Fully Dynamic Electrical Flows: Sparse Maxflow Faster Than Goldberg-Rao

Yu Gao
School of Computer Science
Georgia Institute of Technology
Atlanta, USA
ygao380@gatech.edu

Yang P. Liu
Department of Mathematics
Stanford University
Palo Alto, USA
yangpliu@stanford.edu

Richard Peng
School of Computer Science
Georgia Tech / University of Waterloo
Atlanta, USA / Waterloo, Canada
rpeng@cc.gatech.edu

2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-2055-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS52979.2021.00058

***Abstract*—We give an algorithm for computing exact maximum flows on graphs with m edges and integer capacities in the range $[1, U]$ in $\widetilde{O}(m^{\frac{3}{2}-\frac{1}{328}}\log U)$ time.[1] For sparse graphs with polynomially bounded integer capacities, this is the first improvement over the $\widetilde{O}(m^{1.5}\log U)$ time bound from [Goldberg-Rao JACM '98].**

Our algorithm revolves around dynamically maintaining the augmenting electrical flows at the core of the interior point method based algorithm from [Mądry JACM '16]. This entails designing data structures that, in limited settings, return edges with large electric energy in a graph undergoing resistance updates.

***Keywords*-Maximum flow; Data structures; Interior point methods; Electric flow**

See https://arxiv.org/abs/2101.07233 for the full version of this paper.

I. Introduction

The maxflow problem asks to route the maximum amount of flow between two vertices in a graph such that the flow on any edge is at most its capacity. The efficiency of this problem is well-studied and has numerous applications in scheduling, image processing, and network science [1], [2]. The main result of this paper is a faster exact maxflow algorithm on sparse directed graphs in the weakly polynomial setting, where the runtime depends logarithmically on the capacities.

Theorem 1. *There is an algorithm that on a graph G with m edges and integer capacities in $[1, U]$ computes a maximum flow between vertices s, t in time $\widetilde{O}(m^{\frac{3}{2}-\frac{1}{328}}\log U)$.*

In sparse graphs with polynomially large capacities, this is the first improvement over the classical $O(m^{3/2}\log m\log U)$ time algorithm of Goldberg-Rao [3], which represented the culmination of a long line of work starting from the work of Hopcroft-Karp [4] for bipartite matchings and Karzanov and Even-Tarjan for unit capacity maxflow [5], [6]. Improving over this exponent of 3/2 for graph optimization problems has been intensively studied over the past decade via combinations of continuous optimization and discrete tools.

[1]We use $\widetilde{O}(\cdot)$ to suppress logarithmic factors in m.

This line of work was initiated by Christiano-Kelner-Mądry-Spielman-Teng [7] who gave a $\widetilde{O}(m^{4/3}\epsilon^{-O(1)})$ time algorithm for $(1+\epsilon)$-approximate maxflow. This has since been improved to $m^{1+o(1)}\epsilon^{-O(1)}$ [8], [9] and the focus shifted to achieving improved ϵ dependencies [10], [11] and exact solutions. Towards this, two breakthrough results were the $\widetilde{O}(m^{10/7}U^{1/7})$ time algorithm of Mądry [12], [13] which broke the $3/2$ exponent barrier on unweighted graphs, and the $\widetilde{O}(m\sqrt{n}\log U)$ time algorithm of Lee-Sidford [14], which was an improvement for any dense graph. Since then, these results respectively have been improved to yield algorithms that run in time $m^{4/3+o(1)}U^{1/3}$ [15], [16] and $\widetilde{O}((m+n^{3/2})\log U)$ [17], [18]. However, the $3/2$ exponent of Goldberg-Rao [3] remained the state-of-the-art on sparse capacitated graphs.

Classical approaches to solving maxflow use augmenting paths to construct the final flow. Our algorithm, as well as the recent improvements above, instead computes the maxflow using a sequence of *electric flows*. For resistances $\boldsymbol{r} \in \mathbb{R}^E_{\geq 0}$, the s-t electric flow is the one that routes one unit from s to t while minimizing the quadratic energy:

$$\min_{\boldsymbol{f} \text{ routes one } s\text{-}t \text{ unit}} \sum_{e\in E} \boldsymbol{r}_e \boldsymbol{f}_e^2.$$

Electric flows are induced by vertex potentials, and correspond to solving a linear system in the graph Laplacian. Motivated by this connection with scientific computing, two decades of work on combinatorial preconditioners led to the breakthrough result by Spielman-Teng [19] that Laplacian systems and electrical flows can be computed to high accuracy in $\widetilde{O}(m)$ time.

Our algorithm, as well as the recent faster runtimes for dense graphs [17], [18], are built upon the dynamic processes view of flow augmentations [20], [21] that provided much impetus for the study of dynamic graph data structures. In this view, the final flow is obtained via a sequence of flow modifications, and dynamic tree data structures such as link-cut trees [21], [20] are designed to allow for sublinear time identification and modification of edges that limit flow progress. Concretely, the maxflow is built using a sequence of $\widetilde{O}(\sqrt{m})$ electric flows on graphs with slowly changing resistances. This corresponds to the celebrated interior point

2575-8454/21/$31.00 ©2021 IEEE
DOI 10.1109/FOCS52979.2021.00058

method (IPM henceforth) which shows that linear programs can be solved using $\widetilde{O}(\sqrt{m})$ slowly changing linear system solves [22], [23]. To implement this framework we design data structures that on a graph with dynamic changing resistances:

- Identify all edges e with at least an ϵ^2 fraction of the total electric energy in the electric flow $\boldsymbol{f}$ on a graph with resistances $\boldsymbol{r}$:
$$\boldsymbol{r}_e \boldsymbol{f}_e^2 \geq \epsilon^2 \sum_{e \in E} \boldsymbol{r}_e \boldsymbol{f}_e^2.$$
- Estimate the square root of energy or flow value of an edge up to an additive error of $\pm\epsilon/10 \cdot \sqrt{\sum_{e \in E} \boldsymbol{r}_e \boldsymbol{f}_e^2}$, i.e. a $\epsilon/10$ fraction of the square root of the total electric energy.

Finally, we leverage this data structure along with several modifications to the outer loop to achieve our main result Theorem 1.

A. Key Algorithmic Pieces

At a high level, our algorithm implements an IPM which augments $\widetilde{O}(\sqrt{m})$ electric flows by building a data structure that detects large energy edges in an s-t electric flow on a dynamic graph. In addition to this, our algorithm requires several modifications to the IPM. First, our data structure requires properties specific to s-t electric flows to achieve its guarantees. Consequently, we are forced to design an IPM that only augments via s-t electric flows. On the other hand, a standard IPM alternates between routing electric flows and routing additional electric circulations every step. Second, our data structures are randomized and thus their outputs may affect future inputs when applied within the IPM. This requires delicately modifying our algorithm to bypass this issue. We now give more detailed descriptions of each piece.

Locating high energy edges in s-t electric flows.: Our data structures for dynamic electric flows are based on the interpretation of electrical flow as random walks on the graph [24], which has been used previously for dynamic effective resistances [25], [26]. In our setting we wish to detect edges with at least ϵ^2 fraction of the ℓ_2 electric energy. To achieve this, we use a spectral vertex sparsifier, which approximates the electric flow and potentials on this smaller set of *terminal* vertices. We use this sparsifier as well as additional random walks to maintain the result of an ℓ_2 heavy hitter sketch on the electric flow vector. This allows us to approximately maintain a short sketch vector and thus recover the large entries of the electric flow vector.

Our data structure has several subtleties which affect its interaction with the outer loop. First, it is essential that the electrical flows maintained are s-t to ensure additional stability in our algorithms. s-t electrical flows have additional, sharper, upper bounds on vertex potentials and flow values on edges, which do not hold for electrical flows with more general demands. Secondly, we only maintain an approximate ℓ_2 heavy hitter sketch but argue that this suffices for detection of large energy edges (Lemma 5.1).

IPM with s-t flows.: We must modify the IPM outer loop to interact with our dynamic electric flow data structure described above which fundamentally uses properties specific to s-t flows. The standard IPM [27] which uses electric flows to solve maxflow [28], [13], [15] has both a *progress* phase where an s-t electric flow is augmented, and a *centering* phase where electric circulations are added to slightly fix the flow.

We modify the IPM to only use s-t electric flows to make more than $1/\sqrt{m}$ progress before we pay $\widetilde{O}(m)$ time to center using electric circulations. We leverage two key properties of the method to achieve this. First, we argue that damping the step size of the IPM causes errors to accumulate more slowly. This allows us to use several s-t electric flow steps (maintained in sublinear time by data structures) as opposed to flows with general demands before a centering step. Also, to argue this formally we use the fact that the resistances are multiplicatively stable to within a polynomial factor of the number of steps of standard size $1/\sqrt{m}$.

Randomness in data structures and adaptivity.: Because we are applying randomized data structures inside an outer loop, their outputs may affect future inputs. In the literature, this is referred to as an *adaptive adversary*. On the other hand, our data structures naïvely only work against *oblivious adversaries*, where the inputs are independent of the outputs and randomness of the data structure.

We handle these issues by carefully designing our data structures and outer loop to not leak randomness between components, instead of making our data structures deterministic or work against adaptive adversaries in general. We start by breaking the data structure into a LOCATOR and a CHECKER, based on ideas from [29]. The LOCATOR returns a superset that contains all edges with large energy with high probability, and the CHECKER independently estimates the energies of those edges to decide whether to update them. This way, the randomness of LOCATOR does not affect its inputs. However, the outputs of CHECKER may affect its inputs. Now, we leverage that the sequence of flows encountered during the IPM outer loop are almost deterministic, and there are only a few iterations between deterministic instances. This way, we can use a separate CHECKER for each of these iterations before resetting every CHECKER to the deterministic instance.

B. Heuristic Runtime Calculation

The following key properties of the IPM outer loop are necessary to understand why a sublinear time data structure suffices to achieve a $m^{3/2-\Omega(1)}$ time algorithm for capacitated maxflow.

1) Computing electric flows on graphs whose resistances are within $1\pm\gamma$ of the true resistances suffices to make $1/\sqrt{m}$ progress (for some parameter $\gamma = \widetilde{\Omega}(1)$).

2) The resistances change slowly multiplicatively throughout the course of the algorithm. In fact, at most $\widetilde{O}(T^2\gamma^{-2})$ edges have their resistances change by at least $1 \pm \gamma$ multiplicatively over T steps of the method for any γ (Lemma 6.6). In particular, over all $\widetilde{O}(\sqrt{m})$ iterations of electric flow computation, each edge's resistance changes $\widetilde{O}(1)$ times on average.
3) The resistance of an edge is approximately the inverse of its residual capacity squared. This way, an edge's resistance changes significantly if the electric energy of the edge is large in the computed electric flows.

If we have a data structure which detects edges with large energies in $m^{1-\eta}$ amortized time per edge for some constant $\eta > 0$, then we can leverage it along with the above facts to design the following algorithm. We take steps in batches of size k, after which we pay $\widetilde{O}(m)$ time to fix and recenter our flow to find the true underlying resistances. During each batch, we use the data structure to detect all edges whose resistance changed by more than $1 + \widetilde{\Omega}(1)$ multiplicatively, and return their resistances.

Now we estimate the runtime of this algorithm. The cost of recentering is $O(m) \cdot \widetilde{O}(m^{1/2}/k) = \widetilde{O}(m^{3/2}/k)$, as there are $\sqrt{m}$ total steps and we recenter every k iterations. Also, by the second item above that at most $\widetilde{O}(k^2)$ edges have their resistances change significantly during a batch, so the data structure takes $\widetilde{O}(m^{1-\eta}k^2)$ time per batch. The total time used by the data structure is therefore $\widetilde{O}(m^{1-\eta}k^2) \cdot \widetilde{O}(\sqrt{m}/k) = \widetilde{O}(m^{3/2-\eta}k)$. Taking $k = m^{\eta/2}$ gives a final runtime of $\widetilde{O}(m^{3/2-\eta/2})$, which is less than $m^{3/2}$ as desired. The tradeoffs in our algorithms are significantly higher and more complicated in reality: we have higher exponents on the batch size k due to compounding errors in the method, and we have additional layers of intermediate rebuilds. Nonetheless, the final tradeoffs by which we obtain Theorem 1 are still similar in spirit.

C. Related Work and Discussion

There is a long history of work on the maximum flow problem, as well as work related to each of our key algorithmic pieces in Section I-A: dynamic graph data structures, IPMs in the context of data structures, and random and adaptivity in data structures.

Our discussion below focuses on algorithms whose capacity dependence is logarithmic (weakly polynomial). The weakly polynomial setting also is equivalent to the setting where the edge capacities are positive real numbers, and we wish to compute an ϵ-approximate solution in runtime depending on $\log(1/\epsilon)$. In the strongly polynomial setting, where the algorithm runtime has no capacity dependence, following early work of [30], [20], [21], the best known maxflow runtime is $O(mn)$ and $O(n^2/\log n)$ when $m = O(n)$ [31], [32].

Maxflow Algorithms: Network flow problems are widely studied in operations research, theoretical computer science, and optimization [2]. Among the many variants, the capacitated maxflow problem captures key features of both combinatorial graph algorithms and numerical optimization routines. As a result, it has an extensive history starting from the work of Dinic and Edmonds-Karp [33], [34]. The seminal work by Edmonds-Karp [34] presented two algorithms: an $O(n^2m)$ strongly polynomial time algorithm by finding shortest augmenting paths, and an $O(m^2 \log U)$ weakly polynomial time algorithm based on finding bottleneck shortest paths. Improving these algorithms provided motivation for dynamic tree data structures [20], dual algorithms [35], and numerical primitives such as scaling [3]. These progress culminated in a runtime of $\widetilde{O}(\min(m^{3/2}, mn^{2/3}) \log U)$: for more details, we refer the reader to the review by Goldberg and Tarjan [2].

In the two decades since Goldberg-Rao [3], all improvements on the exact maximum flow problem rely on continuous optimization techniques. These include the $\widetilde{O}(m\sqrt{n} \log U)$ runtime of Lee-Sidford [14], and several results culminating in a $\widetilde{O}(m/\epsilon)$ runtime for ϵ-approximate maxflow on undirected graphs [7], [8], [9], [36], [37], [11]. Additionally, a line of work [12], [13], [15] achieving a $m^{4/3+o(1)}$ runtime in uncapacitated graphs [16] by using weight changes and ℓ_p-norm flows [38] to eliminate high energy edges, as opposed to our approach of using data structures to detect them. Recently, approaches that combine interior point methods (IPMs) with graphical data structures achieved a $\widetilde{O}((m + n^{3/2}) \log U)$ runtime for maxflow [17], [18]. In this way, the bound of Golberg-Rao [3] has been improved in higher error approximate settings (on undirected graphs), for uncapacitated graphs, and for dense capacitated graphs. However, our result is the first to show an improvement for exact maxflow in the weakly polynomial parameter regime central to the line of work spanning from Edmonds-Karp [34] to Goldberg-Rao [3]: sparse directed graphs with polynomially bounded capacities.

Data Structures for IPMs.: Starting from early work of Karmarkar [22] and Vaidya [23], several results leverage the fact that the linear systems resulting from IPMs are slowly changing, and that only approximate solutions are needed to implement the method. In this way, data structures for efficiently maintaining the inverse of dynamically changing linear systems have been used to speed up IPMs for linear programming [39], [40], [41], [42], [43], [17] and recently semidefinite programming [44]. Additionally, our algorithm uses the fact the multiplicative change in resistances is at most polynomial in the number of steps taken. While this type of result was previously known[2], we are not aware of other IPM analyses that use this fact.

In the graphical setting of maxflow, this corresponds to dynamically maintaining electric flows in a graph with

[2]Personal communication with Yin Tat Lee and Aaron Sidford [45], also similar in spirit to [14, Lemma 67].

changing resistances. Our result is heavily motivated by the recent [17] and its follow-up [18] which obtained $\widetilde{O}(m+n^{1.5})$ type running times for flow problems. The flow-based version of these results use dynamic sparsification algorithms to maintain approximate electric flows in $\widetilde{O}(n)$ instead of $\widetilde{O}(m)$ time per iteration. Additionally these works required several other techniques to achieve their runtimes, including robust central paths/different measures of centrality, and weighted barriers. While we do not use these pieces in our algorithm, we are optimistic that understanding how to apply these techniques could improve the runtime of our method.

Also, ℓ_2 heavy hitters are used in [17], [18] and our algorithms; however, we open up the standard statement of ℓ_2 heavy hitter [46] to prove that an approximate matrix-vector product suffices to implement the heavy hitter data structure (Lemma 5.1). Critically, we treat the heavy hitter sketch matrix as demands on which we compute electric flows which allows for interaction with random walks and spectral vertex sparsification.

Dynamic graph data structures.: The data structures we use to make sublinear time steps in interior point methods broadly belong to data structures maintaining approximations to optimization problems in dynamically changing graphs [47], [48], [49], [50], [51], [52], [53], [26]. Our maintenance of electrical flows is most directly related to dynamic effective resistance data structures [54], [55], [25], [26]. In particular, they heavily rely on dynamic vertex sparsifiers, which by itself has also received significant attention in data structures [56], [57], [58]. In particular, our sublinear runtime comes in part from maintaining a spectral vertex sparsifier onto a smaller vertex subset.

Adaptivity and randomness.: Our data structures are randomized, and are accessed in an adaptive manner: queries to it may depend on its own output. While there has been much recent work on making randomized sparsification based data structures more resilient against such adaptive inputs [59], [60], [61], [62], [63], [64], our approach at a high level bypasses most of these issues because the (non-robust, unweighted) central path of IPMs is a fixed object. In this way, our randomized data structures are essentially pseudo-deterministic [65], [66]: while the algorithm is randomized, the output is the same with high probability. Additionally, the top-level interactions of our randomized components involve calling one data structure inside another to hide randomness. This has much in common with the randomized approximate min-degree algorithm from [29].

D. General Notation and Conventions

We use plaintext to denote scalars, bold lower case for vectors, and bold upper case for matrices. A glossary of variables and parameters is given in Appendix B. We will use the $\widehat{\cdot}$ notation to denote a later, modified, copy of a variable. As our update steps are approximate, we will also use the $\widetilde{\cdot}$ notation to denote approximate/error carrying versions of true variables.

We use $\widetilde{O}(\cdot)$ to suppress logarithmic factors in m and $\widetilde{\Omega}(\cdot)$ to suppress the inverse logarithmic factors in m. We let $\mathbf{0}, \mathbf{1} \in \mathbb{R}^n$ denote the all zeroes/ones vectors respectively. For vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ we let $(\boldsymbol{x} \circ \boldsymbol{y})_i \stackrel{\text{def}}{=} \boldsymbol{x}_i \boldsymbol{y}_i$. When context is clear, we also use $\frac{\boldsymbol{x}}{\boldsymbol{y}}$ to denote the entry-wise division of two vectors, that is $\left(\frac{\boldsymbol{x}}{\boldsymbol{y}}\right)_i \stackrel{\text{def}}{=} \frac{\boldsymbol{x}_i}{\boldsymbol{y}_i}$. We use $|\boldsymbol{x}|$ and $|\mathbf{Y}|$ to denote the entry-wise absolute values of vector $\boldsymbol{x}$ and matrix $\mathbf{Y}$.

Instead of tracking explicit constants in our parameters, we sometimes use c and C to denote sufficiently small (respectively large) absolute constants. E.g., for a parameter $k > 1$, we write $\epsilon = ck^{-6}$ to denote that there is a constant c where $\epsilon = ck^{-6}$, and we will set c later to be sufficiently small. c and C may denote different constants in different places. We use "with high probability" or "w.h.p." to mean with probability at least $1 - n^{-10}$.

We say that a symmetric matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is positive semidefinite (psd) if $\boldsymbol{x}^\top \mathbf{A} \boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$. For psd matrices $\mathbf{A}, \mathbf{B}$ we write $\mathbf{A} \preceq \mathbf{B}$ if $\mathbf{B} - \mathbf{A}$ is psd. For positive real numbers a, b we write $a \approx_\gamma b$ to denote $\exp(-\gamma)b \leq a \leq \exp(\gamma)b$. For psd matrices $\mathbf{A}, \mathbf{B}$ we write $\mathbf{A} \approx_\gamma \mathbf{B}$ if $\exp(-\gamma)\mathbf{B} \preceq \mathbf{A} \preceq \exp(\gamma)\mathbf{B}$.

E. Organization of Paper

Due to space constraints, we will only provide an overview of our result in Section II. We elaborate on each major piece of our algorithm introduced in Section I-A: dynamic electric flow data structures, our modified IPM outer loop, and handling of randomness and adaptive adversaries. Then in Section III we give the linear algebraic formulation of the maximum flow problem. We then introduce the key notion of electric flows and its relationship with linear systems and random walks. The full version of the paper is at arXiv 2101.07233.

II. Overview of Approach

In this section we elaborate on the key pieces of our approach described in Section I-A: dynamic electrical flow data structures (Section II-A), an interior point method for maxflow using this data structure (Section II-B), and how to handle issues with randomness and adaptive adversaries (Section II-C).

A. Overview of Locator for Dynamic Electric Flows

Recall the dynamic electric flow problem we solve. For a graph $G = (V, E)$ with changing resistances $\boldsymbol{r} \in \mathbb{R}^E_{\geq 0}$ such that the energy of the electric flow $\boldsymbol{f}$ is at most 1 always, i.e. $\sum_{e \in E} \boldsymbol{r}_e \boldsymbol{f}_e^2 \leq 1$, return a set of at most $\widetilde{O}(\epsilon^{-2})$ edges $S \subseteq E$ that contains all edges with energy at least ϵ^2, i.e. $\boldsymbol{r}_e \boldsymbol{f}_e^2 \geq \epsilon^2$ for $e \in S$. We wish to solve this in amortized sublinear time per resistance update.

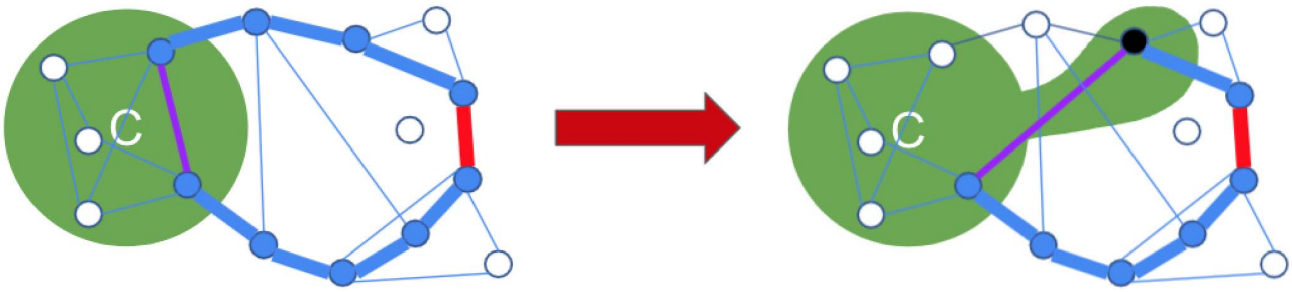

Figure 1. Shortcutting random walks from the red edge to a terminal set C when under the insertion of the black vertex to C.

At a high level, our approach is based on the vertex sparsification view towards data structures. In this view, we achieve sublinear runtimes by maintaining an object onto a smaller subset of terminal vertices $C \subseteq V$ that approximately preserve the desired property in our data structure. For example, in our setting we will leverage *spectral vertex sparsifiers* that maintain the electrical properties of the graph onto the set of terminals, such as pairwise effective resistances. Alternatively, this can be viewed as maintaining the spectral properties of the inverse of the graph Laplacian (and is known as the *Schur complement*). In our algorithms, the set C will increases in size throughout our data structure to ensure that edge changes happen within C. Hence the focus is on maintaining properties onto C while new vertices are added to it throughout the algorithm.

We detect edges with large electric energies by first setting up a linear ℓ_2 heavy hitter sketch [46] against the energy vector $\mathbf{R}^{1/2}\boldsymbol{f}$, where $\mathbf{R}$ is the diagonal matrix of resistances and $\boldsymbol{f}$ is the electric flow. We then approximately maintain the sketch using random walks and spectral vertex sparsifiers. At a high level, an ℓ_2 heavy hitter sketch works by estimating the total ℓ_2-norm / energy of various edge subsets using Johnson-Lindenstrauss sketches up to accuracy ϵ. In this way, for $\widetilde{O}(\epsilon^{-2})$ sketch vectors $\boldsymbol{q} \in \{-1, 0, 1\}^m$, we must maintain the quantity $\langle \boldsymbol{q}, \mathbf{R}^{1/2}\boldsymbol{f}\rangle$. Now we relate the electric flow to the electric potentials $\boldsymbol{\phi}$ using Ohm's law: for any edge $e = (u, v)$ we have $\boldsymbol{f}_e = (\boldsymbol{\phi}_u - \boldsymbol{\phi}_v)/\boldsymbol{r}_e$. Written algebraically, this is $\boldsymbol{f} = \mathbf{R}^{-1}\mathbf{B}\boldsymbol{\phi}$ where $\mathbf{B}$ is the (unweighted) edge-vertex incidence matrix of the graph G. Plugging this into our previous formula gives us

$$\langle \boldsymbol{q}, \mathbf{R}^{1/2}\boldsymbol{f}\rangle = \langle \boldsymbol{q}, \mathbf{R}^{-1/2}\mathbf{B}\boldsymbol{\phi}\rangle = \langle \mathbf{B}^\top \mathbf{R}^{-1/2}\boldsymbol{q}, \boldsymbol{\phi}\rangle.$$

For simplicity we now let $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{B}^\top \mathbf{R}^{-1/2}\boldsymbol{q}$. Intuitively, the sketch vector $\boldsymbol{q}$ is inducing a demand $\boldsymbol{d}$ on the vertices which we now want to dot against the vertex potentials $\boldsymbol{\phi}$.

Now our goal is to use a smaller set of terminal vertices C to estimate the quantity $\langle \boldsymbol{d}, \boldsymbol{\phi}\rangle$. We achieve this by leveraging the fact that we can recover potentials outside C by *harmonically extending* the potentials restricted to C: $\boldsymbol{\phi}_C$. Precisely, the potential $\boldsymbol{\phi}_v$ at vertex $v \neq s, t$ is the average of its neighbors, weighted proportional to inverse resistances. Equivalently, starting a random walk at a vertex $v \notin C$ and taking exit edges proportional to inverse of resistances is a martingale (preserves mean) on the potentials. In this way we can write $\boldsymbol{\phi} = \mathcal{H}\boldsymbol{\phi}_C$ where $\mathcal{H} \in \mathbb{R}^{V(G)\times C}$ is this extension operator. Hence

$$\langle \boldsymbol{d}, \boldsymbol{\phi}\rangle = \langle \boldsymbol{d}, \mathcal{H}\boldsymbol{\phi}_C\rangle = \langle \mathcal{H}^\top \boldsymbol{d}, \boldsymbol{\phi}_C\rangle.$$

To compute this final quantity we must maintain $\mathcal{H}^\top \boldsymbol{d}$ and $\boldsymbol{\phi}_C$ efficiently in sublinear time. For the former, given our random walk interpretation of $\mathcal{H}$, we may interpret $\mathcal{H}^\top \boldsymbol{d}$ as the vector given by "projecting" $\boldsymbol{d}$ onto the terminal set C via random walks, and we write $\boldsymbol{\pi}^C(\boldsymbol{d}) \stackrel{\text{def}}{=} \mathcal{H}^\top \boldsymbol{d}$ (Definition 5.5). In other words, the demand vector $\boldsymbol{d}$ is distributed onto C based on the probabilities that random walks from vertices v hit C for the first time. This interpretation of $\mathcal{H}^\top \boldsymbol{d}$ allows us to build random walks to simulate the changes to this vector under the terminal set C growing in size. For the latter, we maintain $\boldsymbol{\phi}_C$ by using the approximate spectral vertex sparsifier of [25] which approximately maintains the Laplacian inverse on C and hence the potentials. This construction is also based on running random walks from edges outside C until they hit C.

We briefly elaborate on how resistance updates affect the terminal set C and the random walks we maintain. We start by initializing C to be a random set of size βm. (The reader can imagine $\beta = m^{-0.01}$ so that $|C|$ is sublinear.) We run random walks from each edge or vertex until it hits C. These walks are short, specifically visiting $\widetilde{O}(\beta^{-1})$ distinct vertices with high probability, because C was chosen to be βm random vertices. Now, in general when the resistance of an edge $e = (u, v)$ is changed we add both endpoints u, v of e to C. Now the edge e will be contained fully inside C so we can directly perform the resistance change. However we must update our random walks due to C changing. To do this we shortcut each random walk we computed to when it hit the larger set C and update the necessary properties. A depiction of this process is given in Fig. 1.

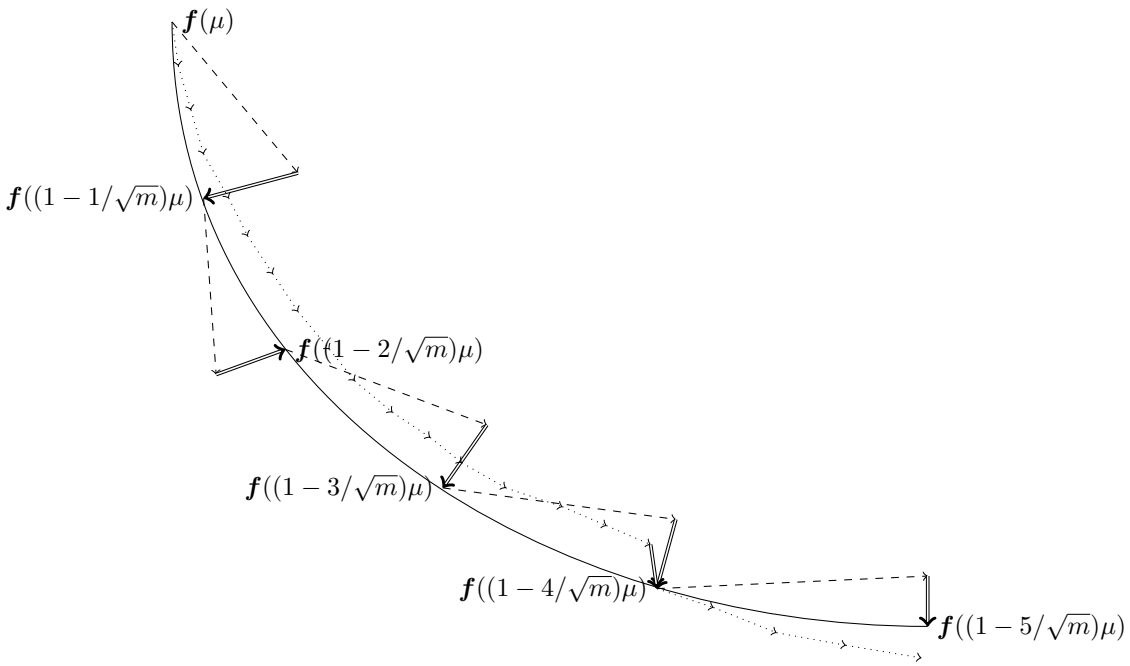

Figure 2. Our algorithm is split into $\widetilde{O}(\sqrt{m}/k)$ *batches* of steps. Each batch is split into $\widetilde{O}(k^4)$ smaller steps to reduce error so that we can recenter in $\widetilde{O}(m)$ time at the end of each batch (Lemma 6.2). Each small step implements an approximate electric flow using data structures. . In the diagram, the dashed line is standard path following which recenters (depicted by the double arrows) after each step of size $1/\sqrt{m}$. The dotted line is the finer steps which takes more total steps but only recenters after $4/\sqrt{m}$ progress.

To conclude, we describe some difficulties with the approach described above, specifically pertaining to maintaining the projected demand $\mathcal{H}^\top \boldsymbol{d} = \boldsymbol{\pi}^C(\boldsymbol{d})$. The first concern is that entries of $\boldsymbol{d} = \mathbf{B}^\top \mathbf{R}^{-1/2}\boldsymbol{q}$ are too large if some edge e has resistance $\boldsymbol{r}_e$ close to 0 (as then $\boldsymbol{r}_e^{-1/2}$ is large). We handle this with the observation that edges with small resistances cannot have large energies in an s-t electric flow (Lemma 5.2), so we can restrict our heavy hitter sketch to edges with sufficiently large resistances. Also, naïvely estimating the projection $\boldsymbol{\pi}^C(\boldsymbol{d})$ with random walks from each vertex accumulates too much variance because the demand vector $\boldsymbol{d}$ is dense. Instead we exactly compute $\boldsymbol{\pi}^C(\boldsymbol{d})$ by solving a linear system to start, and we estimate the *change* in this vector under insertions to C by locally sampling random walks from the inserted vertex (Fact 5.7). Finally, we periodically recalculate this vector to ensure that error does not accumulate.

B. Overview of Interior Point Method

In this section we formalize the outer loop that our algorithm uses to argue that $\widetilde{O}(\sqrt{m})$ approximate electric flow computations suffice to compute a maxflow. We assume that the graph G is undirected [67], [12] and that we know the optimal maxflow value F^* by a standard binary search reduction. Given this, the *central path* is a sequence of flows $\boldsymbol{f}(\mu)$ for $\mu \in (0, F^*]$ defined by the minimizers of a logarithmic barrier potential:

$$\boldsymbol{f}(\mu) \stackrel{\text{def}}{=} \operatorname*{arg\,min}_{\mathbf{B}^\top \boldsymbol{f} = (F^* - \mu)\boldsymbol{\chi}_{st}} V(\boldsymbol{f})$$
$$\text{for} \quad V(\boldsymbol{f}) \stackrel{\text{def}}{=} \sum_{e \in E} -\log(\boldsymbol{u}_e - \boldsymbol{f}_e) - \log(\boldsymbol{u}_e + \boldsymbol{f}_e). \tag{1}$$

Note that for $\mu = F^*$ that $\boldsymbol{f}(F^*) = \mathbf{0}$, the zero flow. Starting there, the goal of our algorithm is to follow this central path by slowly decreasing μ towards 0 while computing the flows $\boldsymbol{f}(\mu)$ along the way. While μ never equals 0 exactly, the flows $\boldsymbol{f}(\mu)$ approach the maximum flow as μ approaches 0. We want to emphasize that the sequence of flows encountered by the algorithm along the way is *deterministic* in this sense, as the minimizer of the convex problem (1) is unique.

We remark that this central path (which is adapted from [13]) differs from the more standard central path used to solve mincost flow with cost $\mathbf{c}^\top \boldsymbol{f}$. While this version can also work by setting $\mathbf{c}$ as a large negative cost on an s-t edge, we choose to work with our formulation because the intuition that we are augmenting by s-t electric flows is useful for our data structure based approach.

Now consider trying to decrease the path parameter μ to $\widehat{\mu} < \mu$ starting from the current central path flow $\boldsymbol{f}(\mu)$. Then we wish compute a flow $\Delta \boldsymbol{f}$ which routes $\mu - \widehat{\mu}$ units from s to t such that adding $\Delta \boldsymbol{f}$ to our current flow $\boldsymbol{f}(\mu)$ gets to

the minimizer of (1) for $\widehat{\mu}$, i.e. $\boldsymbol{f}(\widehat{\mu}) = \boldsymbol{f}(\mu) + \Delta\boldsymbol{f}$. While directly computing $\Delta\boldsymbol{f}$ exactly is more difficult, one can show that up to a first order approximation, $\Delta\boldsymbol{f}$ is given by the electric flow that routes $\mu - \widehat{\mu}$ units from s to t, with resistances given by $\boldsymbol{r}_e = (\boldsymbol{u}_e - \boldsymbol{f}(\mu)_e)^{-2} + (\boldsymbol{u}_e + \boldsymbol{f}(\mu)_e)^{-2}$.

To handle the approximations induced by using s-t electric flows, we require another fact: if we are able to calculate a flow $\widetilde{\boldsymbol{f}}$ that is "close" to $\boldsymbol{f}(\widehat{\mu})$ on all edges, then we can compute $\boldsymbol{f}(\widehat{\mu})$ exactly using $\widetilde{O}(m)$ additional time (by computing additional electric circulations). Here, $\widetilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}(\widehat{\mu})$ if all residual capacities differ by at most a multiplicative 1.1 factor (Lemma 6.2). Now, one can show that if $\widehat{\mu} = (1 - c/\sqrt{m})\mu$ for a small constant c, $\Delta\widetilde{\boldsymbol{f}}$ is the electric s-t flow routing $\mu - \widehat{\mu}$ units (even with approximate resistances), and $\widetilde{\boldsymbol{f}} = \boldsymbol{f}(\mu) + \Delta\widetilde{\boldsymbol{f}}$, then $\widetilde{\boldsymbol{f}}$ is close to $\boldsymbol{f}(\widehat{\mu})$. Thus, this gives a method that terminates in $\widetilde{O}(\sqrt{m})$ iterations and $\widetilde{O}(m^{3/2})$ time.

To achieve a $m^{3/2-\Omega(1)}$ time maxflow algorithm using IPMs we must be able to to decrease μ to $\widehat{\mu} = (1 - k/\sqrt{m})\mu$ for some $k = m^{\Omega(1)}$ in $\widetilde{O}(m)$ amortized time. This would achieve a $\widetilde{O}(m^{3/2}/k)$ time algorithm. Directly adding the electric flow routing $\mu - \widehat{\mu} = k\mu/\sqrt{m}$ units from s to t accumulates too much error. We instead split this step into a batch of smaller steps, each which is an electric flow routing $(\widehat{\mu} - \mu)/k^4 = \mu/(k^3\sqrt{m})$ units. We show in Section 6 that because the electric flow is the first order approximation to the change in the central path, and because residual capacities are stable within a $O(k^2)$ factor during the step (Lemma 6.7), that this sufficiently reduces error.

Now our method approximately implements each of the smaller steps in the batch using the data structure described in Section II-A. We would like to emphasize again that even though the flows encountered during the small steps within a batch are randomized, we can pay $\widetilde{O}(m)$ at the end of each batch to move our flow back to the exact minimizer of (1) so that it is deterministic. A depiction of the batches, splits into small steps, and recentering is given in Fig. 2.

C. Overview of Handling of Randomness

In this section we explain how to adapt our IPM outer loop and data structures to ensure that randomness in the data structures used to produce outputs does not affect the distribution of future inputs to itself. To this end, let us recall our setup described in the above Sections II-A and II-B. We have a heavy hitter data structure which returns a set S of edges that contains all edges with an ϵ^2 fraction of the energy, and estimates their energies up to additive error.

Our first step towards addressing the randomness issue is to decouple the data structure. We split it into two parts: the first part which returns edges with large energies (LOCATOR), and a separate part which estimates again the energies of returned edges (CHECKER). Our reasons for doing this are twofold – it both helps with reasoning about where randomness arises in the algorithm, and provides mild runtime improvements.

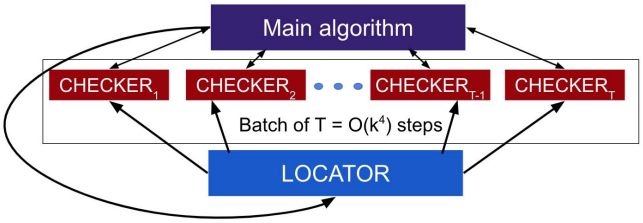

Figure 3. Within a batch of steps, during the i-th step the LOCATOR passes an edge subset to the i-th CHECKER. This communicates with the outside loop which passes updates to later CHECKERs as well as the LOCATOR.

In this new setup the LOCATOR corresponds to the heavy hitter, and returns a set of at most $O(\epsilon^{-2})$ edges that contains all edges with at least $\epsilon^2/10$ fraction of the electric energy. This set is fed to the CHECKER which independently estimates the amount of electric flow on that edge for each edge in the set. The CHECKER wishes to accept any edge with at least ϵ^2 fraction of the electric energy and to estimate its flow value. Due to our IPM setup, these data structures are used within an outer loop consisting of $\widetilde{O}(\sqrt{m}/k)$ batches of steps, each which is split into k^4 smaller steps. After each batch, the algorithm perfectly moves back to the minimizer of (1) in $\widetilde{O}(m)$ time and updates resistances. For each smaller step within the batch, we call LOCATOR and CHECKER together to find edges with large energies / flows, and hence must have their resistances updated. A depiction of the interactions between the LOCATOR, CHECKER data structures, and the algorithmic outer loop is given in Fig. 3.

Note that the flow that we maintain is deterministically equal to the minimizer of (1) at the start and end of each batch, so we can essentially update both the data structures deterministically. Hence we focus on ensuring the property that our data structures outputs do not affect futures inputs or states during the k^4 steps within a batch. We first describe why LOCATOR can be assumed to be against oblivious adversaries, i.e. inputs are independent of the randomness. To understand why this is the case, consider the algorithm that does not use the LOCATOR data structure at all, and instead uses the CHECKER to independently estimate the flow on every single edge and decides whether it believes the edge to have high energy. Clearly this algorithm is valid. We argue that using the LOCATOR data structure simulates this algorithm that checks every edge. Indeed, we may set the thresholds for LOCATOR so that any edge that CHECKER decides to update with non-negligible probability is included in the set of edges LOCATOR returns with high probability. In this way, the outputs of LOCATOR do not affect its future inputs as long as CHECKER is checking each edge independently.

While this explains why LOCATOR may operate against oblivious adversaries, the same is unclear for CHECKER. Indeed, different flow value estimates for an edge e affect whether the resistance of e is changed, and this can affect the internal state of CHECKER itself even during the same batch. To handle this, we actually construct k^4 independent CHECKER data structures, which we call $D_i^{(\text{chk})}$ for $i \in [k^4]$, one for each small step within a batch. We use $D_i^{(\text{chk})}$ to handle the set of edges that LOCATOR returns at small step i out of k^4. After this step, we stop updating $D_i^{(\text{chk})}$ until the end of the batch. At that time, we roll back all changes made to $D_i^{(\text{chk})}$ during the batch and then deterministically update its state to the new exact minimizer flow we compute. In this way, we can argue that the outputs of $D_i^{(\text{chk})}$ can only affect inputs of $D_j^{(\text{chk})}$ for $j > i$, so no $D_i^{(\text{chk})}$ has inputs affecting itself. In this way, we may assume that each $D_i^{(\text{chk})}$ actually operates against oblivious adversaries.

III. PRELIMINARIES: MAXFLOW AND ELECTRICAL FLOWS

We start by formally defining the maxflow problem, the electrical flow subroutine, and key objects for representing both problems. We will use $G = (V, E)$ to denote graphs, $\boldsymbol{u}$ to denote edge capacities, and $\boldsymbol{f}$ to denote flows. We will also use $\boldsymbol{deg}_v$ to denote the combinatorial/unweighted degree of vertex v in G, that is, $\boldsymbol{deg}_v = |\{e \in E \mid e \ni v\}|$.

A. Maxflow

One can reduce directed maxflow to undirected maxflow with linear time overhead [67], [12], so we assume our graph $G = (V, E)$ is undirected throughout. m will be the number $|E|$ of edges and n will be the number $|V|$ of vertices. We assume $m \leq n^2$.

We also assume that G is connected and has at least two vertices and one edge. Thus, each vertex of G has at least one edge incident to it. By standard capacity scaling techniques [68] we may assume that $U = \text{poly}(m)$ throughout this paper. Also, we assume we know the optimal numbers of units $F^* = \text{poly}(m)$, as our algorithm works for any underestimate. Furthermore, our algorithm actually works for general demand maxflows, as we can add a super source s and super sink t to accumulate to positive (respectively negative) demands on vertices.

We can then formalize the decision version of maxflow via linear algebra. Define $\mathbf{B}$ to be the edge-vertex incidence matrix of G:

$$\mathbf{B} \in \mathbb{R}^{E \times V} \qquad \mathbf{B}_{eu} = \begin{cases} 1 & \text{if } u \text{ is the head of } e \\ -1 & \text{if } u \text{ is the tail of } e \\ 0 & \text{otherwise} \end{cases}$$

and $\boldsymbol{\chi}_{st}$ to be the indicator vector with -1 at source s, 1 at sink t and 0 everywhere else. Routing F^* units of flow from s to t then becomes finding $\boldsymbol{f} \in \mathbb{R}^E$ such that

$$\mathbf{B}^\top \boldsymbol{f} = F^* \boldsymbol{\chi}_{st} \qquad \text{and} \qquad -\boldsymbol{u} \leq \boldsymbol{f} \leq \boldsymbol{u}.$$

B. Electrical Flows

Electrical flows are ℓ_2-minimization analogs of maxflow, and underlie all interior point method oriented approaches to high-accuracy maxflow [28], [12], [14], [13], [69].

We use the term *demand vector* for any vector $\boldsymbol{d}$ such that $\boldsymbol{d} \in \mathbb{R}^n$ and $\mathbf{1}^\top \boldsymbol{d} = 0$. We let $\boldsymbol{r} \in \mathbb{R}^E$ be the vector of resistances: $\boldsymbol{r}_e$ denotes the resistance of edge e. For a demand vector $\boldsymbol{d}$, and the vector of resistances $\boldsymbol{r}$, the electrical flow problem is

$$\min_{\boldsymbol{f}: \mathbf{B}^\top \boldsymbol{f} = \boldsymbol{d}} \sum_e \boldsymbol{r}_e \boldsymbol{f}_e^2.$$

Here the energy function can be further abbreviated using the norm notation: by letting $\mathbf{R}$ denote the diagonal matrix with $\boldsymbol{r}$ on the diagonal, the energy can be written as $\|\boldsymbol{f}\|_{\mathbf{R}}^2$. The quadratic minimization nature of this problem means its solution, or the optimal electrical flow, has a linear algebraic closed form, specifically

$$\boldsymbol{f} = \mathbf{R}^{-1}\mathbf{B}\left(\mathbf{B}^\top \mathbf{R}^{-1}\mathbf{B}\right)^\dagger \boldsymbol{d},$$

where $\dagger$ denotes the Moore-Penrose pseudoinverse. The matrix $\mathbf{B}^\top \mathbf{R}^{-1}\mathbf{B}$ is important on its own, and is known as the graph Laplacian matrix, $\mathbf{L} = \mathbf{B}^\top \mathbf{R}^{-1}\mathbf{B}$. Laplacian systems can be solved to high accuracy in nearly linear time [70], [71], [72], [73], [74], [75], [76]. The resulting solution vector on the vertices also have natural interpretations as voltages that induce the electrical flow [24]. Specifically, for the voltages

$$\boldsymbol{\phi} = \mathbf{L}^\dagger \boldsymbol{d} = \left(\mathbf{B}^\top \mathbf{R}^{-1}\mathbf{B}\right)^\dagger \boldsymbol{d}$$

the flow is given by *Ohm's Law*:

$$\boldsymbol{f}_e = \frac{\boldsymbol{\phi}_u - \boldsymbol{\phi}_v}{\boldsymbol{r}_e} \qquad \text{for all } e = (uv).$$

Both this flow, and the voltages, can be computed to high accuracy in nearly-linear time using Laplacian solvers [70].

Theorem 2. *Let G be a graph with n vertices and m edges. Let $\boldsymbol{r} \in \mathbb{R}^E_{>0}$ denote edge resistances. For any demand vector $\boldsymbol{d}$ and $\epsilon > 0$ there is an algorithm which computes in $\widetilde{O}(m \log \epsilon^{-1})$ time* potentials $\boldsymbol{\phi}$ *such that $\|\boldsymbol{\phi} - \boldsymbol{\phi}^*\|_{\mathbf{L}} \leq \epsilon \|\boldsymbol{\phi}^*\|_{\mathbf{L}}$, where $\mathbf{L} = \mathbf{B}^\top \mathbf{R}^{-1}\mathbf{B}$ is the Laplacian of G, and $\boldsymbol{\phi}^* = \mathbf{L}^\dagger \boldsymbol{d}$ are the true potentials determined by the resistances $\boldsymbol{r}$.*

Critical to our data structures are the intuition of electrical flows as random walks. Specifically, that the unit electrical flow from s to t is the expected trajectory of the random

walk from s to t, with cancellations, where from vertex u we go to $v \sim u$ with probability

$$\frac{\boldsymbol{r}_{uv}^{-1}}{\sum_{w \sim u} \boldsymbol{r}_{uw}^{-1}}$$

where the reciprocal of resistances, conductance, plays a role analogous to the weight of edges. Many of our intuitions and notations have overlaps with the electrical flow based analyses of sandpile processes [77]. For a more systematic exposition, we refer the reader to the excellent monograph by Doyle and Snell [24].

ACKNOWLEDGMENTS

Yang P. Liu was supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program. Richard Peng is supported by the National Science Foundation (NSF) under Grant No. 1846218.

We thank Jan van den Brand, Arun Jambulapati, Yin Tat Lee, and Aaron Sidford for helpful discussions and pointing out typos in an earlier version of this manuscript. We especially thank Aaron Sidford for discussions during which an error in the handling of adaptivity and randomness in the original version of this manuscript was pointed out.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. [Online]. Available: http://mitpress.mit.edu/books/introduction-algorithms

[2] A. V. Goldberg and R. E. Tarjan, "Efficient maximum flow algorithms," *Communications of the ACM*, vol. 57, no. 8, pp. 82–89, 2014, available at https://cacm.acm.org/magazines/2014/8/177011-efficient-maximum-flow-algorithms.

[3] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *Journal of the ACM*, vol. 45, no. 5, pp. 783–797, 1998, announced at FOCS'97. [Online]. Available: http://doi.acm.org/10.1145/290179.290181

[4] J. E. Hopcroft and R. M. Karp, "A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[5] 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.

[6] S. Even and R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 507–518, 1975.

[7] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S. Teng, "Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs," in *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, June 6-8 2011*. ACM, 2011, pp. 273–282, available at https://arxiv.org/abs/1010.2921.

[8] J. Sherman, "Nearly maximum flows in nearly linear time," in *54th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, October 26-29, 2013*, 2013, pp. 263–269, available at https://arxiv.org/abs/1304.2077.

[9] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, "An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, OR, USA, January 5-7, 2014*, 2014, pp. 217–226, available at https://arxiv.org/abs/1304.2338.

[10] A. Schild, "An almost-linear time algorithm for uniform random spanning tree generation," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. ACM, 2018, pp. 214–227, available at https://arxiv.org/abs/1711.06455.

[11] A. Sidford and K. Tian, "Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, 2018, pp. 922–933, available at https://arxiv.org/abs/1808.01278.

[12] A. Madry, "Navigating central path with electrical flows: From flows to matchings, and back," in *54th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, October 26-29, 2013*. IEEE Computer Society, 2013, pp. 253–262, available at https://arxiv.org/abs/1307.2205.

[13] ——, "Computing maximum flow with augmenting electrical flows," in *57th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*. IEEE Computer Society, 2016, pp. 593–602, available at https://arxiv.org/abs/1608.06016.

[14] Y. T. Lee and A. Sidford, "Solving linear programs with sqrt(rank) linear system solves," *CoRR*, vol. abs/1910.08033, 2019. [Online]. Available: http://arxiv.org/abs/1910.08033

[15] Y. P. Liu and A. Sidford, "Faster energy maximization for faster maximum flow," in *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. ACM, 2020, pp. 803–814, available at https://arxiv.org/abs/1910.14276.

[16] T. Kathuria, Y. P. Liu, and A. Sidford, "Unit capacity maxflow in almost $O(m^{4/3})$ time," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020, pp. 119–130.

[17] 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," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, 2020, pp. 919–930.

[18] J. v. d. Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang, "Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances," *CoRR*, vol. abs/2101.05719, 2021, available at https://arxiv.org/abs/2101.05719.

[19] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 981–1025, 2011, available at https://arxiv.org/abs/0808.4134.

[20] Z. Galil and A. Naamad, "An $O(EV \log^2 V)$ algorithm for the maximal flow problem," *Journal of Computer and System Sciences*, vol. 21, no. 2, pp. 203–217, 1980.

[21] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *Journal of Computer and System Sciences*, vol. 26, no. 3, pp. 362–391, 1983, announced at STOC'81.

[22] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.

[23] P. M. Vaidya, "Speeding-up linear programming using fast matrix multiplication (extended abstract)," in *30th IEEE Annual Symposium on Foundations of Computer Science, FOCS 1989, Research Triangle Park, NC, USA, October 30 - November 1, 1989*. IEEE Computer Society, 1989, pp. 332–337.

[24] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks*. Mathematical Association of America, 1984, available at https://arxiv.org/abs/math/0001057.

[25] D. Durfee, Y. Gao, G. Goranci, and R. Peng, "Fully dynamic spectral vertex sparsifiers and applications," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. ACM, 2019, pp. 914–925, available at https://arxiv.org/abs/1906.10530.

[26] L. Chen, G. Goranci, M. Henzinger, R. Peng, and T. Saranurak, "Fast dynamic cuts, distances and effective resistances via vertex sparsifiers," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020, pp. 1135–1146, available at https://arxiv.org/abs/2005.02368.

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

[28] S. I. Daitch and D. A. Spielman, "Faster approximate lossy generalized flow via interior point algorithms," in *Proceedings of the 40th annual ACM Symposium on Theory of Computing, STOC 2008, Victoria, BC, Canada, May 17-20, 2008*. New York, NY, USA: ACM, 2008, pp. 451–460, available at http://arxiv.org/abs/0803.0988. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374441

[29] M. Fahrbach, G. L. Miller, R. Peng, S. Sawlani, J. Wang, and S. C. Xu, "Graph sketching against adaptive adversaries applied to the minimum degree algorithm," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. IEEE Computer Society, 2018, pp. 101–112, available at https://arxiv.org/abs/1804.04239.

[30] A. Karzanov, "Determining the maximal flow in a network by the method of preflows," *Doklady Mathematics*, vol. 15, pp. 434–437, 02 1974.

[31] J. B. Orlin, "Max flows in $O(nm)$ time, or better," in *Proceedings of the 45th Annual ACM Symposium on Theory of Computing, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*. ACM, 2013, pp. 765–774.

[32] V. King, S. Rao, and R. Tarjan, "A faster deterministic maximum flow algorithm," *Journal of Algorithms*, vol. 17, no. 3, pp. 447–474, 1994.

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

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

[35] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988.

[36] R. Peng, "Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. SIAM, 2016, pp. 1862–1867.

[37] J. Sherman, "Generalized preconditioning and undirected minimum-cost flow," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. SIAM, 2017, pp. 772–780, available at https://arxiv.org/abs/1606.07425.

[38] 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 2019, Phoenix, AZ, USA, June 23-26, 2019*. ACM, 2019, pp. 902–913, available at https://arxiv.org/abs/1906.10340.

[39] Y. T. Lee and A. Sidford, "Efficient inverse maintenance and faster algorithms for linear programming," in *56th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, October 17-20, 2015*. IEEE Computer Society, 2015, pp. 230–249, available at https://arxiv.org/abs/1503.01752.

[40] M. B. Cohen, Y. T. Lee, and Z. Song, "Solving linear programs in the current matrix multiplication time," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. ACM, 2019, pp. 938–942, available at https://arxiv.org/abs/1810.07896.

[41] J. v. d. Brand, "A deterministic linear program solver in current matrix multiplication time," in *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. SIAM, 2020, pp. 259–278, available at https://arxiv.org/abs/1910.11957.

[42] J. van den Brand, "Unifying matrix data structures: Simplifying and speeding up iterative algorithms," in *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*. SIAM, 2021, pp. 1–13, available at https://arxiv.org/abs/2010.13888.

[43] J. v. d. Brand, Y. T. Lee, A. Sidford, and Z. Song, "Solving tall dense linear programs in nearly linear time," in *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. ACM, 2020, pp. 775–788, available at https://arxiv.org/abs/2002.02304.

[44] H. Jiang, T. Kathuria, Y. T. Lee, S. Padmanabhan, and Z. Song, "A faster interior point method for semidefinite programming," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020, pp. 910–918, available at: https://arxiv.org/abs/2009.10217.

[45] Y. T. Lee and A. Sidford, Personal communication.

[46] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff, "Fast moment estimation in data streams in optimal space," in *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, June 6-8 2011*. ACM, 2011, pp. 745–754, available at https://arxiv.org/abs/1007.4191.

[47] K. Onak and R. Rubinfeld, "Maintaining a large matching and a small vertex cover," in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, MA, USA, June 5-8 2010*. ACM, 2010, pp. 457–464, available at http://people.csail.mit.edu/ronitt/papers/01-maintaining.pdf.

[48] M. Gupta and R. Peng, "Fully dynamic $(1+\epsilon)$-approximate matchings," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, October 26-29, 2013*, 2013, pp. 548–557, available at http://arxiv.org/abs/1304.0378.

[49] A. Bernstein and C. Stein, "Fully dynamic matching in bipartite graphs," in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 9134. Springer, 2015, pp. 167–179, available at https://arxiv.org/abs/1506.07076.

[50] S. Bhattacharya, M. Henzinger, and D. Nanongkai, "New deterministic approximation algorithms for fully dynamic matching," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. ACM, 2016, pp. 398–411, available at https://arxiv.org/abs/1604.05765.

[51] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng, "On fully dynamic graph sparsifiers," in *57th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*. IEEE Computer Society, 2016, pp. 335–344, available at https://arxiv.org/abs/1604.02094.

[52] M. Henzinger, S. Krinninger, and D. Nanongkai, "Decremental single-source shortest paths on undirected graphs in near-linear total update time," *Journal of the ACM*, vol. 65, no. 6, pp. 36:1–36:40, 2018, available at https://arxiv.org/abs/1512.08148.

[53] S. Forster and G. Goranci, "Dynamic low-stretch trees via dynamic low-diameter decompositions," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. ACM, 2019, pp. 377–388, available at https://arxiv.org/abs/1804.04928.

[54] G. Goranci, M. Henzinger, and P. Peng, "The power of vertex sparsifiers in dynamic graph algorithms," in *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, ser. LIPIcs, vol. 87. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 45:1–45:14, available at https://arxiv.org/abs/1712.06473.

[55] ——, "Dynamic effective resistances and approximate schur complement on separable graphs," in *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, ser. LIPIcs, vol. 112. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 40:1–40:15, available at https://arxiv.org/abs/1802.09111.

[56] R. Peng, B. Sandlund, and D. D. Sleator, "Optimal offline dynamic 2, 3-edge/vertex connectivity," in *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019*, ser. Lecture Notes in Computer Science, vol. 11646. Springer, 2019, pp. 553–565, available at https://arxiv.org/abs/1708.03812.

[57] G. Goranci, "Dynamic graph algorithms and graph sparsification: New techniques and connections," *CoRR*, vol. abs/1909.06413, 2019, available at https://arxiv.org/abs/1909.06413.

[58] W. Jin and X. Sun, "Fully dynamic c-edge connectivity in subpolynomial time," *CoRR*, vol. abs/2004.07650, 2020, available at https://arxiv.org/abs/2004.07650.

[59] 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, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 1122–1129, available at https://arxiv.org/abs/1611.03745.

[60] 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, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 1130–1143, available at https://arxiv.org/abs/1611.02864.

[61] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "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*. IEEE Computer Society, 2017, pp. 950–961, available at https://arxiv.org/abs/1708.03962.

[62] T. Saranurak and D. Wang, "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*. SIAM, 2019, pp. 2616–2635, available at https://arxiv.org/abs/1812.08958.

[63] 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 *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020, pp. 1158–1167, available at https://arxiv.org/abs/1910.08025.

[64] 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, available at https://arxiv.org/abs/2004.08432.

[65] E. Gat and S. Goldwasser, "Probabilistic search algorithms with unique answers and their cryptographic applications," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 18, p. 136, 2011, available at https://eccc.weizmann.ac.il/report/2011/136/.

[66] O. Goldreich, S. Goldwasser, and D. Ron, "On the possibilities and limitations of pseudodeterministic algorithms," in *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ser. ITCS '13, 2013, pp. 127–138.

[67] H. Lin, "Reducing directed max flow to undirected max flow," *Unpublished Manuscript*, vol. 4, no. 2, 2009.

[68] R. K. Ahuja and J. B. Orlin, "Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems," *Naval Research Logistics*, vol. 38, no. 3, pp. 413–430, 1991.

[69] M. B. Cohen, A. Madry, P. Sankowski, and A. Vladu, "Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log W)$ time (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. SIAM, 2017, pp. 752–771, available at https://arxiv.org/abs/1605.01717.

[70] D. A. Spielman and S. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004, Chicago, IL, USA, June 13-16, 2004*, 2004, pp. 81–90, available at https://arxiv.org/abs/0809.3232, https://arxiv.org/abs/0808.4134, https://arxiv.org/abs/cs/0607105.

[71] I. Koutis, G. L. Miller, and R. Peng, "Approaching optimality for solving SDD linear systems," in *51th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, NV, USA, October 23-26, 2010*, 2010, pp. 235–244, available at https://arxiv.org/abs/1003.2958.

[72] ——, "A nearly-m log n time solver for SDD linear systems," in *52nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, 2011, pp. 590–598, available at https://arxiv.org/abs/1102.4842.

[73] J. A. Kelner, L. Orecchia, A. Sidford, and Z. Allen Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *Proceedings of the 45th Annual ACM Symposium on Theory of Computing, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 911–920, available at https://arxiv.org/abs/1301.6628.

[74] 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, New York, NY, USA, June 1-3, 2014*, 2014, pp. 343–352.

[75] R. Kyng, Y. T. Lee, R. Peng, S. Sachdeva, and D. A. Spielman, "Sparsified Cholesky and multigrid solvers for connection Laplacians," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, 2016, pp. 842–850, available at https://arxiv.org/abs/1512.01892.

[76] R. Kyng and S. Sachdeva, "Approximate gaussian elimination for Laplacians - fast, sparse, and simple," in *57th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, NJ, USA, October 9-11, 2016*, 2016, pp. 573–582, available at https://arxiv.org/abs/1605.02353.

[77] D. Durfee, M. Fahrbach, Y. Gao, and T. Xiao, "Nearly tight bounds for sandpile transience on the grid," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. SIAM, 2018, pp. 605–624, available at https://arxiv.org/abs/1704.04830.