the instance by adding an identity block to the constraints and corresponding variables so that a centered initial point can be obtained analytically. This allows us to apply the IPM which moves the initial point to a near optimal point of the modified instance in $\tilde{O}((mn + n^{2.5}) \log W)$ time (using data structures from previous sections). The modified instance also guarantees that, at near-optimal points, the added variables must have value very close to zero. This allows us to round the near optimal point of the modified instance to a near optimal point of the original instance by a single linear system solve which takes $\tilde{O}((\text{nnz}(\mathbf{A}) + n^\omega) \log W)$.

This gives an algorithm for solving an LP with two-sided constraints in Theorem 1.1.

As our IPM maintains not only a primal solution but also a dual slack, by solving a linear system involving the dual slack at the near-optimal point, this gives a dual solution and solves the ℓ_1 -regression problem as stated in Theorem 1.2. Given an ℓ_1 -regression algorithm, we immediately obtain an algorithm for discounted MDPs using a known reduction by [82] and obtain Theorem 1.3.

For a given min-cost flow instance, we modify the instance with the same purpose as above by adding a star. Using graph-based data structures, the path following IPM moves the initial point to a near optimal point of the modified instance in faster $\tilde{O}(m \log W + n^{1.5} \log^2 W)$ time. Analogously, by solving a Laplacian system in $\tilde{O}(m \log W)$ time, we get a near-optimal flow of the original instance. Moreover, since the LP for min-cost flow is integral and the optimal solution can be assumed to be unique (using the isolation lemma as in [11, 26]), we can round the flow on each edge to its nearest integer and obtain an exactly optimal flow. This takes time $\tilde{O}(m \log W + n^{1.5} \log^2 W)$ as promised in Theorem 1.4. For the easier maximum flow problem, we can shave a log W factor on $n^{1.5}$ using a standard scaling technique by [2].

ACKNOWLEDGMENT

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. Jan van den Brand is partially supported by the Google PhD Fellowship Program. Aaron Sidford is supported by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship. Yin Tat Lee is supported by NSF awards CCF-1749609, CCF-1740551, DMS-1839116, DMS-2023166, Microsoft Research Faculty Fellowship, Sloan Research Fellowship, Packard Fellowships. Di Wang did part of this work while at Georgia Tech, and was partially supported by NSF grant CCF-1718533. Yang P. Liu was supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program. We sincerely thank Danupon Nanongkai and Richard Peng for helpful discussions on this project. Also, we thank Danupon Nanongkai for his help on improving the quality in this paper.

REFERENCES

- [1] Alekh Agarwal, Sham M. Kakade, and Lin F. Yang. 2020. Model-Based Reinforcement Learning with a Generative Model is Minimax Optimal. In *COLT (Proceedings of Machine Learning Research, Vol. 125)*. PMLR, 67–83.
- [2] Ravindra K Ahuja and James B Orlin. 1991. Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems. *Naval Research Logistics (NRL)* 38, 3 (1991), 413–430.
- [3] Josh Alman and Virginia Vassilevska Williams. 2021. A Refined Laser Method and Faster Matrix Multiplication. In *SODA*. <https://arxiv.org/pdf/2010.05846.pdf>.
- [4] Peter Auer and Ronald Ortner. 2006. Logarithmic Online Regret Bounds for Undiscounted Reinforcement Learning. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, Bernhard Schölkopf, John C. Platt, and Thomas Hofmann (Eds.). MIT Press, 49–56. <http://papers.nips.cc/paper/3052-logarithmic-online-regret-bounds-for-undiscounted-reinforcement-learning>
- [5] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. 2020. Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs. In *FOCS. IEEE*, 93–104.

- [6] Mohammad Gheshlaghi Azar, Rémi Munos, and Hilbert J Kappen. 2013. Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning* 91, 3 (2013), 325–349.
- [7] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. 2020. Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary. *CoRR* abs/2004.08432 (2020). <https://arxiv.org/pdf/2004.08432>.
- [8] Jan van den Brand. 2020. A Deterministic Linear Program Solver in Current Matrix Multiplication Time. In *SODA*. SIAM, 259–278.
- [9] Jan van den Brand. 2021. Unifying Matrix Data Structures: Simplifying and Speeding up Iterative Algorithms. In *SOSA*. SIAM, 1–13.
- [10] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2021. Minimum Cost Flows, MDPs, and L1-Regression in Nearly Linear Time for Dense Instances. *CoRR* abs/2101.05719 (2021).
- [11] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2020. Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs. In *FOCS*. IEEE, 919–930.
- [12] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. 2020. Solving tall dense linear programs in nearly linear time. In *STOC*. ACM, 775–788. <https://arxiv.org/pdf/2002.02304.pdf>.
- [13] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Dynamic Matrix Inverse: Improved Algorithms and Matching Conditional Lower Bounds. In *FOCS*. IEEE Computer Society, 456–480.
- [14] Hui Han Chin, Aleksander Madry, Gary L. Miller, and Richard Peng. 2013. Runtime guarantees for regression problems. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9–12, 2013*, Robert D. Kleinberg (Ed.). ACM, 269–282. <https://doi.org/10.1145/2422436.2422469>
- [15] Kenneth L. Clarkson. 2005. Subgradient and sampling algorithms for l_1 regression. In *SODA*. 257–266.
- [16] Kenneth L. Clarkson, Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, Xiangrui Meng, and David P. Woodruff. 2016. The Fast Cauchy Transform and Faster Robust Linear Regression. *SIAM J. Comput.* 45, 3 (2016), 763–810.
- [17] Kenneth L. Clarkson and David P. Woodruff. 2013. Low rank approximation and regression in input sparsity time. In *STOC*. ACM, 81–90.
- [18] Jean Cochet-Terrasson, Guy Cohen, Stéphane Gaubert, Michael McGettrick, and Jean-Pierre Quadrat. 1998. Numerical computation of spectral elements in max-plus algebra. *IFAC Proceedings Volumes* 31, 18 (1998), 667–674.
- [19] Michael B Cohen, TS Jayram, and Jelani Nelson. 2018. Simple analyses of the sparse Johnson-Lindenstrauss transform. In *1st Symposium on Simplicity in Algorithms (SOSA)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [20] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. 2014. Solving sdd linear systems in nearly $m \log^{1/2} n$ time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 343–352.
- [21] Michael B. Cohen, Yin Tat Lee, and Zhao Song. 2019. Solving linear programs in the current matrix multiplication time. In *STOC*. ACM, 938–942. <https://arxiv.org/pdf/1810.07896>.
- [22] Michael B Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. 2017. Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $O(m^{10/7} \log W)$ Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 752–771.
- [23] Michael B. Cohen and Richard Peng. 2015. ℓ_p Row Sampling by Lewis Weights. In *STOC*. ACM, 183–192.
- [24] Daniel Dadush, Sophie Huiberts, Bento Natura, and László A. Végh. 2020. A scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22–26, 2020*, Konstantin Makarychev, Yuriy Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (Eds.). ACM, 761–774. <https://doi.org/10.1145/3357713.3384326>
- [25] Daniel Dadush, Bento Natura, and László A. Végh. 2020. Revisiting Tardos's Framework for Linear Programming: Faster Exact Solutions using Approximate Solvers. *CoRR* abs/2009.04942 (2020). <https://arxiv.org/abs/2009.04942>
- [26] Samuel I Daitch and Daniel A Spielman. 2008. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC)*. 451–460.
- [27] Ali Dasdan and Rajesh K. Gupta. 1998. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 17, 10 (1998), 889–899. <https://doi.org/10.1109/43.728912>
- [28] Efim A Dinic. 1970. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, Vol. 11. 1277–1280.
- [29] David Durfee, Kevin A. Lai, and Saurabh Sawlani. 2018. ℓ_1 Regression using Lewis Weights Preconditioning and Stochastic Gradient Descent. In *Conference on Learning Theory, COLT 2018, Stockholm, Sweden, 6–9 July 2018 (Proceedings of Machine Learning Research, Vol. 75)*, Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet (Eds.). PMLR, 1626–1656. <http://proceedings.mlr.press/v75/durfee18a.html>
- [30] Jack Edmonds and Richard M Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)* 19, 2 (1972), 248–264.
- [31] Zvi Galil and Éva Tardos. 1988. An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm. *J. ACM* 35, 2 (1988), 374–386. announced at FOCS'86.
- [32] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *ISSAC*. ACM, 296–303.
- [33] Andrew V. Goldberg and Satish Rao. 1998. Beyond the Flow Decomposition Barrier. *J. ACM* 45, 5 (1998), 783–797. <https://doi.org/10.1145/290179.290181> announced at FOCS'97.
- [34] Andrew V Goldberg and Robert E Tarjan. 1990. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research* 15, 3 (1990), 430–466.
- [35] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. 2020. The Expander Hierarchy and its Applications to Dynamic Graph Algorithms. *CoRR* abs/2005.02369 (2020). <https://arxiv.org/abs/2005.02369>
- [36] Thomas Jaksch, Ronald Ortner, and Peter Auer. 2010. Near-optimal Regret Bounds for Reinforcement Learning. *J. Mach. Learn. Res.* 11 (2010), 1563–1600. <http://portal.acm.org/citation.cfm?id=1855902>
- [37] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2020. Faster Dynamic Matrix Inverse for Faster LPs. *CoRR* abs/2004.07470 (2020). <https://arxiv.org/abs/2004.07470>
- [38] William B. Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*. Contemp. Math., Vol. 26. Amer. Math. Soc., Providence, RI, 189–206. <https://doi.org/10.1090/conm/026/737400>
- [39] Daniel M Kane and Jelani Nelson. 2014. Sparsifier johnson-lindenstrauss transforms. *Journal of the ACM (JACM)* 61, 1 (2014), 1–23.
- [40] 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)*. 745–754.
- [41] Narendra Karmarkar. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* 4, 4 (1984), 373–396. Announced at STOC'84.
- [42] Tarun Kathuria. 2020. A Potential Reduction Inspired Algorithm for Exact Max Flow in Almost $O(m^{4/3})$ Time. In *FOCS*. <https://arxiv.org/pdf/2009.03260.pdf>.
- [43] Michael J. Kearns and Satinder P. Singh. 1998. Finite-Sample Convergence Rates for Q-Learning and Indirect Algorithms. In *Advances in Neural Information Processing Systems 11, [NIPS Conference, Denver, Colorado, USA, November 30 - December 5, 1998]*, Michael J. Kearns, Sara A. Solla, and David A. Cohn (Eds.). The MIT Press, 996–1002. <http://papers.nips.cc/paper/1531-finite-sample-convergence-rates-for-q-learning-and-indirect-algorithms>
- [44] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. 2013. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1–4, 2013*. 911–920.
- [45] Ioannis Koutis, Gary L. Miller, and Richard Peng. 2010. Approaching optimality for solving SDD systems. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 235–244.
- [46] Ioannis Koutis, Gary L. Miller, and Richard Peng. 2011. A nearly $m \log n$ -time solver for SDD linear systems. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 590–598.
- [47] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. 2016. Sparsified cholesky and multigrid solvers for connection laplacians. In *STOC'16: Proceedings of the 48th Annual ACM Symposium on Theory of Computing*.
- [48] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate Gaussian Elimination for Laplacians - Fast, Sparse, and Simple. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9–11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. 573–582.
- [49] Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. 2016. Heavy hitters via cluster-preserving clustering. In *57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, <https://arxiv.org/pdf/1604.01357>, 61–70.
- [50] Yin Tat Lee, Richard Peng, and Daniel A. Spielman. 2015. Sparsified Cholesky Solvers for SDD linear systems. *CoRR* abs/1506.08204 (2015).
- [51] Yin Tat Lee and Aaron Sidford. 2014. Path finding methods for linear programming: Solving linear programs in $O(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. <https://arxiv.org/pdf/1312.6677.pdf>, <https://arxiv.org/pdf/1312.6713.pdf>, 424–433.
- [52] Yin Tat Lee and Aaron Sidford. 2015. Efficient Inverse Maintenance and Faster Algorithms for Linear Programming. In *FOCS*. IEEE Computer Society, 230–249.
- [53] Yin Tat Lee and Aaron Sidford. 2019. Solving Linear Programs with $\sqrt{rank}$ Linear System Solves. In *arXiv preprint*. <https://arxiv.org/pdf/1910.08033.pdf>.
- [54] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. 2019. Solving Empirical Risk Minimization in the Current Matrix Multiplication Time. In *COLT (Proceedings of Machine*

- Learning Research*, Vol. 99). PMLR, 2140–2157. <https://arxiv.org/pdf/1905.04447>.
- [55] Gen Li, Yuting Wei, Yuejie Chi, Yuntao Gu, and Yuxin Chen. 2020. Breaking the Sample Size Barrier in Model-Based Reinforcement Learning with a Generative Model. *CoRR* abs/2005.12900 (2020). arXiv:2005.12900 <https://arxiv.org/abs/2005.12900>
- [56] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. 1995. On the Complexity of Solving Markov Decision Problems. In *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, August 18-20, 1995*. 394–402. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=457&proceeding_id=11
- [57] S. Cliff Liu, Zhao Song, and Hengjie Zhang. 2020. Breaking the n-Pass Barrier: A Streaming Algorithm for Maximum Weight Bipartite Matching. *CoRR* abs/2009.06106 (2020).
- [58] Yang P Liu and Aaron Sidford. 2020. Faster Divergence Maximization for Faster Maximum Flow. In *FOCS*. <https://arxiv.org/pdf/2003.08929.pdf>.
- [59] Yang P Liu and Aaron Sidford. 2020. Faster Energy Maximization for Faster Maximum Flow. In *STOC*. <https://arxiv.org/pdf/1910.14276.pdf>.
- [60] Omid Madani. 2002. Polynomial Value Iteration Algorithms for Deterministic MDPs. In *UAI* 311–318.
- [61] Aleksander Madry. 2013. Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back. In *FOCS*. IEEE Computer Society, 253–262.
- [62] Aleksander Madry. 2016. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 593–602.
- [63] Sridhar Mahadevan. 1996. Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results. *Mach. Learn.* 22, 1-3 (1996), 159–195. <https://doi.org/10.1023/A:1018064306595>
- [64] Renato D. C. Monteiro and Takashi Tsuchiya. 2003. A Variant of the Vavasis–Ye Layered-Step Interior-Point Algorithm for Linear Programming. *SIAM J. Optim.* 13, 4 (2003), 1054–1079. <https://doi.org/10.1137/S1052623401388926>
- [65] Renato D. C. Monteiro and Takashi Tsuchiya. 2005. A New Iteration-Complexity Bound for the MTY Predictor-Corrector Algorithm. *SIAM J. Optim.* 15, 2 (2005), 319–347. <https://doi.org/10.1137/S1052623402416803>
- [66] Vasileios Nakos and Zhao Song. 2019. Stronger ℓ_2/ℓ_2 compressed sensing; without iterating. In *STOC*. ACM, 289–297.
- [67] Vasileios Nakos, Zhao Song, and Zhengyu Wang. 2019. (Nearly) Sample-Optimal Sparse Fourier Transform in Any Dimension; RIPless and Filterless. In *FOCS*. IEEE Computer Society, 1568–1577.
- [68] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. 2017. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. 950–961. <https://doi.org/10.1109/FOCS.2017.92>
- [69] Jelani Nelson and Huy L. Nguyễn. 2013. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, <https://arxiv.org/pdf/1211.1002>, 117–126.
- [70] Yuri E. Nesterov. 2009. Unconstrained Convex Minimization in Relative Scale. *Math. Oper. Res.* 34, 1 (2009), 180–193. <https://doi.org/10.1287/moor.1080.0348>
- [71] Yuri E. Nesterov and Arkadii Nemirovskii. 1991. Acceleration and Parallelization of the Path-Following Interior Point Method for a Linearly Constrained Convex Quadratic Problem. *SIAM J. Optim.* 1, 4 (1991), 548–564.
- [72] James B Orlin. 1984. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. (1984).
- [73] James B Orlin. 1993. A faster strongly polynomial minimum cost flow algorithm. *Operations research* 41, 2 (1993), 338–350.
- [74] Rasmus Pagh. 2013. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)* 5, 3 (2013), 1–17.
- [75] Richard Peng and Daniel A. Spielman. 2014. An efficient parallel solver for SDD linear systems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, David B. Shmoys (Ed.). ACM, 333–342. <https://doi.org/10.1145/2591796.2591832>
- [76] Eric Price, Zhao Song, and David P Woodruff. 2017. Fast Regression with an ℓ_1 Guarantee. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [77] James Renegar. 1988. A polynomial-time algorithm, based on Newton’s method, for linear programming. *Math. Program.* 40, 1-3 (1988), 59–93.
- [78] Piotr Sankowski. 2004. Dynamic Transitive Closure via Dynamic Matrix Inverse (Extended Abstract). In *FOCS*. IEEE Computer Society, 509–517.
- [79] Thatchaphol Saranurak and Di Wang. 2019. Expander Decomposition and Pruning: Faster, Stronger, and Simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. 2616–2635. <https://doi.org/10.1137/1.9781611975482.162>
- [80] Bruno Scherrer. 2016. Improved and generalized upper bounds on the complexity of policy iteration. *Mathematics of Operations Research* 41, 3 (2016), 758–774.
- [81] Aaron Sidford, Mengdi Wang, Xian Wu, Lin Yang, and Yinyu Ye. 2018. Near-Optimal Time and Sample Complexities for Solving Markov Decision Processes with a Generative Model. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 5192–5202.
- [82] Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. 2018. Variance Reduced Value Iteration and Faster Algorithms for Solving Markov Decision Processes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. 770–787. <https://doi.org/10.1137/1.9781611975031.50>
- [83] Christian Sohler and David P. Woodruff. 2011. Subspace embeddings for the ℓ_1 -norm with applications. In *STOC*. ACM, 755–764.
- [84] Zhao Song and Zheng Yu. 2020. Oblivious Sketching-based Central Path Method for Solving Linear Programming Problems. In *manuscript*. <https://openreview.net/forum?id=fGiKxvF-eub>.
- [85] Daniel A Spielman and Nikhil Srivastava. 2011. Graph sparsification by effective resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926.
- [86] Daniel A. Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*. ACM, 81–90.
- [87] Éva Tardos. 1985. A strongly polynomial minimum cost circulation algorithm. *Combinatorica* 5, 3 (1985), 247–255.
- [88] Paul Tseng. 1990. Solving H-horizon, stationary Markov decision problems in time proportional to $\log(H)$. *Operations Research Letters* 9, 5 (1990), 287–297.
- [89] Pravin M. Vaidya. 1989. Speeding-Up Linear Programming Using Fast Matrix Multiplication (Extended Abstract). In *FOCS*. IEEE Computer Society, 332–337.
- [90] Stephen A. Vavasis and Yinyu Ye. 1996. A primal-dual interior point method whose running time depends only on the constraint matrix. *Math. Program.* 74 (1996), 79–120. <https://doi.org/10.1007/BF02592148>
- [91] Martin J. Wainwright. 2019. Variance-reduced Q-learning is minimax optimal. *CoRR* abs/1906.04697 (2019). arXiv:1906.04697 <http://arxiv.org/abs/1906.04697>
- [92] Mengdi Wang. 2017. Randomized Linear Programming Solves the Discounted Markov Decision Problem In Nearly-Linear Running Time. *CoRR* abs/1704.01869 (2017). arXiv:1704.01869 <http://arxiv.org/abs/1704.01869>
- [93] Mengdi Wang. 2020. Randomized linear programming solves the Markov decision problem in nearly linear (sometimes sublinear) time. *Mathematics of Operations Research* 45, 2 (2020), 517–546.
- [94] Ruosong Wang and David P. Woodruff. 2019. Tight Bounds for ℓ_p Oblivious Subspace Embeddings. In *SODA*. SIAM, 1825–1843.
- [95] Virginia Vassilevska Williams. 2012. Multiplying matrices faster than coppersmith-winograd. In *STOC*. ACM, 887–898.
- [96] David P. Woodruff and Qin Zhang. 2013. Subspace Embeddings and ℓ_p -Regression Using Exponential Random Variables. In *COLT (JMLR Workshop and Conference Proceedings, Vol. 30)*. JMLR.org, 546–567.
- [97] Jiyan Yang, Yinlam Chow, Christopher Ré, and Michael W. Mahoney. 2016. Weighted SGD for ℓ_p Regression with Randomized Preconditioning. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, Robert Krauthgamer (Ed.). SIAM, 558–569. <https://doi.org/10.1137/1.9781611974331.ch41>
- [98] Yinyu Ye. 2005. A new complexity result on solving the Markov decision problem. *Mathematics of Operations Research* 30, 3 (2005), 733–749.
- [99] Yinyu Ye. 2011. The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. *Mathematics of Operations Research* 36, 4 (2011), 593–603.