Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs

Jan van den Brand*, Yin-Tat Lee†, Danupon Nanongkai*, Richard Peng‡,
Thatchaphol Saranurak§, Aaron Sidford¶, Zhao Song|| and Di Wang**
*KTH Royal Institute of Technology, Sweden †University of Washington and Microsoft Research Redmond, USA
‡Georgia Institute of Technology, USA §Toyota Technological Institute at Chicago, USA ¶Stanford University, USA
||Columbia University, Princeton University and Institute for Advanced Study, USA **Google Research, USA

***Abstract*—We present an $\widetilde{O}(m + n^{1.5})$-time randomized algorithm for maximum cardinality bipartite matching and related problems (e.g. transshipment, negative-weight shortest paths, and optimal transport) on m-edge, n-node graphs. For maximum cardinality bipartite matching on moderately dense graphs, i.e. $m = \Omega(n^{1.5})$, our algorithm runs in time nearly linear in the input size and constitutes the first improvement over the classic $O(m\sqrt{n})$-time [Dinic 1970; Hopcroft-Karp 1971; Karzanov 1973] and $\widetilde{O}(n^{\omega})$-time algorithms [Ibarra-Moran 1981] (where currently $\omega \approx 2.373$). On sparser graphs, i.e. when $m = n^{9/8+\delta}$ for any constant $\delta > 0$, our result improves upon the recent advances of [Madry 2013] and [Liu-Sidford 2020b, 2020a] which achieve an $\widetilde{O}(m^{4/3+o(1)})$ runtime.**

We obtain these results by combining and advancing recent lines of research in interior point methods (IPMs) and dynamic graph algorithms. First, we simplify and improve the IPM of [v.d.Brand-Lee-Sidford-Song 2020], providing a general primal-dual IPM framework and new sampling-based techniques for handling infeasibility induced by approximate linear system solvers. Second, we provide a simple sublinear-time algorithm for detecting and sampling high-energy edges in electric flows on expanders and show that when combined with recent advances in dynamic expander decompositions, this yields efficient data structures for maintaining the iterates of both [v.d.Brand et al.] and our new IPMs. Combining this general machinery yields a simpler $\widetilde{O}(n\sqrt{m})$ time algorithm for matching based on the logarithmic barrier function, and our state-of-the-art $\widetilde{O}(m + n^{1.5})$ time algorithm for matching based on the [Lee-Sidford 2014] barrier (as regularized in [v.d.Brand et al.]).

***Keywords*-bipartite matching, shortest paths, transshipment, optimal transport, nearly linear time, interior point method, linear program**

I. Introduction

The *maximum-cardinality bipartite matching* problem is to compute a matching of maximum size in an m-edge n-vertex bipartite graph $G = (V, E)$. This problem is one of the most fundamental and well-studied problems in combinatorial optimization, theoretical computer science, and operations research. It naturally encompasses a variety of practical assignment questions and is closely related to a wide range of prominent optimization problems, e.g. optimal transport, shortest path with negative edge-lengths, minimum mean-cycle, etc.

Beyond these many applications, this problem has long served as a barrier towards efficient optimization and a proving ground for new algorithmic techniques. Though numerous combinatorial and continuous approaches have been proposed for the problem, improving upon the classic time complexities of $O(m\sqrt{n})$ [2]–[4] and $O(n^{\omega})$ [5][1] has proven to be notoriously difficult. Since the early 80s, these complexities have only been improved by polylogarithmic factors (see, e.g., [8]) until a breakthrough result of Madry [9] showed that faster algorithms could be achieved when the graph is *moderately sparse*. In particular, Madry [9] showed that the problem could be solved in $\widetilde{O}(m^{10/7})$ and a line of research [10]–[14] led to the recent $\widetilde{O}(m^{4/3+o(1)})$-time algorithm [15].[2] Nevertheless, for *moderately dense graphs*, i.e. $m \geq n^{1.5+\delta}$ for any constant $\delta > 0$, the $O(\min(m\sqrt{n}, n^{\omega}))$ runtime bound has remained unimproved for decades.

The more general problem of *minimum-cost perfect bipartite b-matching*, where an edge can be used multiple times and the goal is to minimize the total edge costs in order to match every node v for $b(v)$ times, for given non-negative integers $b(v)$, has been even more resistant to progress. An $\widetilde{O}(m\sqrt{n})$ runtime for this problem with arbitrary polynomially bounded integer costs and b was achieved only somewhat recently by [16]. Improving this runtime by even a small polynomial factor for moderately dense graphs, is a major open problem (see Table II).

The minimum-cost perfect bipartite b-matching problem can encode a host of problems ranging from transshipment, to negative-weight shortest paths, to maximum-weight bipartite matching. Even more recently, the problem has been popularized in machine learning through its encapsulation of the optimal transport problem [17]–[20]. There has been progress on these problems in a variety of settings (see tables in Section I-A and the full version [1]), including recent improvements for sparse-graphs [12], [14], and nearly linear time algorithms for computing $(1 + \epsilon)$ approximate solutions for maximum-cardinality/weight matching [2]–[4], [21]–[23] and undirected transshipment [24]–[26]. However, obtaining *nearly linear time* algorithms for solving these problems to high-precision for any density regime has been elusive.

A. Our Results

In this paper, we show that maximum cardinality bipartite matching, and more broadly minimum-cost perfect bipartite b-matching, can be solved in $\widetilde{O}(m + n^{1.5})$ time. Tables I and II compare these results with previous ones. Compared to the state-of-the-art algorithms for maximum-cardinality matching our bound is the fastest whenever $m = n^{9/8+\delta}$ for any constant $\delta > 0$, ignoring polylogarithmic factors. Our bound is the first (non-fast-matrix multiplication based) improvement in decades for the case of dense graphs. More importantly, our bound is nearly linear when $m \geq n^{1.5}$. This constitutes the first near-optimal runtime in any density regime for the bipartite matching problem.

The full version of this paper is available as [1] at https://arxiv.org/abs/2009.01802.

[1]Here ω is the matrix multiplication exponent and currently $\omega \approx 2.373$ [6], [7]

[2]For simplicity, we use $\widetilde{O}(\cdot)$ to hide $\operatorname{polylog} n$ and sometimes $\operatorname{polylog}(W)$, where W typically denotes the largest absolute value used for specifying any value in the problem.

Year	Authors	References	Time ($\widetilde{O}(\cdot)$)	
			Sparse	Dense
1969-1973	Hopcroft, Karp, Dinic, Karzanov	[2]–[4]	$m\sqrt{n}$	
1981	Ibarra, Moran	[5]		n^{ω}
2013	Madry	[9]	$m^{10/7}$	
2020	Liu, Sidford	[15]	$m^{4/3}$	
2020	**This paper**			$m + n^{1.5}$

Table I
THE SUMMARY OF THE RESULTS FOR THE **max-cardinality bipartite matching** PROBLEM. FOR A MORE COMPREHENSIVE LIST, SEE [23].

Year	Authors	References	Time ($\widetilde{O}(\cdot)$)
1972	Edmonds and Karp	[27]	mn
2008	Daitch, Spielman	[28]	$m^{3/2}$
2014	Lee, Sidford	[16]	$m\sqrt{n}$
2020	**This paper**		$m + n^{1.5}$

Table II
THE SUMMARY OF THE RESULTS FOR THE **min-cost perfect bipartite b-matching** PROBLEM (EQUIVALENTLY, **transshipment**) FOR POLYNOMIALLY BOUNDED INTEGER COSTS AND b. FOR A MORE COMPREHENSIVE LIST, SEE CHAPTERS 12 AND 21 IN [8]. NOTE THAT THERE HAVE BEEN FURTHER RUNTIME IMPROVEMENTS TO THIS PROBLEM (NOT INCLUDED IN THE TABLE) UNDER THE ASSUMPTION THAT $\|b\|_1 = O(m)$, SEE [12]. RECENTLY, A STATE-OF-THE-ART RUNTIME OF $\widetilde{O}(m^{4/3+o(1)})$ WAS ACHIEVED BY [14] UNDER THIS ASSUMPTION.

Year	Authors	References	Time ($\widetilde{O}(\cdot)$)
1972	Edmonds, Karp	[27]	n^3
2014	Lee, Sidford	[16]	$n^{2.5}$
2017-19	Altschuler, Weed, Rigollet	[20]	n^2W^2/ϵ^2
2018-19	Lin, Ho, Jordan	[19]	$n^{2.5}W^{0.5}/\epsilon$
2018	Quanrud/Blanchet, Jambulapati, Kent, Sidford	[17], [18]	n^2W/ϵ
2020	**This paper**		n^2

Table III
THE SUMMARY OF THE RESULTS FOR THE **optimal transport** PROBLEM.

As a consequence, by careful application of standard reductions, we show that we can solve a host of problems within the same time complexity. These problems are those that can be described as or reduced to the following *transshipment* problem. Given $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, and matrix $\mathbf{A} \in \{0, 1, -1\}^{m\times n}$ where each row of A consists of two nonzero entries, one being 1 and the other being -1, we want to find $x \in \mathbb{R}^m_{\geq 0}$ that achieves the following objective:

$$\min_{\mathbf{A}^\top x=b,\ x\geq 0} c^\top x. \qquad (1)$$

Viewed as a graph problem, we are given a directed graph $G = (V, E)$, a demand function $b : V \to \mathbb{R}$ and a cost function $c : E \to \mathbb{R}$. This problem is then to compute a transshipment $f : E \to \mathbb{R}_{\geq 0}$ that minimizes $\sum_{uv\in E} f(uv)c(uv)$, where a transshipment is a flow $f : E \to \mathbb{R}_{\geq 0}$ such that for every node v,

$$\sum_{uv\in E} f(uv) - \sum_{vw\in E} f(vw) = b(v). \qquad (2)$$

The main result of this paper is the following Theorem I.1 providing our runtime for solving the transshipment problem.

Theorem I.1. *The transshipment problem can be solved to ϵ-additive accuracy in $\widetilde{O}((m+n^{1.5})\log^2(\|b\|_\infty\|c\|_\infty/\epsilon))$ time. For the integral case, where all entries in b, c, and x are integers, the problem can be solved exactly in $\widetilde{O}((m+n^{1.5})\log^2(\|b\|_\infty\|c\|_\infty))$ time.*

Leveraging Theorem I.1 we obtain the following results.

1) A maximum-cardinality bipartite matching can be computed in $\widetilde{O}(m + n^{1.5})$ time, where $\widetilde{O}$ hides $\operatorname{poly}\log(n)$ factors.
2) The minimum-cost perfect bipartite b-matching on graph $G = (V, E)$, with integer edge costs in $[-W, W]$ and non-negative integer $b(v) \leq W$ for all $v \in V$, can be computed in $\widetilde{O}((m+n^{1.5})\log^2(W))$ time.
3) The $\widetilde{O}((m + n^{1.5})\log^2(W))$ time complexity also holds for maximum-weight bipartite matching, negative-weight shortest paths, uncapaciated min-cost flow, vertex-capacitated min-cost s-t flow, minimum mean cost cycle, and deterministic Markov decision processes (here, W denotes the largest absolute value used for specifying any value in the problem).
4) The optimal transport problem can be solved to ϵ-additive accuracy in $\widetilde{O}(n^2 \log^2(W/\epsilon))$ time.

We have already discussed the first two results. Below we briefly discuss some additional results. See Section 8.6 in the full version [1] for the details of all results.

Single-source shortest paths with negative weights and minimum weight bipartite perfect matching.: Due to Gabow and Tarjan's algorithm from 1989 [21], this problem can be solved in $O(m\sqrt{n}\log(nW))$ time where W is the absolute maximum weight of an edge in the graph. For sparse graphs, this has been improved to $\widetilde{O}(m^{10/7}\log W)$ [12] and recently to $\widetilde{O}(m^{4/3+o(1)}\log W)$ [14]. Here, our algorithm obtains a running time of $\widetilde{O}((m + n^{1.5})\log^2(W))$, which again is near-linear for dense graphs and is the lowest known when $m = n^{9/8+\delta}$ for any constant $\delta > 0$ and W is polynomially bounded.

Optimal Transport.: Algorithms with ϵ-additive error received much attention from the machine learning community since the introduction of the algorithm of Altschuler, Weed, Rigollet [20] (e.g. [17]–[19]). The algorithm of [20] runs in time $\widetilde{O}(n^2W^2/\epsilon^2)$, and [18], [19] runs in time $\widetilde{O}(n^2W/\epsilon)$. We improve these running times to $\widetilde{O}(n^2\log^2(W/\epsilon))$. (Note that the problem size is $\Omega(n^2)$.)

B. Techniques

Here we provide a brief high-level overview of our approach (see Section III for a much more detailed and formal overview which links to the main theorems of the paper).

Our results constitute a successful fusion and advancement of two distinct lines of research on *interior point methods* (IPMs) for linear programming [11], [16], [29]–[39] and *dynamic graph* algorithms [40]–[45]. This fusion was precipitated by a breakthrough result of Spielman and Teng [46] in 2004 that Laplacian systems could be solved in nearly linear time. As IPMs for linear programming essentially reduce all the problems considered in this paper to solving Laplacian systems in each iteration, one can hope for a faster algorithm via a combination of fast linear system solvers and interior point methods. Via this approach, Daitch and Spielman [28] showed in 2008 that minimum cost flow and other problems could be solved in $\widetilde{O}(m^{3/2})$ time. Additionally, along this line the results of Madry and others [9]–[14] all showed that a variety of problems could be solved faster. However, as discussed, none of these results lead to improved runtimes for computing maximum cardinality bipartite matching in significantly dense graphs.

More recently, the result of v.d.Brand, Lee, Sidford and Song [38], which in turn was built on [16], [35], [37], led to new possibilities. These methods provide a *robust* IPM framework which allows one to solve many sub-problems required by each iteration *approximately*, instead of doing so *exactly* as required by the previous interior point frameworks. Combining this framework with sophisticated dynamic matrix data structures (e.g., [35]–[37], [47]–[51]) has led to the linear programming algorithm of v.d.Brand et al. [38]. Unfortunately, this algorithm runs in time $\widetilde{O}(mn)$ for graph problems, and this running time seems inherent to the data structures used. Moreover, solving sub-problems only approximately in each iteration leads to infeasible solutions, which were handled by techniques which somewhat complicated and in certain cases inefficient (as this work shows).

Correspondingly this paper makes two major advances. First we show that the data structures (from sparse recovery literature [52]–[59]) used by v.d.Brand et al. [38] can be replaced by more efficient data structures in the case of graph problems. These data structures are based on the *dynamic expander decomposition* data structure developed in the series of works in [40]–[45]. For an unweighted undirected graph G undergoing edge insertions and deletions given as input, this data structure maintains a partition of edges in G into expanders. This data structure was originally developed for the dynamic connectivity problem [41], [44], [45], but has recently found applications elsewhere (e.g. [43], [60], [61]). We can use this data structure to detect entries in the solution that change significantly between consecutive iterations of the robust interior point methods. It was known that this task is a key bottleneck in efficiently implementing prior IPMs methods. Our data structures solve this problem near optimally. We therefore hope that they may serve in obtaining even faster algorithms in the future.

The above data structures allow us to solve the problem needed for each iteration (in particular, some linear system) *approximately*. It is still left open how to use this approximate solution. The issue is that we might not get a feasible solution (we may get x such that $\mathbf{A}x \neq b$ when we try to solve the LP (1)). In [38], this was handled in a complicated way that would at best give an $\widetilde{O}(n^2)$ time complexity for the graph problems we consider. In this paper, we simplify and further improve the method of [38] by sub-sampling entries of the aforementioned approximate solution (and we show that such sampling can be computed efficiently using the aforementioned dynamic expander decompositions). Because of the sparsity of the sampled solution, we can efficiently measure the infeasibity (i.e. compute $\mathbf{A}x - b$) and then fix it in a much simpler way than [38]. We actually provide a general framework and analysis for these types of interior point methods that (i) when instantiated on the log barrier, with our data structures, yields a $\widetilde{O}(n\sqrt{m})$-time algorithm and (ii) when applied using the more advanced barriers of [16] gives our fastest running time.

We believe that our result opens new doors for combining continuous and combinatorial techniques for graph related problems. The recent IPM advances for maximum flow and bipartite matching problems, e.g. [9], [10], [12]–[16], [28] all use Laplacian system solvers [46] or more powerful smoothed-ℓ_p solver [51], [62], [63] *statically* and ultimately spend almost linear work per iteration. In contrast, in addition to using such solvers, we leverage dynamic data-structures for maintaining expanders to implement IPM iterations possibly in sublinear time. Our ultimate runtimes are then achieved by considering the amortized cost of these data structures. We hope this proof of concept of intertwining continuous and combinatorial techniques opens the door to new algorithmic advances.

II. Preliminaries

We write $[n]$ for the interval $\{1, 2, ..., n\}$. When we write *with high probability* (or w.h.p), we mean with probability $1 - n^c$ for any constant $c > 0$.

Diagonal Matrices: Given a vector $v \in \mathbb{R}^d$ for some d, we write $\mathbf{Diag}(v)$ for the $d \times d$ diagonal matrix with $\mathbf{Diag}(v)_{i,i} = v_i$. For vectors $x, s, \overline{s}, \overline{x}, x_t, s_t, w, \overline{w}, w_t, \tau, g$ we let $\mathbf{X} \stackrel{\text{def}}{=} \mathbf{Diag}(x)$, $\mathbf{S} \stackrel{\text{def}}{=} \mathbf{Diag}(s)$, and define $\overline{\mathbf{X}}$, $\overline{\mathbf{S}}$, $\mathbf{X}_t$, $\mathbf{S}_t$, $\mathbf{W}$, $\overline{\mathbf{W}}$, $\mathbf{W}_t$, $\mathbf{T}$, $\mathbf{G}$ analogously.

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 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 $\succ$, $\prec$, and $\succeq$ analogously. We let $\mathbb{S}^{n \times n}_{>0} \subseteq \mathbb{R}^{n \times n}$ denote the set of $n \times n$ symmetric positive definite matrices. We call any matrix (not necessarily symmetric) *non-degenerate* if its rows are all non-zero and it has full column rank.

We use $a \approx_\epsilon b$ to denote that $\exp(-\epsilon) b \leq a \leq \exp(\epsilon) b$ entry-wise and $\mathbf{A} \approx_\epsilon \mathbf{B}$ to denote that $\exp(-\epsilon)\mathbf{B} \preceq \mathbf{A} \preceq \exp(\epsilon)\mathbf{B}$. Note that this notation implies $a \approx_\epsilon b \approx_\delta c \Rightarrow a \approx_{\epsilon+\delta} c$, and $a \approx_\epsilon b \Rightarrow a^x \approx_{\epsilon \cdot x} b^x$ for $x \geq 0$.

For any matrix $\mathbf{A}$ over reals, let $\mathrm{nnz}(\mathbf{A})$ denote the number of non-zero entries in $\mathbf{A}$.

Leverage Scores and Projection Matrices: For any non-degenerate matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ 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. The definition extends to degenerate matrices via the Penrose-Pseudoinverse, i.e. $\mathbf{P}(\mathbf{A}) = \mathbf{A}(\mathbf{A}^\top \mathbf{A})^\dagger \mathbf{A}^\top$. Further, we let $\sigma(\mathbf{A}) \in \mathbb{R}^m$ with $\sigma(\mathbf{A})_i \stackrel{\text{def}}{=} \mathbf{P}(\mathbf{A})_{i,i}$ denote $\mathbf{A}$'s *leverage scores* and let $\mathbf{\Sigma}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{Diag}(\sigma(\mathbf{A}))$, and we let $\tau(\mathbf{A}) \stackrel{\text{def}}{=} \sigma(\mathbf{A}) + \frac{n}{m}\vec{1}$ denote $\mathbf{A}$'s regularized leverage scores and $\mathbf{T}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{Diag}(\tau(\mathbf{A}))$. Finally, we let $\mathbf{P}^{(2)}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{P}(\mathbf{A}) \circ \mathbf{P}(\mathbf{A})$ (where $\circ$ denotes entrywise product), and $\mathbf{\Lambda}(\mathbf{A}) \stackrel{\text{def}}{=} \mathbf{\Sigma}(\mathbf{A}) - \mathbf{P}^{(2)}(\mathbf{A})$.

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 $\|v\|_{\tau+\infty} := \|v\|_\infty + 40\log(4m/n)\|v\|_\tau$, where $m \geq n$ are the dimensions of the constraint matrix of the linear program.

Graph Matrices: Given a directed graph $G = (V, E)$, we define the (edge-vertex) incidence matrix $\mathbf{A} \in \{-1, 0, 1\}^{E \times V}$ via $\mathbf{A}_{e,u} = -1$, $\mathbf{A}_{e,v} = 1$ for every edge $e = (u, v) \in E$. We typically refer to the number of edges by m and the number of nodes by n, so the incidence matrix is an $m \times n$ matrix, which is why we also allow indices $\mathbf{A}_{i,j}$ for $i \in [m]$, $j \in [n]$ by assuming some order to the edges and nodes.

For edge weights $w \in \mathbb{R}^E_{\geq 0}$ we define the Laplacian matrix as $\mathbf{L} = \mathbf{A}^\top \mathbf{W} \mathbf{A}$. For an unweighted undirected simple graph the Laplacian matrix has $\mathbf{L}_{u,v} = -1$ if $\{u, v\} \in E$ and $\mathbf{L}_{v,v} = \deg(v)$, which is the same as the previous definition when assigning arbitrary directions to each edge.

Our algorithm must repeatedly solve Laplacian systems. These types of linear systems are well studied [46], [64]–[72] and we use the following result for solving Laplacian systems (see e.g. Theorem 1.2 of [72]):

Lemma II.1. *There is a randomized procedure that given any n-vertex m-edge graph G with incidence matrix $\mathbf{A}$, diagonal non-negative weight matrix $\mathbf{W}$, and vector $b \in \mathbb{R}^V$ such that there exists an $x \in \mathbb{R}^V$ with $(\mathbf{A}^\top \mathbf{W} \mathbf{A})x = b$ computes $\overline{x} \in \mathbb{R}^V$ with $\|\overline{x} - x\|_{\mathbf{A}^\top \mathbf{W} \mathbf{A}} \leq \epsilon \|x\|_{\mathbf{A}^\top \mathbf{W} \mathbf{A}}$ in $\widetilde{O}(m \log \epsilon^{-1})$ w.h.p.*

Note that we can express the approximation error of Lemma II.1 as some spectral approximation, i.e. that there exists some $\mathbf{H} \approx_{20\epsilon} \mathbf{A}^\top \mathbf{W} \mathbf{A}$ such that $\mathbf{H}\overline{x} = b$ [38, Section 8].

Expanders: We call an undirected graph $G = (V, E)$ a ϕ-expander, if

$$\phi \leq \min_{\emptyset \neq S \subsetneq V} \frac{|\{\{u, v\} \in E \mid u \in S, v \in V \setminus S\}|}{\min\{\sum_{v \in S} \deg(v), \sum_{v \in V \setminus S} \deg(v)\}}.$$

For an edge partition $\bigcup_{i=1}^t E_i = E$, consider the set of subgraphs $G_1, ..., G_t$, where G_i is induced by E_i (with isolated vertices removed). We call this edge partition and the corresponding set of subgraphs a *ϕ-expander decomposition* of G if each G_i is a ϕ-expander.

III. Overview

We start the overview by explaining how our new interior point method (IPM) works in Section III-A. A graph-algorithmic view of this IPM can be found in Section III-B and the full detail can be found in the full version [1]. This new IPM reduces solving linear programs to efficiently performing a number of computations approximately. To efficiently perform these computations for graph problems and implement our IPM we provide new data structures, outlined in Section III-C. Some of these data structures are easy to obtain via known tools, e.g. Laplacian solvers, and some constitute new contributions. In Section III-D we outline our main data structure contributions. The details for these data structures are found in the full version.

A. Interior Point Method

Here we provide an overview of our new efficient sampling-based primal-dual IPMs. Our method builds upon the recent IPM of v.d.Brand, Lee, Sidford, and Song [38] and a host of recent IPM advances [11], [35]–[37]. As with many of these recent methods, given a non-degenerate $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$ and $c \in \mathbb{R}^m$, these IPMs are applied to linear programs represented in the following primal (P) and dual (D) form:

$$(P) \stackrel{\text{def}}{=} \min_{x \in \mathbb{R}^m_{\geq 0} : \mathbf{A}^\top x = b} c^\top x \text{ and } (D) \stackrel{\text{def}}{=} \max_{y \in \mathbb{R}^n, s \in \mathbb{R}^m_{\geq 0} : \mathbf{A}y + s = c} b^\top y. \quad (3)$$

In the remainder of this subsection we explain, motivate, and compare our IPM for this general formulation. For more information about how this IPM is applied in the case of matching problems, see the next subsections.

Path following: As is typical for primal-dual IPMs, both our IPM and the IPMs in [35]–[39] maintain primal $x^{(i)} \in \mathbb{R}^m_{\geq 0}$ and dual slack $s^{(i)} \in \mathbb{R}^m_{\geq 0}$ and proceed for iterations $i = 0, 1, \ldots$ attempting to iteratively improve their quality. In each iteration i, they attempt to compute $(x^{(i)}, s^{(i)})$ so that

$$x^{(i)} s^{(i)} \approx \mu^{(i)} \tau(x^{(i)}, s^{(i)}) \quad (4)$$

for some *path parameter* $\mu^{(i)} \in \mathbb{R}_{\geq 0}$ and *weight function* $\tau(x^{(i)}, s^{(i)}) \in \mathbb{R}^m_{\geq 0}$. (Recall from Section II that $x^{(\ell)} s^{(\ell)}$ is an element-wise multiplication.)

The intuition behind this approach is that for many weight functions, e.g. any constant positive vector, the set of primal-dual pairs $(x_\mu, s_\mu) \in \mathbb{R}^m_{\geq} \times \mathbb{R}^m_{\geq 0}$, that are *feasible*, i.e. satisfy $\mathbf{A}^\top x_\mu = b$ and $\mathbf{A}y + s = c$ for some $y \in \mathbb{R}^n$, and are *μ-centered*, i.e. $xs = \mu \tau(x, s)$, form a continuous curve from solutions to (3), at $\lim_{\mu \to 0}(x_\mu, s_\mu)$, to a type of center of the primal and dual polytopes (in the case they are bounded), at $\lim_{\mu \to \infty}(x_\mu, s_\mu)$. This curve is known as the *central path* and consequently these methods can be viewed as maintaining approximately centrality to approximately follow the central path.

Our methods follow a standard step-by-step approach (similar to [38]) to reduce solving a linear program to efficiently following the central path, i.e. maintaining (4) for changing μ.

Where the IPMs of [35]–[39] and ours all differ is in what weight function is used and how the central path is followed. There is a further complication in some of these methods in that in some cases feasibility of x is not always maintained exactly. In some, linear systems can only be solved to high-precision, however this can be handled by natural techniques, see e.g. [16], [28]. Further, in [38], to allow for approximate linear system solves in the iterations and thereby improve the iteration costs, feasibility of x was maintained more crudely through complicated modifications to the steps. A key contributions of this paper, is a simple sampling-based IPM that also maintains approximately feasible x to further decrease the iteration costs of [38].

Weight function: In this paper we provide a general IPM framework that we instantiate with our sampling-based techniques on two different weight functions $\tau(x^{(i)}, s^{(i)})$. While there are many possible weight functions we restrict our attention to $\tau_{\log}$ induced by the standard logarithmic barrier: $\tau_{\log}(x^{(i)}, s^{(i)}) \stackrel{\text{def}}{=} \vec{1}$ and τ_{LS} a regularized variant of the weights induced by the Lee-Sidford barrier function [11] (also used in [38]) defined as $\tau_{\text{LS}}(x^{(i)}, s^{(i)}) = \sigma(x^{(i)}, s^{(i)}) + \frac{n}{m}\vec{1}$. Above, $\sigma(x^{(i)}, s^{(i)}) \in \mathbb{R}^m$ are *leverage scores* of $\mathbf{A}$ under a particular row re-weighting by $x^{(i)}$, and $s^{(i)}$ as used in, e.g., [11], [16], [38]. Roughly, $\sigma(x^{(i)}, s^{(i)})$ measures the importance of each row of $\mathbf{A}$ with respect to the current primal dual pair $x^{(i)}$ and $s^{(i)}$ in a way that the induced central path is still continuous and can be followed efficiently.

On the one hand, $\tau_{\log}$ is perhaps the simplest weight function one could imagine. The central path it induces is the same as the one induced by penalizing approach the constraints of (P) and (D) with a logarithmic barrier function. Starting with the seminal work of [30] there have been multiple $\widetilde{O}(\sqrt{m})$ iteration IPMs induced by $\tau_{\log}$. On the other hand, τ_{LS} is closely related to the Lewis weight barrier or Lee-Sidford barrier given in [11] and its analysis is more complex. However, in [38] it was shown that this weight function induces a $\widetilde{O}(\sqrt{n})$ iteration IPM. (See [11], [38] for further explanation and motivation of τ_{LS}):

Though the bounds achieved by τ_{LS} in this paper are never worse than those achieved by $\tau_{\log}$ (up to logarithmic factors), we consider both weight functions for multiple reasons. First, the analysis of $\tau_{\log}$ in this paper is simpler than that for $\tau_{\log}$ and yet is sufficient to still obtain $\widetilde{O}(n\sqrt{m}) = \widetilde{O}(n^2)$ time algorithms for the matching problems considered in this paper (and consequently on a first read of this paper one might want to focus on the use of $\tau_{\log}$). Second, the analysis of the two weight functions is very similar and leverage much common algorithmic and analytic machinery. Consequently, considering both barriers demonstrates the versatility of our sampling-based IPM approach.

Centrality potentials: To measure whether (4) holds (for $\tau \in \{\tau_{\log}, \tau_{\mathrm{LS}}\}$) and design our steps, as with previous IPM advances [11], [35]–[39] we use the softmax potential function $\Phi : \mathbb{R}^m \to \mathbb{R}$ defined for all vectors v by $\Phi(v) \stackrel{\text{def}}{=} \sum_{i\in[n]} \phi(v_i)$ where $\phi(v_i) \stackrel{\text{def}}{=} \exp(\lambda(v_i - 1)) + \exp(-\lambda(v_i - 1))$ for some parameter λ. We then define the *centrality measures* or *potentials* as

$$\Phi(x^{(i)}, s^{(i)}, \mu^{(i)}) \stackrel{\text{def}}{=} \Phi\left(\frac{x^{(i)}s^{(i)}}{\mu^{(i)}\tau(x^{(i)}, s^{(i)})}\right)$$

where $\tau \in \{\tau_{\log}, \tau_{\mathrm{LS}}\}$ depending on which weight function is used. Φ intuitively measure how far $x^{(i)}s^{(i)}$ is from $\mu^{(i)}\tau(x^{(i)}, s^{(i)})$, i.e. how far $x^{(i)}$ and $s^{(i)}$ are from being centered and having (4) hold. Observe that $\Phi(v)$ is small when $v = \vec{1}$ (thus $x^{(i)}s^{(i)} = \mu^{(i)}\tau(x^{(i)}, s^{(i)})$) and increases very quickly as v deviates from $\vec{1}$.

The centrality potential we consider has been leveraged extensively by previous IPM advances. In particular, Φ with $\tau = \tau_{\log}$ was used in [35]–[37], [39] and Φ with $\tau = \tau_{\mathrm{LS}}$ was used in [38]. Where our method differs from prior work is in how we design our steps for controlling the value of this potential function a discussed in the next section.

Improvement Step (Short Step): Given the choice of weight function $\tau \in \{\tau_{\log}, \tau_{\mathrm{LS}}\}$ our IPM follows the central path by taking improvement steps (called *short steps*) given by $x^{(i+1)} = x^{(i)} + \eta_x\delta_x^{(i)}$, $s^{(i+1)} = s^{(i)} + \eta_s$, and $\mu^{(i+1)} = \mu^{(i)} + \delta_\mu^{(i)}$. where η_x, η_s are constants depending on whether we use $\tau_{\log}$ or τ_{LS}, and $\delta_x^{(i+1)}$, $\delta_s^{(i+1)}$, and $\delta_\mu^{(i+1)}$ are defined next. Informally, these steps are defined as approximate projected Newton steps of Φ in the appropriate norm. Formally, $\delta_x^{(i)}$ and $\delta_s^{(i)}$ are given by the following equations

$$\begin{aligned}\delta_x^{(i)} &= \overline{\mathbf{X}}^{(i)}\overline{g}^{(i)} \\ &\quad - \mathbf{R}^{(i)}[\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}(\overline{\mathbf{H}}^{(i)})^{-1}\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)} + \delta_c^{(i)}], \quad (5)\\ \delta_s^{(i)} &= \mathbf{A}(\overline{\mathbf{H}}^{(i)})^{-1}\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}. \quad (6)\end{aligned}$$

where the variables in (5) and (6) are described below.

(I) $\overline{x}^{(i)}, \overline{s}^{(i)} \in \mathbb{R}^m$ are any entry-wise, multiplicative approximations of $x^{(i)}$, $s^{(i)}$, and $\overline{\mathbf{X}}^{(i)} = \mathbf{Diag}(\overline{x}^{(i)})$, $\overline{\mathbf{S}}^{(i)} = \mathbf{Diag}(\overline{s}^{(i)})$.

(II) $\overline{g}^{(i)} \in \mathbb{R}^m$: This is an approximate steepest descent direction of Φ with respect to some norm $\|\cdot\|$. Formally, for $\overline{v}^{(i)} \in \mathbb{R}^m$ which is an element-wise approximation of $\frac{x^{(i)}s^{(i)}}{\mu^{(i)}\tau(x^{(i)},s^{(i)})}$, we choose

$$\overline{g}^{(i)} = \underset{z\in\mathbb{R}^m:\|z\|\leq 1}{\operatorname{argmax}} \langle\nabla\Phi(\overline{v}^{(i)}), z\rangle \quad (7)$$

for some norm $\|\cdot\|$ that depends on whether we use $\tau_{\log}$ or τ_{LS}.

(III) $\overline{\mathbf{H}}^{(i)} \in \mathbb{R}^{n\times n}$ is any matrix such that $\overline{\mathbf{H}}^{(i)} \approx \mathbf{A}^\top\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}$.

(IV) $\mathbf{R}^{(i)} \in \mathbb{R}^{m\times m}$ is a randomly selected PSD diagonal matrix chosen so that (i) $\mathbb{E}[\mathbf{R}^{(i)}] = \mathbf{I}$, (ii) $\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{R}^{(i)}\mathbf{A} \approx \mathbf{A}^\top\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}$, and (iii) the second moments of δ_x are bounded. The number of non-zero entries in $\mathbf{R}^{(i)}$ is $\widetilde{O}(n + \sqrt{m})$ when we use $\tau_{\log}$ and $\widetilde{O}(n + m/\sqrt{n})$ when we use τ_{LS}. Intuitively $\mathbf{R}$ randomly samples some rows of the matrix following it in (5) with overestimates of importance measures of the row.

(V) $\delta_c^{(i)} \in \mathbb{R}^m$: a "correction vector" which (as discussed more below), helps control the infeasibility of $x^{(i+1)}$. For a parameter η_c of value $\eta_c \approx 1$ (more precisely, $\eta_c = 1$ for $\tau_{\log}$ and $\eta_c = \frac{1}{1-1/O(\log n)}$ for τ_{LS}) this is defined as

$$\delta_c^{(i)} \stackrel{\text{def}}{=} \eta_c\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}(\overline{\mathbf{H}}^{(i)})^{-1}(\mathbf{A}^\top x^{(i)} - b). \quad (8)$$

Flexibility of variables: Note that there is flexibility in choosing variables of the form $\overline{\square}$, i.e. $\overline{x}^{(i)}$, $\overline{s}^{(i)}$, $\overline{g}^{(i)}$ and $\overline{\mathbf{H}}^{(i)}$. Further, our algorithms have flexibility in the choice of $\mathbf{R}^{(i)}$, we just need too sample by overestimates. This flexibility gives us freedom in how we implement the steps of this method and thereby simplifies the data-structure problem of maintaining them. As in [38], this flexibility is key to our obtaining our runtimes.

Setting $\delta_\mu^{(i+1)}$: As we discuss more below, if $\mathbf{R}^{(i)} = \mathbf{I}$ and $\delta_c^{(i)} = 0$ in (5), our short steps would be almost the same as those in the IPMs in [35], [37]–[39]. For such IPMs, it was shown in [35], [37], [39] (respectively in [38]) that $\delta_\mu^{(i)}$ can be set to be roughly $\widetilde{O}(1/\sqrt{m})\mu^{(i)}$ if we use $\tau_{\log}$ (respectively $\widetilde{O}(1/\sqrt{n})\mu^{(i)}$ if we use τ_{LS}), leading to a method with $\widetilde{O}(\sqrt{m})$ (respectively $\widetilde{O}(\sqrt{n})$) iterations.

In this paper, we can adjust the analyses in [35], [37]–[39] to show that our IPMs require the same number of iterations. In particular, we provide a general framework for IPMs of this type and show that by carefully choosing the distribution for $\mathbf{R}^{(i)}$ (and restarting when necessary) we can preserve the typical convergence rates from [35], [37]–[39] for using $\tau_{\log}$ and τ_{LS} while ensuring that the infeasibility of x is never too large. Provided $\mathbf{R}^{(i)}$ can be sampled efficiently, our new framework supports arbitrary crude polylogarithmic multiplicative approximations of $\overline{\mathbf{H}}^{(i)}$ to $\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\overline{\mathbf{A}}$, in contrast to the high precision approximations required by [35], [37], [39] and a more complicated approximation required in [38]

Motivations and comparisons to previous IPMs: The IPM in [38] and ours share a common feature that they only *approximately* solve linear systems in each iteration, i.e. they apply $(\overline{\mathbf{H}}^{(i)})^{-1}$

to a vector for $\overline{\mathbf{H}}^{(i)} \approx \mathbf{A}^\top \overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}$. While [38] carefully modified the steps to make them feasible in expectation, here we provide a new technique of simply sampling from δ_x to essentially *sparsify* the change in x so that we always know the infeasibility and therefore can better control it. In particular, the short steps in [38] are almost the same as ours with $\mathbf{R}^{(i)} = \mathbf{I}$ and $\delta_c^{(i)} = 0$ in (5). This means that we modify the previous short steps in two ways. First, we sparsify the change in $x^{(i)}$ using a sparse random matrix $\mathbf{R}^{(i)}$ defined in (IV). Since $\mathbb{E}[\mathbf{R}^{(i)}] = \mathbf{I}$, in expectation the behavior of our IPM is similar to that in [38]. However, since $\mathbf{R}^{(i)}$ has $\widetilde{O}(n + m/\sqrt{n})$ non-zero entries (and less for $\tau_{\log}$), we can quickly compute $\mathbf{A}^\top x^{(i+1)} - b$ from $\mathbf{A}^\top x^{(i)} - b$ by looking at $\widetilde{O}(n + m/\sqrt{n})$ rows of $\mathbf{A}$. This information is very useful in fixing the feasibility of $x^{(i+1)}$ so that $\mathbf{A}^\top x^{(i+1)} = b$ in the LP. In particular, while [38] requires a complicated process to keep $x^{(i+1)}$ feasible, we only need our second modification: a "correction vector" $\delta_c^{(i)}$. The idea is that we choose $\delta_c^{(i)}$ so that $x^{(i+1)} = x^{(i)} + \delta_x^{(i)} + \delta_c^{(i)}$ would be feasible if we use $\overline{\mathbf{H}}^{(i)} = \mathbf{A}^\top \overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}$. Although we will still have $\overline{\mathbf{H}}^{(i)} \approx \mathbf{A}^\top \overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}$ and so $x^{(i+1)}$ will be infeasible, the addition of $\delta_c^{(i)}$ fixes some of the previous induced infeasibility. This allows us to bypass the expensive infeasibility fixing step in [38] which takes $\widetilde{O}(mn + n^3)$ time, and improve the running time to $\widetilde{O}(mn + n^{2.5})$, and even less when $\mathbf{A}$ is an incidence matrix.

To conclude, we advance the state-of-the-art for IPMs by providing new methods which can tolerate crude approximate linear system solvers and gracefully handle the resulting loss of infeasibility. Provided certain sampling can be performed efficiently, our methods improve and simplifying aspects of [38]. This new IPM framework, together with our new data structures (discussed next), allow $\tau_{\log}$ to be used to obtain an $\widetilde{O}(n\sqrt{m})$-time matching algorithm and τ_{LS} to be used to obtain our $\widetilde{O}(m+n^{1.5})$-time matching algorithm. We believe that this framework is of independent interest and may find further applications.

B. A Graph-Algorithmic Perspective on our IPM

Here we provide an overview of the IPM discussed in Section III-A, specialized to the graph problems we consider, such as matching and min-cost flow. This subsection is intended to provide further intuition on both our IPM and the data structures we develop for implementing the IPM efficiently. For simplicity, we focus on our IPM with $\tau_{\log}$ in this subsection. In the case of graph problems, typically the natural choice of $\mathbf{A}$ in the linear programming formulations is the incidence matrix $\mathbf{A} \in \{-1, 0, 1\}^{E \times V}$ of a graph (see Section II). The structure of this matrix ultimately enables our methods to have the graph interpretation given in this section and allows us to achieve more efficient data structures (as compared to the case of general linear programs). This interpretation is discussed here and the data structures are discussed in Sections III-C and III-D.

Note that incidence matrices are degenerate; the all-ones vector is always in the kernel and therefore $\mathbf{A}$ is not full column rank (and $\mathbf{A}^\top\mathbf{A}$ is not invertible). Consequently, the algorithms in Section III-A do not immediately apply. This can be fixed by standard techniques (e.g. [28]). In this paper we fix this issue by appending an identity block at the bottom of $\mathbf{A}$ (which can be interpreted as adding self-loops to the input graph). For simplicity, we ignore this issue in this subsection.

Min-cost flow: We focus on the uncapacited min-cost flow (a.k.a. transshipment) problem, where the goal is the find the flow satisfying nodes' demands. Other graph problems can be solved by reducing to this problem (see full version). For simplicity, we focus on computing $\delta_x^{(i)}$ as in (5) and assume that $\eta_x = \eta_s = 1$. Below, entries of any n-dimensional (respectively m-dimensional) vectors are associated with vertices (respectively edges). After i iterations of our IPM, we have

- a flow $\overline{x}^{(i)} \in \mathbb{R}^m$ that is an approximation of a flow $x^{(i)}$ (we do not explicitly maintain $x^{(i)}$ but it is useful for the analysis),
- an approximated slack variable $\overline{s}^{(i)} \in \mathbb{R}^m$, and
- $\mathbf{A}^\top x^{(i)} - b \in \mathbb{R}^n$ called *infeasibility* (a reason for this will be clear later).

We would like to improve the cost of $x^{(i)}$ by augmenting it with flow $\overline{x}^{(i)}\overline{g}^{(i)} \in \mathbb{R}^m$, for some "gradient" vector $\overline{g}^{(i)}$. This corresponds to the first term in (5) and gives us an intermediate m-dimensional flow vector $\dot{x}^{(i+1)} \stackrel{\text{def}}{=} x^{(i)} + \overline{x}^{(i)}\overline{g}^{(i)}$. Let us oversimplify the situation by assuming that $\overline{g}^{(i)}$ has $\widetilde{O}(n)$ non-zero entries, so that computing $\overline{x}^{(i)}\overline{g}^{(i)}$ is not a bottleneck in our runtime. We will come back to this issue later.

Infeasibility: The main problem of $\dot{x}^{(i+1)}$ is that it might be *infeasible*, i.e. $\mathbf{A}^\top \dot{x}^{(i+1)} \neq b$. The infeasibility $\mathbf{A}^\top \dot{x}^{(i+1)} - b$ is due to (i) the infeasibility of $x^{(i)}$ (i.e. $\mathbf{A}^\top x^{(i)} - b$), and (ii) the excess flow of $\overline{x}^{(i)}\overline{g}^{(i)}$, which is $(\mathbf{A}^\top \overline{\mathbf{X}}^{(i)}\overline{g}^{(i)})_v = \sum_{uv \in E} \overline{x}_{uv}^{(i)}\overline{g}_{uv}^{(i)} - \sum_{vu \in E} \overline{x}_{vu}^{(i)}\overline{g}_{vu}^{(i)}$ on each vertex v. This infeasibility would be fixed if we *subtract* $\dot{x}^{(i+1)}$ with some "correction" flow $f_c^{(i)}$ that satisfies, for every vertex v, the demand vector $d^{(i)} \in \mathbb{R}^n$ where

$$d^{(i)} \stackrel{\text{def}}{=} \mathbf{A}^\top \dot{x}^{(i+1)} - b = \mathbf{A}^\top \overline{\mathbf{X}}^{(i)}\overline{g}^{(i)} + (\mathbf{A}^\top x^{(i)} - b). \quad (9)$$

Note that given sparse $\overline{g}^{(i)}$ (as assumed above) and $\mathbf{A}^\top x^{(i)} - b$, we can compute the demand vector $d^{(i)}$ in $\widetilde{O}(n)$ time.

Electrical flow: A standard candidate for $f_c^{(i)}$ is an electrical flow on the input graph $G^{(i)}$ with resistance $r_e^{(i)} = \overline{s}_e^{(i)}/\overline{x}_e^{(i)}$ on each edge e. In a close form, such electrical flow is $f_c^{(i)} = \overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}(\mathbf{H}^{(i)})^{-1}d^{(i)}$, where $\mathbf{H}^{(i)}$ is the Laplacian of $G^{(i)}$. (Note that $(\mathbf{H}^{(i)})^{-1}$ does not exist. This issue can be easily fixed (see full version), so we ignore it for now.) Observe that $f_c^{(i)}$ is exactly the second term of (5) (also see (8)) with $\mathbf{R}^{(i)} = \mathbf{I}$ and $\mathbf{H}^{(i)} = \overline{\mathbf{H}}^{(i)}$. Such $f_c^{(i)}$ can be computed in $\widetilde{O}(m)$ time in every iteration via *fast Laplacian solvers* (Lemma II.1).[3] Since known IPMs require $\Omega(\sqrt{n})$ iterations, this leads to $\widetilde{O}(m\sqrt{n})$ total time at best. This is too slow for our purpose. The main contribution of this paper is a combination of new IPM and data structures that reduces the time per iteration to $\widetilde{O}(n)$.

Spectral sparsifier: A natural approach to avoid $\widetilde{O}(m)$ time per iteration is to approximate $f_c^{(i)}$ using a spectral approximation of $\mathbf{H}^{(i)}$, denoted by $\overline{\mathbf{H}}^{(i)}$. In particular, consider a new intermediate flow $\ddot{x}^{(i+1)} \stackrel{\text{def}}{=} x^{(i)} + \overline{x}^{(i)}\overline{g}^{(i)} - \overline{f}_c^{(i)}$ where $\overline{f}_c^{(i)} \stackrel{\text{def}}{=} \overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}(\overline{\mathbf{H}}^{(i)})^{-1}d^{(i)}$. Note that the definition of $\overline{f}_c^{(i)}$ is exactly the second term of (5) with $\mathbf{R}^{(i)} = \mathbf{I}$, and it differs from $f_c^{(i)}$ only in $\overline{\mathbf{H}}^{(i)}$. Given $d \in \mathbb{R}^n$, computing $(\overline{\mathbf{H}}^{(i)})^{-1}d \in$

[3]We use a $(1 + \epsilon)$-approximation Laplacian solver. Its runtime depends logarithmically on ϵ^{-1}, so we can treat it essentially as an exact algorithm.

$\mathbb{R}^n$ is straightforward: a spectral sparsifier $\overline{\mathbf{H}}^{(i)}$ with $(1+\epsilon)$-approximation ratio and $\widetilde{O}(n/\epsilon^2)$ edges can be maintained in $\widetilde{O}(n/\epsilon^2)$ time per iteration (under the change of resistances), either using the leverage scores [38] or the dynamic sparsifier algorithm of [43]. We then run a fast Laplacian solver on top of such sparsifier to compute $(\overline{\mathbf{H}}^{(i)})^{-1}d$. This requires only $\widetilde{O}(n)$ time per iteration.

Difficulties: There are at least two difficulties in implementing the above idea:

1) *Infeasibility*: An approximate electrical flow $\overline{f}_c^{(i)}$ might not satisfy demand $d^{(i)}$, thus does not fix the infeasibility of $x^{(i)}$.
2) *Time:* Computing $\overline{f}_c^{(i)} \in \mathbb{R}^m$ explicitly requires $\Omega(m)$ time even just to output the result.

Bounding infeasibility and random correction: For the first issue, it turns out that while we cannot keep each $x^{(i)}$ feasible, we can prove that the infeasibility remains small throughout. As a result, we can bound the number of iterations as if every $x^{(i)}$ is feasible (e.g. $\widetilde{O}(\sqrt{m})$ iterations using $\tau_{\log}$). To get around the second issue, we apply the correction flow $\overline{f}_c^{(i+1)}$ only on $\widetilde{O}(n)$ carefully *sampled and rescaled* edges[4]; i.e. our new (final) flow is $x^{(i+1)} \stackrel{\text{def}}{=} x^{(i)} + \overline{x}^{(i)}\overline{g}^{(i)} - \mathbf{R}^{(i)}\overline{f}_c^{(i)}$, for some random diagonal matrix $\mathbf{R}^{(i)} \in \mathbb{R}^{m\times m}$ with $\widetilde{O}(n)$ non-zero entries; in other words, $x_e^{(i+1)} = x_e^{(i)} + \overline{x}_e^{(i)}\overline{g}_e^{(i)} - \mathbf{R}_{e,e}^{(i)}(\overline{f}_c^{(i)})_e$ for every edge e. Observe that this is equivalent to how we define $x^{(i+1)}$ in our IPM ((5) and (6)). Since $\mathbf{R}^{(i)}$ has $\widetilde{O}(n)$ non-zero entries[4], we can compute $h^{(i)} = \mathbf{R}^{(i)}\overline{f}_c^{(i)}$ in $\widetilde{O}(n)$ time.[5]

Our sampled edges basically form an *enhanced* spectral sparsifier, $\mathbf{A}^\top\mathbf{R}^{(i)}\mathbf{A}$. For each edge e, let $p_e^{(i)}$ be a probability that is proportional to the effective resistance of e and $(\overline{f}_c^{(i)})_e$. With probability $p_e^{(i)}$, we set $\mathbf{R}_{e,e}^{(i)} = 1/p_e^{(i)}$ and zero otherwise. Without $(\overline{f}_c^{(i)})_e$ influencing the probability, this graph would be a standard spectral sparsifier. Our enhanced spectral sparsifier can be constructed in $\widetilde{O}(n)$ time using our new data structure based on the dynamic expander decomposition data structure, called *heavy hitter* (discussed in Section III-D and Section 5 of the full version). Compared to a standard spectral sparsifier, it provides some new properties (e.g. $\|\mathbf{R}f_c\|_\infty$ is small in some sense and some moments are bounded) that allow us to bound the number of iterations to be the same as when we do not have $\mathbf{R}^{(i)}$. In other words, introducing $\mathbf{R}^{(i)}$ does not create additional issues (though it does change the analysis and make the guarantees probabilistic), and helps speeding up the overall runtime.

Computing $\overline{x}^{(i+1)}$, $\overline{s}^{(i+1)}$ and $\mathbf{A}^\top x^{(i+1)} - b$: Above, we show how to compute $x^{(i+1)}$ in $\widetilde{O}(n)$ time under an oversimplifying assumption that $\overline{g}^{(i)}$ is sparse. In reality, $\overline{g}^{(i)}$ may be dense and we cannot afford to compute $x^{(i+1)}$ explicitly. A more realistic assumption (although still simplified) is that we can guarantee that the number of non-zero entries in $\overline{g}^{(i)} - \overline{g}^{(i-1)}$ is $\widetilde{O}(\sqrt{m})$.[6] In this case we cannot explicitly compute $\overline{x}^{(i)}\overline{g}^{(i)}$, and thus $x^{(i+1)}$. Instead, we explicitly maintain $\overline{x}^{(i+1)}$ such that for each edge e, $\overline{x}_e^{(i+1)}$ is within a constant factor of $x_e^{(i+1)}$. This means that, for any edge e, if $\ell_e(i)$ is the last iteration before iteration i that we set $\overline{x}_e^{(\ell_e(i))} = x_e^{(\ell_e(i))}$, and $|\sum_{t=\ell_e(i)}^{i}\overline{g}_e^{(t)}| = \Omega(1)$, then we have to set $\overline{x}_e^{(i)} = x_e^{(i)}$. Using the fact that $\overline{g}^{(i)}$ is a unit vector, we can show that we do not have to do this often; i.e. there are $\widetilde{O}(m)$ pairs of (i,e) such that $|\sum_{t=\ell_e(i)}^{i}\overline{g}_e^{(t)}| = \Omega(1)$. By exploiting the fact that $\overline{g}^{(i)} - \overline{g}^{(i-1)}$ contains $\widetilde{O}(\sqrt{m})$ non-zero entries, we can efficiently detect entries of $\overline{x}^{(i)}$ that need to be changed from $\overline{x}^{(i-1)}$. Also by the same fact, we can maintain $d^{(i)}$, thus $\mathbf{R}^{(i)}\overline{f}_c^{(i)}$, in $\widetilde{O}(n)$ time per iteration. This implies that we can computed $\overline{x}^{(i+1)}$ in $\widetilde{O}(n+\sqrt{m}) = \widetilde{O}(n)$ amortized time per iteration.

We are now left with computing $\overline{s}^{(i+1)}$ and $\mathbf{A}^\top x^{(i+1)} - b$. Observe that $\delta_s^{(i)}$ (Eq. (6)) appear as part of $\delta_x^{(i)}$ in (5); so, intuitively, $\overline{s}^{(i)}$ can be computed in a similar way to $\overline{x}^{(i)}$. Note that although $\mathbf{R}^{(i)}$ does not appear in (6), we can use our heavy hitter data structure (mentioned earlier and discussed in Section III-D and Section 5 of the full version) to also detect edges e where $\overline{s}_e^{(i)}$ is no longer a good approximation of $s_e^{(i)}$. That is, when j was the last iteration when we set $\overline{s}_e^{(j)} = s_e^{(i)}$ then we can use the heavy hitter data structure to detect when $|s_e^{(i)} - \overline{s}_e^{(i)}| = |s_e^{(i)} - s_e^{(j)}|$ grows too large, because the difference $s^{(i)} - s^{(j)}$ can be interpreted as some flow again. Finally, note that $\mathbf{A}^\top x^{(i+1)} - b = (\mathbf{A}^\top x^{(i)} - b) + \mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)} - \mathbf{A}^\top\mathbf{R}^{(i)}\overline{f}_c^{(i)}$. The first term is given to us. The last term can be computed quickly due to the sparsity of $\mathbf{R}^{(i)}\overline{f}_c^{(i)}$. The middle term can be maintained in $\widetilde{O}(\sqrt{m})$ time by exploiting the fact that there are $\widetilde{O}(\sqrt{m})$ non-zero entries in $\overline{g}^{(i)} - \overline{g}^{(i-1)}$.

C. Data Structures

As noted earlier, our IPMs are analyzed assuming that the constraint matrix $\mathbf{A}$ of the linear program is non-degenerate (i.e. the matrix $(\mathbf{A}^\top\mathbf{A})^{-1}$ exists). If $\mathbf{A}$ is an incidence matrix, then this is not satisfied. We fix this by appending an identity block at the bottom of $\mathbf{A}$. For proving and discussing the data structures we will, however, assume that $\mathbf{A}$ is just an incidence matrix without this appended identity block, as it results in a simpler analysis.

Ultimately we would like to compute $x^{(\ell)}$ in the final iteration ℓ of the IPM. However, we do *not* compute $x^{(i)}$ or $s^{(i)}$ in iterations $i < \ell$ because it would take to much time. Instead, we implement efficient data structures to maintain the following information about (5) and (6) in every iteration.

i **Primal and Gradient Maintenance** Maintain vectors $\overline{g}^{(i)}$, $\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}$ and $\overline{x}^{(i)} \in \mathbb{R}^m$.
ii **Dual Vectors Maintenance:** Maintain vector $\overline{s}^{(i)} \in \mathbb{R}^m$.

[4]If we use τ_{LS}, the number of edges becomes $\widetilde{O}(m/\sqrt{n})$.

[5]Given $d^{(i)} \in \mathbb{R}^n$, we can compute $(\mathbf{H}^{(i)})^{-1}d^{(i)} \in \mathbb{R}^n$ using spectral sparsifiers and Laplacian solvers as discussed earlier. We can then compute $(\mathbf{R}^{(i)}\overline{f}_c^{(i)})_{uv} = \mathbf{R}_{uv,uv}^{(i)}(\overline{x}^{(i)}/\overline{s}^{(i)})(h_v^{(i)} - h_u^{(i)})$ for every edge uv such that $\mathbf{R}_{uv,uv} \neq 0$.

[6]The actual situations are slightly more complicated. If we use $\tau_{\log}$, we can guarantee that we know some $t^{(i)} \in \mathbb{R}$, for all i, such that $\sum_i \|\overline{g}^{(i)} - t^{(i)}\overline{g}^{(i-1)}\|_0 = \widetilde{O}(m)$; i.e. we can obtain $\overline{g}^{(i)}$ by rescaling $\overline{g}^{(i-1)}$ and change the values of amortized $\widetilde{O}(\sqrt{m})$ non-zero entries. We will stick with the simplified version in this subsection. Note further that if we use τ_{LS}, we can guarantee that entries of each $\overline{g}^{(i)}$ can be divided into $\mathrm{polylog}(n)$ buckets where entries in the same bucket are of the same value. For every i, we can describe the bucketing of $\overline{g}^{(i)}$ by describing $\mathrm{polylog}(n)$ entries in the buckets of $\overline{g}^{(i-1)}$ that move to different buckets in the bucketing of $\overline{g}^{(i)}$. Additionally, each bucket of $\overline{g}^{(i)}$ may take different values than its $\overline{g}^{(i-1)}$ counterpart.

iii **Row (edge) sampling:** Maintain $\mathbf{R}^{(i)}$.

iv **Inverse Maintenance:** Maintain (*implicitly*) $(\overline{\mathbf{H}}^{(i)})^{-1}$. Given $w \in \mathbb{R}^n$, return $(\overline{\mathbf{H}}^{(i)})^{-1}w$.

v **Leverage Scores Maintenance ($\overline{\tau_{\mathrm{LS}}}(x^{(i)}, s^{(i)})$):** When using the faster $\widetilde{O}(\sqrt{n})$-iteration IPM with potential τ_{LS}, we must maintain an approximation $\overline{\tau_{\mathrm{LS}}}(x^{(i)}, s^{(i)})$ of $\tau_{\mathrm{LS}}(x^{(i)}, s^{(i)}) = \sigma(x^{(i)}, s^{(i)}) + n/m$, so we can maintain $\overline{v}^{(i)} \approx \frac{x^{(i)}s^{(i)}}{\mu^{(i)}(\tau_{\mathrm{LS}}x^{(i)}, s^{(i)})}$ which is needed for $\overline{g}^{(i)}$ (in (7)).

vi **Infeasibility Maintenance:** Maintain the n-dimensional vector $(\mathbf{A}^\top x^{(i)} - b)$.

The above values, except for $\overline{g}^{(i)}$ (i) and $(\overline{\mathbf{H}}^{(i)})^{-1}$ (iv), are computed *explicitly*, meaning that their values are maintained in the working memory in every iteration. The vector $\overline{g}^{(i)}$ is maintained in an implicit form, and for $(\overline{\mathbf{H}}^{(i)})^{-1}$, we maintain a data structure that, given $w \in \mathbb{R}^n$, can quickly return $(\overline{\mathbf{H}}^{(i)})^{-1}w$.

Implementing the IPM via i-vi: Below, we repeat (5), (6) and (8) to summarize how we use our data structures to maintain the information in these equations.

$$\delta_x^{(i)} = \underbrace{\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}}_{\text{(i)}} - \underbrace{\mathbf{R}^{(i)}}_{\text{(iii)}}\underbrace{\overline{\mathbf{X}}^{(i)}}_{\text{(i)}}\underbrace{(\overline{\mathbf{S}}^{(i)})^{-1}}_{\text{(ii)}}\mathbf{A}\underbrace{(\overline{\mathbf{H}}^{(i)})^{-1}}_{\text{(iv)}}\underbrace{\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}}_{\text{(i)}} + \underbrace{\mathbf{R}^{(i)}\delta_c^{(i)}}_{\text{below}},$$

$$\delta_s^{(i)} = \mathbf{A}\underbrace{(\overline{\mathbf{H}}^{(i)})^{-1}}_{\text{(iv)}}\underbrace{\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}}_{\text{(i)}}. \tag{10}$$

$$\mathbf{R}^{(i)}\delta_c^{(i)} = \eta_c\underbrace{\mathbf{R}^{(i)}}_{\text{(iii)}}\underbrace{\overline{\mathbf{X}}^{(i)}}_{\text{(i)}}\underbrace{(\overline{\mathbf{S}}^{(i)})^{-1}}_{\text{(ii)}}\mathbf{A}\underbrace{(\overline{\mathbf{H}}^{(i)})^{-1}}_{\text{(iv)}}\underbrace{(\mathbf{A}^\top x^{(i)} - b)}_{\text{(vi)}}.$$

Here we see, that all information required to compute δ_x and δ_s is provided by the data structures i-iv.

Constructing the data structures: Next, we explain how to implement these data structures efficiently. Our main contribution with respect to the data structures are for i, ii, and iii (primal, dual and gradient maintenance and row sampling). These data structures are outlined in Section III-D.

When $\mathbf{A}$ is an incidence matrix, maintaining the inverse implicitly (iv) can be done by maintaining a sparse spectral approximation $\overline{\mathbf{H}}^{(i)}$ of the Laplacian $\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}(\overline{\mathbf{S}}^{(i)})^{-1}\mathbf{A}$ and then running an existing approximate Laplacian system solver [46], [64]–[72]. The spectral approximation $\overline{\mathbf{H}}^{(i)}$ can be maintained using existing tools such as the dynamic spectral sparsifer data structure from [43] or sampling from the leverage scores upper bounds.

Maintaining the leverage scores (v) is done via a data structure from [38] which reduces leverage scores maintenance to dual slack maintenance (ii) with some overhead.

To maintain $\mathbf{A}^\top x^{(i)} - b$ (vi), observe that $\mathbf{A}^\top x^{(i)} - b = \mathbf{A}^\top x^{(i-1)} - b + \mathbf{A}^\top\delta_x^{(i-1)}$. Here $\mathbf{A}^\top x^{(i-1)} - b$ is known from the previous iteration and

$$\mathbf{A}^\top\delta_x^{(i-1)} = \mathbf{A}^\top\overline{\mathbf{X}}^{(i-1)}\overline{g}^{(i-1)} - \mathbf{A}^\top\mathbf{R}^{(i-1)}[\overline{\mathbf{X}}^{(i-1)}(\overline{\mathbf{S}}^{(i-1)})^{-1} \mathbf{A}(\overline{\mathbf{H}}^{(i-1)})^{-1}\mathbf{A}^\top\overline{\mathbf{X}}^{(i-1)}\overline{g}^{(i-1)} + \delta_c^{(i-1)}]$$

can be computed efficiently because of the sparsity of $\mathbf{R}^{(i-1)}$ and the fact that we know the vector $\mathbf{A}^\top\overline{\mathbf{X}}^{(i-1)}\overline{g}^{(i-1)}$ from gradient maintenance (i).

Time complexities: The total time to maintain the data structures i, ii, iv, and vi over ℓ iterations is $\widetilde{O}(m + n\ell)$ when using the slower $\sqrt{m}$-iteration IPM, and $\widetilde{O}(m + (n + m/\sqrt{n})\ell)$ when using the faster $\sqrt{n}$-iteration IPM. The exception is for the leverage scores $\overline{\tau_{\mathrm{LS}}}(x^{(i)}, s^{(i)})$ (v) which is only needed for the $\sqrt{n}$-iteration IPM, where we need $\widetilde{O}(m + n\ell + \ell^2 m/n)$ time. So, in total these data structures take $\widetilde{O}(n\sqrt{m})$ time when we use $\tau_{\log}$ and $\widetilde{O}(m + n\sqrt{n})$ time using τ_{LS}.

D. Primal, Dual, and Gradient Maintenance and Sampling

We first describe how to maintain the approximation $\overline{s}^{(i)} \approx s^{(i)}$, i.e. the data structure of ii. Via a small modification we then obtain a data structure for iii. Finally, we describe a data structure for i which allows us to maintain the gradient $\overline{g}$ and the primal solution $\overline{x}$.

Approximation of s (See Section 5 and 6 of the full version): In order to maintain an approximation $\overline{s}^{(i)} \approx s^{(i)}$ (i.e. a data structure for ii), we design a data structures for the following two problems:

(D1): Maintain the exact vector $s^{(i)} \in \mathbb{R}^m$ implicitly, such that any entry can be queried in $O(1)$ time.

(D2): Detect all indices $j \in [m]$ for which the current $\overline{s}_j^{(i)}$ is no longer a valid approximation of $s_j^{(i)}$.

Task (D1) can be solved easily and we explain further below how to do it. Solving task (D2) efficiently is one of our main contributions and proven in Section 5 of the full version, though we also given an outline in this section further below. Once we solve both tasks (D1) and (D2), we can combine these data structures to maintain a valid approximation of $s^{(i)}$ as follows (details in Section 6 of the full version): Whenever some entry $s_j^{(i)}$ changed a lot so that $\overline{s}_j^{(i)}$ is no longer a valid approximation, (which is detected by (D2)) then we simply query the exact value via (D1) and update $\overline{s}_j^{(i)} \leftarrow s_j^{(i)}$. To construct these data structure, observe that by (10), we have

$$s^{(i+1)} = s^{(i)} + \mathbf{A}\underbrace{(\overline{\mathbf{H}}^{(i)})^{-1}}_{\text{(iv)}}\underbrace{\mathbf{A}^\top\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}}_{\text{(i)}} = s^{(i)} + \mathbf{A}h^{(i)}.$$

Here the vector $h^{(i)} \in \mathbb{R}^n$ can be computed efficiently, thanks to iv and i. So we are left with the problem of maintaining $\overline{s}^{(i+1)} \approx s^{(i+1)} = s^{(\mathrm{init})} + \mathbf{A}\sum_{k=1}^i h^{(k)}$. Here we can maintain $\sum_{k=1}^i h^{(k)}$ in $O(n)$ time per iteration by simply adding the new $h^{(i)}$ to the sum in each iteration. For any j one can then compute $s_j^{(i+1)}$ in $O(1)$ time so we have a data structure that solves (D1).

To get some intuition for (D2), assume we have some $\overline{s}^{(i)}$ with $\overline{s}^{(i)} \approx s^{(i)}$. Now if an entry $(\delta_s^{(i)})_j$ is small enough, then we have $\overline{s}_j^{(i)} \approx s_j^{(i)} + (\delta_s^{(i)})_j = s_j^{(i+1)}$. This motivates why we want to detect a set $J \subset [m]$ containing all j where $|(\delta_s^{(i)})_j|$ is large, and then update $\overline{s}^{(i)}$ to $\overline{s}^{(i+1)}$ by setting $\overline{s}_j^{(i+1)} \leftarrow s_j^{(i+1)}$ for $j \in J$. So for simplicity we start with the simple case where we only need to detect entries of $s^{(i+1)}$ that changed by a lot within a *single* iteration of the IPM. That is, we want to find every index j such that $|(\delta_s^{(i)})_j| = |s_j^{(i+1)} - s_j^{(i)}| > \epsilon s_j^{(i)}$ for some $\epsilon \in (0, 1)$; equivalently, $|(\mathbf{A}h^{(i)})_j| > \epsilon s_j^{(i)}$. We assume that in each iteration we are given the vector $h^{(i)}$. Since $\mathbf{A}$ is an incidence matrix, index j corresponds to some edge (u, v) and thus finding large entries is equivalent to finding edges with $|h_v^{(i)} - h_u^{(i)}| > \epsilon s_{(u,v)}^{(i)}$ where $s_{(u,v)}^{(i)} := s_j^{(i)}$. Assume by induction $\overline{s}_j^{(i-1)} \approx s_j^{(i-1)}$ for all j, so then finding all j's where s_j is changed by a lot in iteration $(i+1)$ reduces to the problem of finding all edges (u,v) such that $|(h_v^{(i)} - h_u^{(i)})/\overline{s}_{(u,v)}^{(i)}| > \epsilon'$ for some $\epsilon' = \Theta(\epsilon)$.

To get the intuition of why we can efficiently find all such edges, start with the simplified case when the edges have uniform weights (i.e. $\overline{s}^{(i)} = \vec{1}$). Since we only care about the differences between entries in the vector h, we can shift h by any constant vector $c \cdot \vec{1}$ in $O(n)$ time to make $h \perp d$, where d is the vector of degrees of the nodes in the graph. For any edge $j = (u,v)$ to have $|h_u - h_v| \geq \epsilon'$, at least one of $|h_u|$ and $|h_v|$ has to be at least $\epsilon'/2$. Thus, it suffices to check the adjacent edges of a node u only when $|h_u|$ is large, or equivalently $h_u^2 \epsilon'^{-2}$ is at least $1/4$. Since checking the adjacent edges of any u takes time $\deg(u)$, the time over all such nodes is bounded by $O(\sum_u \deg(u) h_u^2 \epsilon^{-2})$, which is $O(h^\top \mathbf{D} h \epsilon^{-2})$ where $\mathbf{D}$ is the diagonal degree matrix. If the graph G has conductance at least ϕ (i.e., G is a ϕ-expander), we can exploit the classic spectral graph theory result of Cheeger's inequality to bound the running time by $O(h^\top \mathbf{L} h \phi_G^{-2} \epsilon^{-2})$, where $\mathbf{L} = \mathbf{A}^\top \mathbf{A}$ is the graph Laplacian. Here $h^\top \mathbf{L} h = \|\mathbf{A} h\|_2^2$ will be small due to properties of our IPM, and thus this already gives us an efficient implementation if our graph has large conductance $\phi = 1/\operatorname{polylog}(n)$.

To extend the above approach to the real setting where $\overline{s}$ is non-uniform and the graph is not an expander, we only need to partition the edges of G so that these two properties hold in each induced subgraph. For the non-uniform $\overline{s}$ part, we bucket the edges by their weights in $\overline{s}^{(i)}$ into edge sets $E_k^{(i)} = \{(u,v) \mid \overline{s}^{(i)}(u,v) \in [2^k, 2^{k+1})\}$, so edges in each bucket have roughly uniform $\overline{s}$ weights. To get the large conductance condition, we further partition each $E_k^{(i)}$ into expander subgraphs, i.e., $E_{k,1}^{(i)}, E_{k,2}^{(i)}, \ldots$, each inducing an $(1/\operatorname{polylog}(n))$-expander. Note that edges move between buckets over the iterations as $\overline{s}^{(i)}$ changes, so we need to maintain the expander decompositions in a dynamic setting. For this we employ the algorithm of [43] (building on tools developed for dynamic minimum spanning tree [40]–[42], [44], [45], especially [40]). Their dynamic algorithm can maintain our expander decomposition efficiently (in $\operatorname{polylog}(n)$ time per weight update). With the dynamic expander decomposition, we can essentially implement the method discussed above in each expander as follows. For each expander subgraph, we constrain the vector h to the nodes of the expander. Then, we translate h by the all-one vector so that h is orthogonal to the degree vector of the nodes in the expander. To perform the translation on all expanders, we need the total size (in terms of nodes) of the induced expanders to be small for the computation to be efficient. We indeed get this property as the dynamic expander decomposition algorithm in [43] guarantees $\sum_q |V(E_{k,q}^{(i)})| = O(n \log n)$. In total this will bound the running time in the i^{th} iteration of the IPM to be

$$\widetilde{O}((\epsilon')^{-2} \|(\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} h^{(i)}\|_2^2 + n \log W),$$

where W is a bound on the ratio of largest to smallest entry in $\overline{s}$. By properties of the IPM, which bound the above norm, the total running time of our data structure over all $\widetilde{O}(\sqrt{n})$ iterations of the IPM becomes $\widetilde{O}(m + n^{1.5})$, or $\widetilde{O}(\sqrt{m}n)$ when using the slower $\widetilde{O}(\sqrt{m})$ iteration IPM.

In the above we only consider detecting entries of $s^{(i)}$ undergoing large changes in a single iteration. In order to maintain $\overline{s}^{(i)}$, we also need to detect entries of $s^{(i)}$ that change slowly every iteration, but accumulate enough change across multiple iterations so our approximation is no longer accurate enough. This can be handled via a reduction similar to the one performed in [38], where we employ lazy update and batched iteration tracking. In particular, for every $k = 0, 1, \ldots, \lceil (\log n)/2 \rceil$, we use a copy of (D2) to check every 2^k iterations of the IPM, if some entry changes large enough over the past 2^k iterations. This reduction only incurs a $\operatorname{polylog}(n)$ factor overhead in running time comparing to the method that only detects large single iteration changes, so the total running time is the same up to $\operatorname{polylog}$ factors.

Row Sampling (See Section 5 in the full version): Another task is to solve data structure problem iii which is about constructing the random matrix $\mathbf{R}^{(i)}$. The desired distribution of $\mathbf{R}^{(i)}$ is as follows. For some large enough constant $C > 0$ let $q \in \mathbb{R}^m$ with $q_j \geq \sqrt{m}((\delta_r^{(i)})_j^2 / \|\delta_r\|_2^2 + 1/m) + C \cdot \sigma((\overline{\mathbf{X}}^{(i)})^{1/2} (\overline{\mathbf{S}}^{(i)})^{-1/2} \mathbf{A})_j \operatorname{polylog} n$ where $\delta_r^{(i)} = \overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} (\overline{\mathbf{H}}^{(i)})^{-1} \mathbf{A} \overline{\mathbf{X}}^{(i)} \overline{g}^{(i)} + \delta_c^{(i)}$ then we have $\mathbf{R}_{j,j}^{(i)} = (\min(q_i, 1))^{-1}$ with probability $\min(q_i, 1)$ and 0 otherwise.

This sampling task can be reduced to the two tasks of (i) sampling according to $\sqrt{m}((\delta_r)_j^2 / \|\delta_r\|_2^2$ and (ii) sampling according to $C \cdot \sigma((\overline{\mathbf{X}}^{(i)})^{1/2} (\overline{\mathbf{S}}^{(i)})^{-1/2} \mathbf{A}) \operatorname{polylog} n$. The latter can be implemented easily as we have approximate leverage scores via data structure v. The former is implemented in a similar way as data structure (D2) of the previous paragraph. Instead of finding large entries of some vector $(\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} h^{(i)}$, we now want to sample the entries proportional to $\overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} h'^{(i)}$ where $h'^{(i)} = (\overline{\mathbf{H}}^{(i)})^{-1} (\mathbf{A} \overline{\mathbf{X}}^{(i)} \overline{g}^{(i)} + \mathbf{A}^\top x^{(i)} - b)$.

This sampling can be constructed via a simple modification of the previous (D2) data structure. Where (D2) tries to find edges (u,v) with large $|((\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} h^{(i)})_{(u,v)}|$ by looking for nodes v with large $|h_v^{(i)}|$, we now similarly sample edges (u,v) proportional to $(\overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} h'^{(i)})_{(u,v)}^2$ by sampling for each node v incident edges proportional to $(h_v^{(i)})^2$.

Gradient Maintenance and Approximation of x (Section 7 in the full version): For the primal solution x, again we aim to maintain a good enough approximation $\overline{x}$ through our IPM algorithm. Consider the update to $x^{(i)}$ in (5),

$$\begin{aligned} x^{(i+1)} = x^{(i)} + \overline{\mathbf{X}}^{(i)} \overline{g}^{(i)} \\ - \mathbf{R}^{(i)} \overline{\mathbf{X}}^{(i)} (\overline{\mathbf{S}}^{(i)})^{-1} \mathbf{A} (\overline{\mathbf{H}}^{(i)})^{-1} \mathbf{A}^\top \overline{\mathbf{X}}^{(i)} \overline{g}^{(i)} + \mathbf{R}^{(i)} \delta_c^{(i)} \end{aligned}$$

the last two terms will be sparse due to the sparse diagonal sampling matrix $\mathbf{R}^{(i)}$, so we can afford to compute that part of the updates explicitly. For the part of $\overline{\mathbf{X}}^{(i)} \overline{g}^{(i)}$ where $\overline{g}^{(i)} = \operatorname{argmax}_{z \in \mathbb{R}^m : \|z\| \leq 1} \langle \nabla \Phi(\overline{v}^{(i)}), z \rangle$ (see (7)) we will show that $\overline{g}$ admits a low dimensional representation. Here by low dimensionality of $\overline{g} \in \mathbb{R}^m$ we mean that the m indices in the vector can be put into $\widetilde{O}(1)$ buckets, where indices j, j' in the same bucket share the common value $\overline{g}_j = \overline{g}_{j'}$. This allows us to represent the values of $\overline{g}$ as a $\widetilde{O}(1)$ dimensional vector so we can efficiently represent and do computations with $\overline{g}$ in a very compact way.

For simplicity consider the case where we use $\|\cdot\|_2$ as the norm for the maximization problem that defines $\overline{g}^{(i)}$ (this norm is used by the $\sqrt{m}$-iteration IPM, while the $\sqrt{n}$-iteration IPM uses a slightly more complicated norm). In that case $\overline{g}^{(i)} =$

$\nabla\Phi(\overline{v}^{(i)})/\|\nabla\Phi(\overline{v}^{(i)})\|_2$ and the way we construct the $\widetilde{O}(1)$ dimensional approximation is fairly straightforward. We essentially discretize $\overline{v}^{(i)}$ by rounding each entry down to the nearest multiple of some appropriate granularity to make it low dimensional. Once $\overline{v}^{(i)}$ is made to be $\widetilde{O}(1)$ dimensional, it is simple to see from the definition of the potential function $\Phi(\cdot)$ that $\nabla\Phi(\overline{v}^{(i)})$ will also be in $\widetilde{O}(1)$ dimension. For the faster $\sqrt{n}$ iteration IPM where a different norm $\|\cdot\|$ is used, we can show that the low dimensionality of $\nabla\Phi(\overline{v}^{(i)})$ also translates to the maximizer being in low dimensional.

Once we compute the low dimensional updates, we still need to track accumulated changes of $\overline{\mathbf{X}}^{(i)}\overline{g}^{(i)}$ over multiple iterations. Because of properties of the IPM, we have that on average any index j switches its bucket (of the low dimensional representation of $\overline{g}$) only some $\operatorname{polylog}(n)$ times. Likewise, the value of any entry $\overline{\mathbf{X}}_j^{(i)}$ changes only some $\operatorname{polylog}(n)$ number of times. Thus the rate in which $\sum_{k=1}^{i}\overline{\mathbf{X}}_j^{(k)}\overline{g}_j^{(k)}$ changes, stays the same for many iterations. This allows us to (A) predict when $\overline{x}_j^{(i)}$ is no longer a valid approximation of $x_j^{(i)}$, and (B) the low dimensionality allows us to easily compute any $x_j^{(i)}$. In the same way as $\overline{s}^{(i)}$ was maintained via (D1) and (D2), we can now combine (A) and (B) to maintain $\overline{x}^{(i)}$.

Acknowledgment

We thank Yang Liu for helpful conversations, feedback on earlier drafts of the paper, and technical suggestions. 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. Danupon Nanongkai is also partially supported by the Swedish Research Council (Reg. No. 2015-04659 and 2019-05622). Aaron Sidford is supported by NSF CAREER Award CCF-1844855 and a PayPal research gift. Yin Tat Lee is supported by NSF awards CCF-1749609, CCF-1740551, DMS-1839116, Microsoft Research Faculty Fellowship, a Sloan Research Fellowship. Di Wang did part of this work while at Georgia Tech, and was partially supported by NSF grant CCF-1718533. Richard Peng was partially supported by NSF grants CCF-1718533 and CCF-1846218. Zhao Song was partially supported by Ma Huateng Foundation, Schmidt Foundation, Simons Foundation, NSF, DARPA/SRC, Google and Amazon.

References

[1] J. v. d. Brand, Y. T. Lee, D. Nanongkai, R. Peng, T. Saranurak, A. Sidford, Z. Song, and D. Wang, "Bipartite matching in nearly-linear time on moderately dense graphs," *CoRR*, vol. abs/2009.01802, 2020.

[2] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput.*, vol. 2, no. 4, pp. 225–231, 1973, announced at FOCS'71.

[3] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," in *Soviet Math. Doklady*, vol. 11, 1970, pp. 1277–1280.

[4] A. V. Karzanov, "On finding maximum flows in networks with special structure and some applications," *Matematicheskie Voprosy Upravleniya Proizvodstvom*, vol. 5, pp. 81–94, 1973.

[5] O. H. Ibarra and S. Moran, "Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication," *Inf. Process. Lett.*, vol. 13, no. 1, pp. 12–15, 1981.

[6] F. L. Gall, "Powers of tensors and fast matrix multiplication," in *ISSAC*. ACM, 2014, pp. 296–303.

[7] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *STOC*. ACM, 2012, pp. 887–898.

[8] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[9] A. Madry, "Navigating central path with electrical flows: From flows to matchings, and back," in *FOCS*. IEEE Computer Society, 2013, pp. 253–262.

[10] ——, "Computing maximum flow with augmenting electrical flows," in *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2016, pp. 593–602.

[11] Y. T. Lee and A. Sidford, "Solving linear programs with $\sqrt{rank}$ linear system solves," in *arXiv preprint*. https://arxiv.org/pdf/1910.08033.pdf, 2019.

[12] M. B. Cohen, A. Madry, P. Sankowski, and A. Vladu, "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, 2017, pp. 752–771.

[13] Y. P. Liu and A. Sidford, "Faster energy maximization for faster maximum flow," in *STOC*. https://arxiv.org/pdf/1910.14276.pdf, 2020.

[14] K. Axiotis, A. Madry, and A. Vladu, "Circulation control for faster minimum cost flow in unit-capacity graphs," in *arXiv preprint*. https//arxiv.org/pdf/2003.04863.pdf, 2020.

[15] Y. P. Liu and A. Sidford, "Faster divergence maximization for faster maximum flow," in *arXiv preprint*. https://arxiv.org/pdf/2003.08929.pdf, 2020.

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

[17] J. H. Blanchet, A. Jambulapati, C. Kent, and A. Sidford, "Towards optimal running times for optimal transport," *CoRR*, vol. abs/1810.07717, 2018.

[18] K. Quanrud, "Approximating optimal transport with linear programs," in *SOSA*, 2019.

[19] T. Lin, N. Ho, and M. I. Jordan, "On efficient optimal transport: An analysis of greedy and accelerated mirror descent algorithms," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 3982–3991.

[20] J. Altschuler, J. Weed, and P. Rigollet, "Near-linear time approximation algorithms for optimal transport via sinkhorn iteration," in *NIPS*, 2017, pp. 1964–1974.

[21] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM J. Comput.*, vol. 18, no. 5, pp. 1013–1036, 1989.

[22] ——, "Faster scaling algorithms for general graph-matching problems," *J. ACM*, vol. 38, no. 4, pp. 815–853, 1991.

[23] R. Duan and S. Pettie, "Linear-time approximation for maximum weight matching," *J. ACM*, vol. 61, no. 1, pp. 1:1–1:23, 2014.

[24] J. Sherman, "Generalized preconditioning and undirected minimum-cost flow," in *SODA*. SIAM, 2017, pp. 772–780.

[25] A. Andoni, C. Stein, and P. Zhong, "Parallel approximate undirected shortest paths via low hop emulators," *STOC*, vol. https://arxiv.org/pdf/1911.01956.pdf, 2020.

[26] J. Li, "Faster parallel algorithm for approximate shortest path," in *STOC*. https://arxiv.org/pdf/1911.01626.pdf, 2020.

[27] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[28] S. I. Daitch and D. A. Spielman, "Faster approximate lossy generalized flow via interior point algorithms," in *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC)*, 2008, pp. 451–460.

[29] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–396, 1984, announced at STOC'84.

[30] J. Renegar, "A polynomial-time algorithm, based on newton's method, for linear programming," *Math. Program.*, vol. 40, no. 1-3, pp. 59–93, 1988.

[31] P. M. Vaidya, "An algorithm for linear programming which requires $O(((m+n)n^2+(m+n)^{1.5}n)L)$ arithmetic operations," in *STOC*. ACM, 1987, pp. 29–38.

[32] P. M. Vaidya and D. S. Atkinson, "A technique for bounding the number of iterations in path following algorithms," in *Complexity in Numerical Optimization*. World Scientific, 1993, pp. 462–489.

[33] K. M. Anstreicher, "Volumetric path following algorithms for linear programming," *Math. Program.*, vol. 76, pp. 245–263, 1996.

[34] Y. E. Nesterov and M. J. Todd, "Self-scaled barriers and interior-point methods for convex programming," *Math. Oper. Res.*, vol. 22, no. 1, pp. 1–42, 1997.

[35] M. B. Cohen, Y. T. Lee, and Z. Song, "Solving linear programs in the current matrix multiplication time," in *STOC*, 2019, https://arxiv.org/pdf/1810.07896.

[36] Y. T. Lee, Z. Song, and Q. Zhang, "Solving empirical risk minimization in the current matrix multiplication time," in *COLT*. https://arxiv.org/pdf/1905.04447, 2019.

[37] J. v. d. Brand, "A deterministic linear program solver in current matrix multiplication time," in *SODA*. SIAM, 2020, pp. 259–278.

[38] J. v. d. Brand, Y. T. Lee, A. Sidford, and Z. Song, "Solving tall dense linear programs in nearly linear time," in *STOC*. https://arxiv.org/pdf/2002.02304.pdf, 2020.

[39] S. Jiang, Z. Song, O. Weinstein, and H. Zhang, "Faster dynamic matrix inverse for faster lps," *CoRR*, vol. abs/2004.07470, 2020.

[40] T. Saranurak and D. Wang, "Expander decomposition and pruning: Faster, stronger, and simpler," in *SODA*. SIAM, 2019, pp. 2616–2635.

[41] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "Dynamic minimum spanning forest with subpolynomial worst-case update time," in *FOCS*. IEEE Computer Society, 2017, pp. 950–961.

[42] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," in *FOCS*, 2020, https://arxiv.org/pdf/1910.08025.pdf.

[43] A. Bernstein, J. v. d. Brand, M. P. Gutenberg, D. Nanongkai, T. Saranurak, A. Sidford, and H. Sun, "Fully-dynamic graph sparsifiers against an adaptive adversary," *CoRR*, vol. abs/2004.08432, 2020.

[44] D. Nanongkai and T. Saranurak, "Dynamic spanning forest with worst-case update time: adaptive, las vegas, and $O(n^{1/2-\epsilon})$-time," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2017, pp. 1122–1129.

[45] C. Wulff-Nilsen, "Fully-dynamic minimum spanning forest with improved worst-case update time," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2017, pp. 1130–1143.

[46] D. A. Spielman and S. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC'04: Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*. ACM, 2004, pp. 81–90.

[47] P. M. Vaidya, "Speeding-up linear programming using fast matrix multiplication (extended abstract)," in *FOCS*. IEEE Computer Society, 1989, pp. 332–337.

[48] P. Sankowski, "Dynamic transitive closure via dynamic matrix inverse (extended abstract)," in *FOCS*. IEEE Computer Society, 2004, pp. 509–517.

[49] J. v. d. Brand, D. Nanongkai, and T. Saranurak, "Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds," in *FOCS*. IEEE Computer Society, 2019, pp. 456–480.

[50] Y. T. Lee and A. Sidford, "Efficient inverse maintenance and faster algorithms for linear programming," in *FOCS*. IEEE Computer Society, 2015, pp. 230–249.

[51] D. Adil, R. Kyng, R. Peng, and S. Sachdeva, "Iterative refinement for ℓ_p-norm regression," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SIAM. https://arxiv.org/pdf/1901.06764.pdf, 2019, pp. 1405–1424.

[52] A. C. Gilbert, Y. Li, E. Porat, and M. J. Strauss, "Approximate sparse recovery: optimizing time and measurements," *SIAM Journal on Computing 2012 (A preliminary version of this paper appears in STOC 2010)*, vol. 41, no. 2, pp. 436–453, 2010.

[53] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff, "Fast moment estimation in data streams in optimal space," in *Proceedings of the forty-third annual ACM symposium on Theory of computing (STOC)*, 2011, pp. 745–754.

[54] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Nearly optimal sparse Fourier transform," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, ACM. https://arxiv.org/pdf/1201.2501.pdf, 2012, pp. 563–578.

[55] R. Pagh, "Compressed matrix multiplication," *ACM Transactions on Computation Theory (TOCT)*, vol. 5, no. 3, pp. 1–17, 2013.

[56] K. G. Larsen, J. Nelson, H. L. Nguyen, and M. Thorup, "Heavy hitters via cluster-preserving clustering," in *57th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE. https://arxiv.org/pdf/1604.01357, 2016, pp. 61–70.

[57] M. Kapralov, "Sample efficient estimation and recovery in sparse FFT via isolation on average," in *58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. https://arxiv.org/pdf/1708.04544, 2017.

[58] V. Nakos and Z. Song, "Stronger l2/l2 compressed sensing; without iterating," in *STOC*. https://arxiv.org/pdf/1903.02742, 2019.

[59] V. Nakos, Z. Song, and Z. Wang, "(Nearly) Sample-optimal sparse Fourier transform in any dimension; RIPless and Filterless," in *FOCS*. https://arxiv.org/pdf/1909.11123.pdf, 2019.

[60] A. Bernstein, M. P. Gutenberg, and T. Saranurak, "Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing," 2020, to appear at FOCS'20.

[61] G. Goranci, H. Räcke, T. Saranurak, and Z. Tan, "The expander hierarchy and its applications to dynamic graph algorithms," *CoRR*, vol. abs/2005.02369, 2020. [Online]. Available: https://arxiv.org/abs/2005.02369

[62] R. Kyng, R. Peng, S. Sachdeva, and D. Wang, "Flows in almost

linear time via adaptive preconditioning," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. https://arxiv.org/pdf/1906.10340.pdf, 2019, pp. 902–913.

[63] D. Adil and S. Sachdeva, "Faster p-norm minimizing flows, via smoothed q-norm problems," in *SODA*. SIAM, 2020, pp. 892–910.

[64] P. M. Vaidya, "Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners," Unpublished manuscript, UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation Mineapolis, Tech. Rep., October 1991.

[65] D. A. Spielman and S.-H. Teng, "Solving sparse," in *FOCS'03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. diagonally-dominant linear systems in time $O(m^{1.31})$. In: symmetric, 2003, pp. 416–427.

[66] I. Koutis, G. L. Miller, and R. Peng, "Approaching optimality for solving SDD systems," in *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010, pp. 235–244.

[67] ——, "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)*, 2011, pp. 590–598.

[68] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple," in *STOC'13: Proceedings of the 45th Annual ACM Symposium on the Theory of Computing*. combinatorial algorithm for solving SDD systems in nearly-linear time. In, 2013, pp. 911–920.

[69] Y. T. Lee and A. Sidford, "Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 147–156.

[70] M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao, and S. C. Xu., "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)*, 2014, pp. 343–352.

[71] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, "Sparsified cholesky and multigrid solvers for connection laplacians," in STOC'16: Proceedings of the 48th Annual ACM Symposium on Theory of Computing, 2016.

[72] R. Kyng and S. Sachdeva, "Approximate gaussian elimination for laplacians - fast, sparse, and simple," in *FOCS*. IEEE Computer Society, 2016, pp. 573–582.