Faster Maxflow via Improved Dynamic Spectral Vertex Sparsifiers

Jan van den Brand
Simons Institute & UC Berkeley
Berkeley, USA
vdbrand@berkeley.edu

Yu Gao
Georgia Tech
Atlanta, USA
ygao380@cc.gatech.edu

Arun Jambulapati
Stanford University
Stanford, USA
jmlpati@stanford.edu

Yin Tat Lee
University of Washington
Seattle, USA
yintat@uw.edu

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

Richard Peng
Georgia Tech
Atlanta, USA
rpeng@cc.gatech.edu

Aaron Sidford
Stanford University
Stanford, USA
sidford@stanford.edu

ABSTRACT
We make several advances broadly related to the maintenance of electrical flows in weighted graphs undergoing dynamic resistance updates, including:

1. A more efficient dynamic spectral vertex sparsification, obtained by faster length estimation of random walks in weighted graphs using Morris counters [Morris 1978, Nelson-Yu 2020].

2. A direct reduction from detecting edges with large energy in dynamic electric flows to dynamic spectral vertex sparsifiers.

3. A procedure for turning algorithms for estimating a sequence of vectors under updates from an oblivious adversary to one that tolerates adaptive adversaries via the Gaussian-mechanism from differential privacy.

Combining these pieces with modifications to prior robust interior point frameworks gives an algorithm that on graphs with m edges computes a mincost flow with edge costs and capacities in (1, U) in time $O(m^{3/2-1/58} \log^2 U)$. In prior and independent work, [Axiotis-Madry-Vladau FOCS 2021] also obtained an improved algorithm for sparse mincost flows on capacitated graphs. Our algorithm implies a $O(m^{3/2-1/58} \log U)$ time maxflow algorithm, improving over the $O(m^{3/2-1/328} \log U)$ time maxflow algorithm of [Gao-Liu-Peng FOCS 2021].

CCS CONCEPTS
• Theory of computation → Convex optimization.

KEYWORDS
Interior Point Methods, Dynamic Effective Resistance, Maximum Flow

ACM Reference Format:
Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. 2022. Faster Maxflow via Improved Dynamic Spectral Vertex Sparsifiers. In Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC ’22), June 20–24, 2022, Rome, Italy. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3520068

1 INTRODUCTION
The maximum flow (maxflow) problem asks to route the maximum amount of flow between two vertices s and t in a directed graph G such that the amount of flow on every edge is at most its capacity. The more general minimum cost flow (mcf) problem asks to route a fixed demands in a directed graph G without sending more flow on any edge than its capacity, while minimizing a linear cost. Together these well-studied problems cover a wide range of combinatorial and numerical problems, including maximum cardinality bipartite matching, minimum s-t cut, shortest paths in graphs with negative edge length, and optimal transport (see e.g. [14, 15]).

While classical algorithms for these problems revolved around using augmenting paths or cycle primitives (such as blocking flows) [30, 33, 34, 42], the last decade has seen significant runtime improvements for maxflow and mincost flow in various settings based on electrical flows. For a graph G with vertex set V and edge set E, with edge resistances $r \in \mathbb{R}_{>0}^E$, and a demand vector $d \in \mathbb{R}^E$, the electric flow on G is the flow that routes a fixed demand while minimizing the energy $\sum_{e \in E} f_e r_e$. Better maxflow algorithms have been given in several regimes using electrical flows and stronger primitives [1, 50], including unit capacity graphs [3, 23, 43, 58–60], approximate maxflow on undirected graphs [17, 44, 52, 66, 70–72], and dense graphs [14, 15, 55].

However, it has been particularly challenging to obtain running time improvements for solving mincost flow and maxflow to high precision in sparse capacitated graphs. Recently, [31] gave an $O(m^{3/5-1/328} \log U)$ time algorithm for sparse graphs with capacities bounded by U, the first improvement over $O(m^{1.5} \log U)$ for

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

STOC ’22, June 20–24, 2022, Rome, Italy.
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9264-8/22/06...
https://doi.org/10.1145/3519935.3520068
Full version available at https://arxiv.org/pdf/2112.00722.pdf.
maxflow on sparse graphs with arbitrary, polynomially bounded capacities. Their improvement involved an intricate interaction of dynamic data structures for electrical flows, a modification of the standard interior point method (IPM) outer loop which builds an maxflow via $\sqrt{m}$ approximate electric flows [41, 74], and sketching techniques. Additionally, issues relating to randomness in data structures and combinatorial reasoning about errors resulting from random walks required careful analysis that significantly increase the runtime and resulted in the small improvement over $m^{1.5}$.

Our main result is an algorithm that while has a similar high-level picture as [31], significantly simplifies the major pieces described previously and their interactions. Specifically, we give a general purpose sketching tool for electrical flows, a graph theoretic (instead of algebraic) way of constructing the random walks at the core of the data structure, and handle randomness dependencies using ideas from differential privacy. Further, we provide modifications to prior robust interior point frameworks to do $\ell_2$-based recentering (Definition 7.3) within a robust IPM via additional spectral vertex sparsification techniques. As a result, we achieve a faster runtime than in [31]. Also, as a result of our simplified electric flow data structure, our algorithm and IPM seamlessly extend to mincost flow. In constrast, the data structure complications in [31] restricted their algorithm to be applied to maxflow. \footnote{[4], in FOCS 2021, claims an improvement to sparse capacitated mincost flow in the title. A preprint was recently made available at https://arxiv.org/abs/2111.10368v1. The results in this paper were derived independently: we defer detailed comparisons to a future version.}

**Theorem 1.1.** There is an algorithm which given any $m$-edge directed graph $G$ with integral capacities in $[1, U]$, feasible demand vector $d \in \mathbb{Z}^E$, and an integral cost vector $c \in [-U, U]^E$, computes a flow $f$ that routes demand $d$, satisfies the capacity constraints, and minimizes $c^\top f$. The algorithm succeeds whp. and runs in time $\tilde{O}(m^{1.2} \sqrt{\epsilon^{-1}} \log^2 \epsilon U)$.

Overall, this paper simplifies the key pieces of [31] and, as a result, clarifies the important components used to achieve faster maxflow algorithms via dynamic electric flows. Further, we consider each of these pieces to be interesting in their own right: dynamic maintenance of Schur complements, sketching and maintenance of high energy edges in dynamic electric flows, and understanding reductions between adaptive and oblivious adversaries. Ultimately, this paper be read independently of [31] and the proofs are simpler and more natural in many cases.

### 1.1 Key Algorithmic Pieces

Here we cover the key algorithmic pieces underlying our algorithm. The first major piece is a faster algorithm for generating random walks of a fixed length from a vertex, a core primitive in all random-walk based approaches to dynamic electric flows [27, 31]. Our second key contribution is an algorithm for detecting large energy edges in electric flows for graphs with dynamically changing resistances and demands based on a direct reduction to dynamic spectral vertex sparsifiers (Schur complements). As our data structures for maintaining dynamic electric flows naturally work only against oblivious adversaries, we develop an approach that black-box reduces dynamic electric flows against adaptive adversaries to the same problem against oblivious adversaries, at the cost of a small runtime increase.

**Faster generation of random walks and Schur complements.** All previous algorithms for dynamic electric flows [27, 31] require dynamic maintenance of spectral vertex sparsifiers or Schur complements, which approximate the electric flow and potentials onto a smaller set of terminal vertices. The Schur complement is generated by sampling several random walks from vertices $v$ with exit probabilities proportional to inverse resistances until the walk visited a fixed number $L$ of distinct vertices, and by estimating the sum of resistances of edges along the walk.

Because there may be edges with very large or small resistances, a naïve simulation may get stuck for polynomially many steps. Consequently, [27] gave an algorithm for this based on taking high powers of the random walk matrix (which [31] applied in a black-box fashion). This generated large factors in the runtime of the data structures. We give an approach to significantly speed up the sampling of vertices and length estimation by applying a Morris counter from the streaming/sketching literature [63], and reducing the problem to solving a sequence of electric flow computations (Laplacian systems) as opposed to the more expensive matrix multiplication operations of [27]. This significant runtime improvement immediately translates to our dynamic electric flow data structure described above, which directly uses dynamic Schur complements.

**Simplified electric flow heavy hitter.** To design our dynamic algorithm for detecting edges with large energy in dynamic electric flows, we maintain an $\ell_2$ heavy hitter sketch of the electric flow vector. The algorithm of [31] maintained this sketch by using a dynamic spectral vertex sparsifier or Schur complement, which approximates the electric flow and potentials on a smaller set of terminal vertices, and several random walks for “moving” the heavy hitter sketch vector to the terminal set. This latter piece (maintaining random walks for moving the heavy hitter vector) introduced several complications into the analysis and generated a large overall running time for the data structure. On the other hand, our algorithm is more directly based on spectral approximations. In particular, we show how to dynamically maintain the result of moving the heavy hitter vector onto the terminal set by simply calling another dynamic Schur complement data structure, and carefully reasoning about spectral approximations to bound how that affects the resulting error.

**Simplified IPM outer loop.** As a result of our more linear algebraic approach to maintaining electric flows, our data structure for detecting large energy edges in electric flows works for both dynamic resistance changes and dynamic demands, while the algorithm of [31] required restricting to only $s$-$t$ flows. Our generalization also allows us to use a more standard and efficient robust IPM (from [26]) to implement the outer loop utilizing the data structure, while [31] had to redesign the IPM to carefully only use $s$-$t$ electric flows to interact with their data structure. Our robust IPM implements an additional batching, or $\ell_2$-based recentering step, by computing the changes on a small subset of edges to higher accuracy by using spectral vertex sparsifiers.
Black-box reduction from adaptive to oblivious adversaries. As we are applying randomized data structure inside an algorithmic outer loop, their previous responses may affect future updates. This is referred to an adaptive adversary in the literature. On the other hand, our data structures which are based on random walks naturally only work against oblivious adversaries, where the input sequence does not depend on the outputs and randomness of the data structure. The algorithm of [31] handled this issue in their data structures by carefully controlling the total number of adaptive phases of their algorithm before snapping back to a deterministic state.

Our approach on the other hand is more black-box, and gives a more general approach for converting data structures against oblivious adversaries to handle adaptive queries. We build a LOCATOR which returns a superset of edges with large energies, and several EVALUATORS with differing accuracy parameters which separately estimate the energies of the edges. By leveraging ideas from the Gaussian-mechanism from differential privacy [29] we show how to apply the EVALUATOR data structures to simulate estimating adding Gaussian noise to the true energy vector that we wish to output. We simulate this by making several queries to the EVALUATORS, where we query the least accurate EVALUATORS most often, and only query more accurate EVALUATORS when the estimate of the energy vector is close to certain thresholds and we require finer estimates to decide how to round. Because we are simulating adding noise to the true output, the algorithm succeeds against an adaptive adversary.

1.2 Related Work
We briefly survey the lines of work most relevant to our results, and refer the reader to [31] for more comprehensive discussion. Recently, [25] gave a mincost flow algorithm on planar graphs running in nearly linear time. Similar to our paper, it is based on the robust IPM framework of [26] and dynamic Schur complements. However, [25] relies on the fact that the terminal set C is small due to the existence of planar separators, while our paper relies on the fact that C is slowly changing.

Data structures for IPMs. IPMs are a powerful framework which reduces linear programming with m variables to a sequence of $O(\sqrt{m})$ linear system solutions [67]. For maxflow and mincost flow, these linear systems correspond to computing electrical flows, and Daitch-Spielman [24] leveraged this observation to give a $O(m^{1.5} \log U)$ mincost flow algorithm. Recently, several works have leveraged the key fact dating back to early works of Karmarkar [41] and Vaidya [74] that the linear systems change slowly and only need to be solved approximately, both in the context of linear programs [12–14, 16, 22, 39, 54, 56] and mincost flows [14, 15, 31].

Dynamic electrical flows. Recent works applying dynamic data structures to IPMs for maxflow require maintaining various properties of electrical flows on dynamically changing graphs. The improvements on dense graphs [14, 15] required dynamically maintaining spectral sparsifiers of the Laplacian in $O(1)$ time per edge update and $O(n)$ per query, as well as detecting edges with large electrical energies in $O(n)$ time per query. Both of these pieces were done using dynamic expander decompositions [7, 19, 63, 64, 68, 75].

The work of [31] desired sublinear time per query and hence required dynamically maintaining Schur complements, whose study was initiated in [27] to dynamically maintain approximate effective resistances.

Adaptivity and differential privacy. There has been significant work towards building techniques to apply oblivious data structures in the context of an algorithmic outer loop, which requires adaptivity. To date, most approaches to this problem involve either making the algorithm deterministic [8–10, 18, 20, 36], or resparsiifying [7], both of which heavily leverage properties provided by dynamic expander decompositions [7, 19, 63, 64, 68, 75]. Our work takes a different perspective and instead more carefully analyzes whether the adversary can learn any randomness leaked from the distribution of our output vector. This perspective is motivated by ideas from differential privacy, and in fact our key result is an adaptation of the Gaussian mechanism [29] which simulates adding unbiased Gaussian noise to the true output vector by using a sequence of oblivious estimates. Our recursive scheme is also broadly related to the idea of multilevel Monte Carlo [11, 32] and its recent applications in leveraging approximate optimization procedures to obtain nearly unbiased estimates of minimizers [2]. Differential privacy has also previously been used in streaming algorithms to make them robust against adaptive adversaries [6, 35, 37]. Recent independent work has transferred some of these adaptive streaming techniques to dynamic algorithms [5].

1.3 General Notation
We use plaintext to denote scalars, bold lowercase for vectors, and bold uppercase for matrices. For resistances r and conductances w $\equiv r^{-1}$, the corresponding capital matrices are diagonal matrices with the vector entries on the diagonal, i.e. $R \equiv \text{diag}(r)$ and $W \equiv \text{diag}(w)$. As our algorithm heavily use approximations, we will use $\approx$ to denote the approximate versions of true variables.

We use $O(\cdot)$ to suppress logarithmic factors in $m$ and $\Omega(\cdot)$ to suppress inverse logarithmic factors in $m$. For vectors $x, y$, we sometimes let $xy$ denote the entry-wise product of $x, y$, so $(xy)_i \equiv x_i y_i$. Similarly, we let $(x/y)_i \equiv x_i/y_i$. We say that an event holds with high probability (whp.) if for any constant $C > 0$, the event succeeds with probability at least $1 - n^{-C}$ by adjusting parameters. We let $[n] = \{1, 2, \ldots, n\}$. We denote the (unweighted) degree of a vertex $v$ as $\text{deg}(v)$.

We say that a symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite (PSD) if $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$. For PSD matrices $A, B$ we write $A \preceq B$ if $B - A$ 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 $A, B$ we write $A \approx_{\gamma} B$ if $\exp(-\gamma)B \preceq A \preceq \exp(\gamma)B$.

1.4 Organization
In Section 2, we give a technical overview of each of our improvements to each of the key components of [31]: faster sampling of Schur complements, operator-based electric flow heavy hitters, and black-box reduction of adaptive to oblivious adversaries. We also overview the robust IPM we use. In Section 3 we give preliminaries for maxflow, mincost flow, and electric flows that we require for the remainder of our paper. In Section 4 we give our algorithm for
faster sampling of random walks and Schur complements, and we combine this with an operator-based heavy hitter in Section 5 to give a faster algorithm for detecting edges with large energy. In Section 6 we show how to black-box reduce adaptive to oblivious adversaries for the problem of estimating dynamic vectors. We give our robust IPM in Section 7, which is an adaptation of that in [26], and additional tools to apply it. Finally, we combine all pieces and compute the final runtime in Section 8.

2 OVERVIEW

Here we provide a technical overview of our contributions.

2.1 Overview of Faster Schur Complements via the Morris walk

Our data structures, as in [31], heavily rely on dynamically maintaining spectral vertex sparsifiers (Schur complements) of \(G\), which approximate the inverse spectral form of \(G\) onto a subset of the vertices. This was achieved using the algorithm of [27], which showed how to dynamically maintain an approximate Schur complement under edge resistance updates. The main primitive behind the dynamic Schur complement data structure was a procedure to sample random walks from a vertex with exit probability proportional to inverse resistances, i.e. the probability of going from a vertex \(v\) to a neighbor \(u\) is given by

\[
\frac{r_{vu}^{-1}}{\sum_{w \text{ neighbor of } v} r_{uw}^{-1}}.
\]

For this walk and a parameter \(L\), we must run the walk until the total degree of visited vertices is \(L\), and to estimate the total resistive length of the walk up to a \((1+\epsilon)\)-factor, where resistive length refers to the sum of resistances of edges on the walk. Directly simulating the random walk is not efficient enough, because there may be polynomially large and small resistances, which cause the walks to "get stuck" on a small set of edges, without visiting new vertices. Thus it can take a long time to visit \(L\) distinct vertices. Despite this, [27] showed how to sample the walk and resistive length in \(O(L^3\epsilon^{-2})\) time per vertex, and this large runtime directly led to the fact that [31] only achieved a small 1/328 improvement in the exponent.

Interestingly, if one is only interested in obtaining distinct vertices on the walk (and not the resistive length) until the total degree is \(L\), this can be done in \(\tilde{O}(L^2)\) time by solving a Laplacian system (corresponding to computing an electric flow) for each of at most \(L\) steps to compute the next exit vertex. However, the approach of [27] which also computed the resistive length, i.e. the sum of resistances of edges on the walk, was based on matrix-powering/matrix multiplication, and instead had a larger \(\tilde{O}(L^4\epsilon^{-2})\) runtime. Our main idea is to resolve this runtime discrepancy between sampling the distinct vertices and computing a \((1+\epsilon)\)-resistive length estimate by giving an algorithm that computes both quantities by solving a sequence of Laplacian systems. In total, we solve \(O(L + \epsilon^{-2})\) systems, for a runtime of \(\tilde{O}(L^2 + \epsilon^{-2})\). In our settings, \(L\) will generally be \(O(\epsilon^{-2})\), so our runtime is \(\tilde{O}(L^2)\), matching the time to generate the first \(L\) vertices using a sequence of Laplacian systems, and significantly improving over the \(\tilde{O}(L^4\epsilon^{-2})\) runtime of [27, 31].

Our algorithm for this task is derived from the Morris counter [62, 65], a probabilistic algorithm for maintaining low-space approximations to a counter \(N\) undergoing increments. For simplicity of exposition, we assume our input graph has integer, polynomially-bounded edge weights. Our algorithm intuitively begins by running a random walk in \(G\). However, we replace the naive procedure for computing the resistive length with a Morris counter. More precisely, assume we have run a random walk starting from a vertex \(u\) for \(k\) steps and have estimated the resistive length of the walk via the Morris counter. To estimate the resistive length of this walk after a further step, we simply sample one new step of the walk: if we sampled an edge of resistance length \(w\), we increment the Morris counter \(w\) times. In this way, the Morris counter enables us to maintain estimates of the resistive length of a random walk.

We use the following properties of Morris counters as shown in [65]. First, they take discrete values of \(\frac{1}{\epsilon} \left( (1 + a)^i - 1 \right)\) for some real number \(a > 0\) and integers \(i \geq 0\), and if \(a = \epsilon^2 / \text{poly log } n\), the value of the Morris counter is always a \((1+\epsilon)\)-approximation of the true value with high probability. In particular, for graphs with polynomially bounded weights, the Morris counters takes at most \(\tilde{O}(n \epsilon^{-2})\) distinct values.

Our key insight we can simulate incrementing of the Morris counter and the sequence of vertices visited as a random walk on a "lifted" graph with \(\tilde{O}(\epsilon^{-2})\) layers. Each time we explore a "new" neighbor in this lifted space, we either find a new unexplored vertex along the random walk or increment the Morris counter. However, there are only \(\tilde{O}(\epsilon^{-2})\) distinct values of the counter: thus we must explore only an additional \(\tilde{O}(\epsilon^{-2})\) distinct vertices in this lifted space to obtain the desired guarantee on the number of new vertices seen. We obtain our final algorithm by replacing the explicit random walks with a subroutine based on Laplacian linear system solvers: in this way our final complexity of \(\tilde{O}(L^2 + \epsilon^{-2})\) follows. Overall, this provides a graph-theoretic approach for estimating lengths of random walks on graphs, as opposed to the previous algorithm in [27] which was based on matrix multiplication.

2.2 Overview of Operator-based Electric Flow Heavy Hitters

Our next major improvement over [31] is a data structure that detects large energy edges in electric flows on graphs with dynamic resistances and demands by direct reduction to maintaining dynamic Schur complements. To be precise, we give a data structure that on a graph \(G\) with dynamically changing resistances and demands, solves a electric flow heavy hitter problem, by returning a set \(S\) of \(\tilde{O}(\epsilon^{-2})\) edges containing all edges \(e\) with at least \(\epsilon^2 f_e\) fraction of the electric energy, i.e. \(r_e f_e \geq \epsilon^2 \sum_{e \in S} r_e f_e\) where \(f\) is the electric flow vector. [31] gave a data structure that solved this problem in sublinear time per resistance update and query as a core piece of their algorithm. We give improved runtimes for solving this problem and shed light on its complexity by directly reducing to dynamically maintaining Schur complements.

Note that the dynamic electric flow heavy hitter problem is equivalent to detecting large coordinates of the vector \(R^{1/2} f\) compared to its \(\ell_2\) norm. Hence, it is natural to apply an \(\ell_2\) heavy-hitter sketch [48], which at a high-level consists of \(\tilde{O}(\epsilon^{-2})\) Johnson-Lindenstrauss \(\ell_2\) sketches. In total, this consists of maintaining the
The dynamic electric flow data structures built in Section 2.2 naïvely value of \( q^T R^{-1/2} f \) for \( \Omega(\epsilon^{-2}) \) random sketch vectors \( q \in \{-1, 0, 1\}^F \). The flow \( f \) can be represented as \( f = R^{-1} B \phi \) for electric potentials \( \phi \) and edge-vertex incidence matrix \( B \), so

\[
q^T R^{-1/2} f = (B^T R^{-1/2} q)^T \phi.
\]

Let \( y = B^T R^{-1/2} q \), so that we focus on maintaining \( y^T \phi \). However, \( \phi \) is still a \( |V| \)-dimensional vector, so in order to achieve sublinear time [31] used a smaller terminal set \( C \) to estimate \( y^T \phi \). In particular, they write \( \phi = H_C \phi_C \) where \( \phi_C \) is the restriction of \( \phi \) to \( C \), and \( H_C \in \mathbb{R}^{V(G) \times C} \) is the harmonic extension (Definition 5.1) operator which extends \( \phi_C \) to \( \phi \) using that for any vertex \( v \), \( \phi_v \) is the average of its neighbors, weighted proportional to inverse resistances. This way, \( y^T \phi = (H_C^0 y, \phi_C) \). Assuming that we can approximate maintain \( \phi_C \) (which we discuss towards of the end of this section’s overview), it suffices to maintain \( H_C^0 y \).

Our major difference from [31] is in how we maintain \( H_C^0 y \). While [31] used the combinatorial interpretation of the operator \( H_C^0 \) as using random walks to “move” the mass from vector \( y \) onto \( C \), we use the spectral fact that

\[
H_C = L^1 \left[ 0 \right. \begin{array}{c} \text{SC}(L, C) \end{array} \left. \right].
\]

Thus, we get

\[
H_C^0 y = \left[ \begin{array}{c} \text{SC}(L, C) \end{array} \right] L^1 y.
\]

Thus, we could optimistically precompute \( L^1 y \) and then compute \( H_C^0 y \) as long as we can dynamically maintain the Schur complement \( \text{SC}(L, C) \), which is a size \( C \) object. The remaining issue is that the Laplacian \( L \) may change because the resistances change. However, the operator \( H_C^0 \) does not depend on the resistances of edges completely inside \( C \) (by definition), so we may actually let \( L = \text{Laplacian of the original graph} \) as long as all endpoints of edges with resistance changes are added to \( C \). Finally we are able to show that using an approximate Schur complement in place of \( \text{SC}(L, C) \) still suffices for our data structures (Theorem 4.3).

Finally we discuss the (approximate) maintenance of the potential \( \phi_C \). For an electric flow \( f \) routing demand \( d \), i.e. \( B^T f = d \), the potentials \( \phi_C \) are given by

\[
\phi_C = \text{SC}(L, C)^{-1} H_C^0 d.
\]

In other words, we first “move” the demands to the terminal set using \( H_C^0 \) just as above, and then invert the Schur complement on it. Therefore we can maintain \( \phi_C \) as follows: maintain \( H_C^0 d \) approximately as above, and then also approximate maintain \( \text{SC}(L, C) \) using an approximate Schur complement data structure. In all, this reduces the maintenance of the heavy hitter vector to three calls to an approximate Schur complement oracle.

### 2.3 Overview of Reduction from Adaptive to Oblivious Adversaries

The dynamic electric flow data structures built in Section 2.2 naïvely only work against adaptive adversaries, i.e. the inputs must be independent of the outputs and randomness of the data structure. In [31] this was handled by carefully designing the data structures to utilize the fact that the IPM central path is a deterministic object. However, we take a more general approach, by applying ideas from

---

Figure 1: Density function \( d \) of \( N(\nu, \sigma^2) \), and density function \( \tilde{d} \) of \( N(\tilde{\nu}, \tilde{\sigma}^2) \) scaled by some \( \exp(-x) \), \( \alpha > 0 \) so that \( \tilde{d}(x) \exp(-x) \leq d(x) \).

---

This is actually only true for \( x \in D \) for some event \( D \) that holds whp. We ignore this here for simplicity.
When case (i) happens, we handle it directly by computing \( v \) exactly (which is expensive), which gives us the distributions \( d \) and \( \bar{d} \) explicitly. However, note that if \( \alpha \) is close to 0, then this case only occurs infrequently: with probability \( 1 - \exp(-\alpha) = O(\alpha) \), which balances out the expensive cost of computing \( v \). On the other hand, case (ii), which occurs with probability \( \exp(-\alpha) \), corresponds to flipping an unbalanced coin and with probability \( \exp(-\alpha) \) we sample a \( z' \sim N(\bar{B}, \sigma^2) \). So with probability \( \exp(-\alpha) \) we do not need to know/compute the exact vector \( v \) in order to obtain a sample with distribution \( N(\alpha, \sigma^2) \) and just knowing the approximate result \( \bar{v} \) already suffices.

Now, this scheme can be extended recursively to handle case (ii), i.e. sampling from \( z \sim N(B, \sigma^2) \). We can use the same scheme again via some \( \bar{B}' \), i.e. sampling from \( N(\bar{B}', \sigma^2) \) with probability \( \exp(-\alpha) \) instead of \( N(B, \sigma^2) \). This leads to another speed-up because of the approximation quality of \( \bar{B}' \) compared to \( B \). We want to use a large \( \alpha \) in order to reduce the probability of computing \( v \), but this requires \( \bar{B}' \) to be a better approximation. Thus, we are able to compute higher accuracy approximations (which take more runtime) less frequently, and this leads to a speedup. Overall, by using this scheme, our data structures will work against an adaptive adversary because of the following reason: the probability \( \exp(-\alpha) \) depends on which is close to \( \alpha \).

2.4 Overview of IPM Outer Loop

Here we overview how we apply the above primitives in a robust IPM to give an algorithm for algorithm, which reduces solving maxflow to computing a sequence of \( \bar{O}(\sqrt{m}) \) approximate electric flows. The IPM of [31] required several nonstandard modifications, including restricting to using \( s-t \) flows, which resulted in using more than \( \bar{O}(\sqrt{m}) \) steps, and overall higher runtime. On the other hand, our algorithm is based on the more standard robust IPM of [26], with an additional procedure that allows for recentering in the context of a robust IPM that allows us to control errors that accumulate over longer periods of time.

We start by briefly introducing a standard robust IPM setup for the mincost flow problem based on [26] (in Section 7 we change notation slightly to work with general linear programs)

\[
\min_{f \in \mathbb{R}^m, \mathbf{B}^T f = d} c^T f,
\]

where \( c \in \mathbb{R}^E \) is the cost vector, and \( \mathbf{t}, \mathbf{u} \in \mathbb{R}^E \) are lower/upper capacities on edges. For \( e \in E \) and real number \( f \in \mathbb{R} \), define the logarithmic barrier function \( \phi_e(f) \) of \( f \) as \( -\log(f-e) - \log(u_e-f) \), and for flow \( f \in \mathbb{R}^E \) define \( \phi(f) \) as \( \sum_{e \in E} \phi_e(f_e) \). For a path parameter \( \mu \) that decreases towards 0 over the course of \( \bar{O}(\sqrt{m}) \) steps, the robust IPM maintains an approximate minimizer to the expression

\[
f_{\mu} \overset{\text{def}}{=} \min_{f \in \mathbb{R}^m, \mathbf{B}^T f = d, \mathbf{t} \leq f \leq \mathbf{u}} c^T f + \mu \phi(f).
\]

Since \( \phi \) is convex, the KKT conditions for (2) give that there is a vector \( y \) such that \( c + \mu \nabla \phi(f_{\mu}) = \mu B y \). Thus there is a vector \( s_{y} \in \mathbb{R}^E \) such that \( B y + s_{y} = c/\mu \) and \( s_{y} + \nabla \phi(f_{\mu}) = 0 \). In this way, we define a \( \mu \)-centered point as a pair \((f,s)\) such that \( B y + s = c/\mu \) for some \( y \in \mathbb{R}^E \) and \( \|\nabla^2 \phi(f)^{-1/2} (s + \nabla \phi(f))\|_{\infty} \leq 1/64 \).

The robust IPM maintains \( \mu \)-centered points throughout by tracking the potential function

\[
\sum_{e \in E} \cosh(\lambda \phi_e(f_e)^{-1/2} (s_e + \phi_e((f_e)))
\]

for \( \lambda = 128 \log(16m) \). Now IPM steps are taken to simulate gradient descent steps on the potential to keep it small, and hence maintain \( \mu \)-centered points at all times.

Because our data structures work in time sublinear in the number of vertices, the flows we maintain during the robust IPM are stored implicitly, even without the ability to query in \( \bar{O}(1) \) time the “true flow” on an edge \( e \). Further, the error of our flow estimate from the true value accumulates over the steps of our method. Hence we require a procedure to recompute a feasible \( \mu \)-centered flow in a robust IPM every \( k \) steps in \( \bar{O}(m) \) time for some \( k = m^{O(1)} \) (Theorem 7.4). To see why this could be possible, note that the true step per iteration is an electric flow with some resistances and demands. Additionally, over the course of \( k \) steps, these resistances and demands will only change at most \( \text{poly}(k) \) total times. Thus, we can put all edges whose resistance or demand changed into a terminal set \( \mathcal{C} \) and compute an \( \epsilon \)-approximate Schur complement onto \( G \). [57] shows that such a Schur complement (onto a slightly larger set) can be computed in time \( \bar{O}(m + |E|/\epsilon^2) = \bar{O}(m + \text{poly}(k)/\epsilon^2) = \bar{O}(m) \) for some \( k = m^{O(1)} \). Leveraging this, we show that we can recover a centered point in the context of a robust IPM in \( \bar{O}(m) \) time every \( k \) steps.

Overall, our algorithm splits the \( \bar{O}(\sqrt{m}) \) robust IPM steps in \( \bar{O}(\sqrt{m}k) \) batches of \( k \) steps. Within each batch, we ensure that at most \( \text{poly}(k) \) edges have their resistances change in the graph \( G \) (but there may be more resistance updates in between batches). Each step in the batch is maintained using the dynamic electric flow heavy hitter data structure we built, as described in Sections 2.1 to 2.3. At the end of each batch, we use the approximate recentering procedure described in the previous paragraph. Combining these pieces along with the standard bound that over \( T \) IPM steps, at most \( \bar{O}(T^2) \) resistances change by a constant factor, gives our final runtimes.

3 PRELIMINARIES

We give preliminaries on maxflow, mincost flow, electric flows, and Schur complements.

\textit{Maxflow and mincost flow.} Throughout, we let \( G = (V,E) \) be our graph with \( n = |V| \) vertices and \( m = |E| \) edges. We let \( \mathbf{B} \in \mathbb{R}^{E \times V} \) denote the edge-vertex incidence matrix of \( G \). Additionally, we let \( \mathbf{t}, \mathbf{u} \in \mathbb{Z}^E \) denote the lower/upper capacities on edges in \( G \). We assume that \( \|\mathbf{t}\|_{\infty}, \|\mathbf{u}\|_{\infty} \leq U \). A flow \( f \in \mathbb{R}^E \) is any assignment of real numbers of edges of \( G \). We say that a flow \( f \) is feasible if \( \mathbf{t} \leq f \leq \mathbf{u} \) for all \( e \in E \). We say that \( f \) routes the demand \( \mathbf{d} \in \mathbb{R}^V \) if \( \mathbf{B}^T f = \mathbf{d} \).

The maximum flow problem asks to find a feasible flow routing the maximum multiple of a demand \( \mathbf{d} \) (generally assumed to be \( s-t \)). Written linear algebraically, this asks to find the largest \( F^* \) such that there is a flow \( f \) satisfying \( B^T f = F^* \) and \( t_e \leq f_e \leq u_e \) for all \( e \in E \). The minimum cost flow problem asks to minimize a linear cost \( c \) over flows routing a fixed demand \( \mathbf{d} \). Linear algebraically,
this can be written as
\[
\min_{B^T f = d} c^T f.
\]

We work with mincost flow throughout, as it is known to generalize maxflow. We also focus on finding high-accuracy solutions in runtime depending logarithmically on \(U\) and \(|e|\)\(_\infty\), as it is known that this suffices to get an exact solution with linear time overhead [14, 24].

**Electric flows and Schur complements.** Electric flows are \(\ell_2\) minimization analogues of maxflow on undirected graphs, and are used in all current state-of-the-art high accuracy maxflow algorithms [3, 14, 15, 31, 43] based on IPMs. On a graph \(G\) with resistances \(r\), the electric flow routing demand \(d\) is given by
\[
\arg \min_{B^T f = d} \sum_{e \in E} r_e f_e^2.
\]  
(3)

The minimizer in (3) is given by the solution to a linear system: \(f = R^{-1}B(B^T R^{-1}B)^T d\). The matrix \(B^T R^{-1}B\) is known as the Laplacian of \(G\), which can be solved in nearly-linear time [21, 38, 46–49, 51, 53, 73]. Precisely, solving a Laplacian system gives high accuracy vertex potentials, defined as \(\phi = (B^T R^{-1}B)^T d\).

**Theorem 3.1.** Let \(G\) be a graph with \(n\) vertices and \(m\) edges. Let \(r \in \mathbb{R}_+^E\) denote edge resistances. For any demand vector \(d\) and \(d > 0\) there is an algorithm which computes in \(\tilde{O}(m \log \epsilon^{-1})\) time potentials \(\phi\) such that \(\|\phi - \phi^\ast\|_L \leq \epsilon \|\phi^\ast\|_L\), where \(L = B^T R^{-1}B\) is the Laplacian of \(G\), and \(\phi^\ast = L^\dagger d\) are the true potentials determined by the resistances \(r\).

For notational convenience, we define the conductances \(w \overset{\text{def}}{=} r^{-1}\), and let \(L(w) \overset{\text{def}}{=} B^T WB\).

Several of our algorithms want to solve Laplacian systems in sublinear time. This can be done in the following natural sense: instead of returning the full potential vector \(\phi\), we only wish to determine \(\phi\) restricted to a subset of vertices \(C \subseteq V\). This is captured by a Schur complement, which is defined as \(\text{SC}(L, C) \overset{\text{def}}{=} L_{CC} - L_{CF}L_{FC}^{-1}L_{CC}\), where \(F = V \setminus C\) and \(L_{FF}, L_{FC}, L_{FC}, L_{CC}\) are blocks of the Laplacian \(L\) corresponding to rows/columns in \(F, C\). Schur complements satisfy two key properties which are essential for our algorithm: they are also graph Laplacians, and they are directly related to \(L^\dagger\) via the Cholesky factorization.

**Lemma 3.2 (Cholesky factorization).** For a connected graph \(G\) with Laplacian \(L \in \mathbb{R}^{V \times V}\), subset \(C \subseteq V\), and \(F \overset{\text{def}}{=} V \setminus C\),
\[
L^\dagger \begin{bmatrix}
0 & \frac{-L_{FF}L_{FC}}{I} \\
0 & \frac{I}{0}
\end{bmatrix}
\begin{bmatrix}
L_{FF} & 0 \\
0 & \text{SC}(L, C)^{-1}
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
-L_{CF}L_{FF}^{-1} & 1
\end{bmatrix}.
\]

The matrix appearing in the Cholesky factorization corresponds to mapping the potentials on \(C\) back to the whole graph via a harmonic extension. In other words, a random walk on \(G\), with exit probabilities proportional to conductances is a martingale on potentials. We give a more formal definition and properties later in Section 5.

Finally, it is very useful intuition that electric flows are inherently connected with the following random walk on \(G\): a vertex \(v\) goes to a neighbor \(u\) with probability proportional to conductance (inverse resistances), i.e. \(\sum_{w \in N(v)} w_{uw}\), where \(N(v)\) are the neighbors of \(v\) in \(G\). This random walk is the one used to define the harmonic extension, and also is used more directly in our algorithm for sampling Schur complements (see Theorem 4.3). Throughout, any mention of random walks refers to this random walk.

### 4 Improved Dynamic Schur Complements

In this section, we give our main algorithm for maintaining a Schur complement in a dynamic graph. Our main result is the following (see Theorem 4.3 for a more precise statement):

**Theorem 4.1 (Dynamic Schur Complement (informal)).** There is a data structure that supports the following operations against oblivious adversaries given a graph \(G = (V, E)\) with dynamic edge conductances \(w \in \mathbb{R}^{E(G)}\) and parameters \(\beta \leq e^{-\epsilon} < 1\):

- **Initialize** \((G, w, e, \beta)\). Initializes the data structure with accuracy parameter \(e\), and chooses a set of \(O(\beta m)\) terminals \(C\). \(\tilde{w}\) is initialized as \(w\). Runtime: \(\tilde{O}(m \beta^{-2} e^{-2})\).
- **AddTerminal** \((v)\). Makes \(v\) a terminal, i.e. \(C \leftarrow C \cup \{v\}\). Runtime: amortized \(\tilde{O}(1)\).
- **SC**(). Returns a Laplacian \(SC\) with \(O(\beta m e^{-2})\) edges which \((1+\epsilon)\)-spectrally approximates the Schur complement of \(L\) with terminal set \(C\) in time \(O(\beta m e^{-2})\).

All outputs and runtimes are correct with high probability if \(|C| = O(\beta m)\) at all times and there are at most \(O(\beta m)\) total calls to **UPDATE**.

Our proof is organized in two parts. In Section 4.1 we give an algorithm to efficiently generate useful attributes of a random walk in graphs with polynomially bounded edge weights. Next, in Section 4.2 we describe how to use these walk attributes to maintain a Schur complement under modifications to the terminal set and edge weights.

#### 4.1 Approximate Random Walks with Morris Counters

Our main contribution in this section is an improved algorithm to sample random walks in weighted graphs, based on the Morris counter of [62, 65]. The main technical result of this section is the following:

**Theorem 4.2 (Morris Walk).** Let \(G = (V, E, w, t)\) be a graph with edge weights \(w\) and edge lengths \(t\) bounded between 1 and \(n^{O(1)}\). For any vertex \(u\), and parameters \(L, \epsilon \geq 0\), there is an algorithm that with high probability runs in \(O(L^2 + L \epsilon^{-2})\) time and generates the following attributes of a random walk \(W_u\) in \(G\) which starts from \(u\), samples the edges it traverses with probabilities proportional to \(w\), and stops when \(\sum_{v \in V} \deg(v) > L^2\):

- \(u_1, u_2, \ldots\), the \(O(L)\) distinct vertices of \(W_u\) in order of their encounter.

\(^{(1)}\)Here, \(\deg\) denotes the unweighted degree of a vertex in \(G\).
• For each $u_i$, $\delta_{u_i}$ is a $(1 + \epsilon)$-approximation of

$$f_i \left( \sum_{k=1}^{f_{i+1}} \ell_{f_i(u_k,u_{k+1})} \right),$$

where $f_i$ is the index of the first visit of $u_i$ in $W_u$.

Our algorithm is based on simulating random walks in a graph by repeatedly solving linear systems, a technique that has been used in prior work on sampling random spanning trees and dynamically maintaining Schur complements [27, 28, 31, 45, 61, 69]. However, a difficulty in applying this approach to our setting is the need to estimate the length of the resulting random walk. We address this issue by appealing to an approximate counter algorithm to estimate the length of prefixes of the walk by simulating random walks on a larger graph.

The proof of Theorem 4.2 is deferred to the full version.

### 4.2 Improved Dynamic Schur Complement

Here we provide our main result regarding the dynamic maintenance of Schur complements under edge resistance changes in $G$. We achieve this by plugging in our improved algorithm Theorem 4.2 for estimating lengths of random walks visiting a fixed number of vertices into previous frameworks [27, 31]. Below, the additional operation $\text{InitializeSC}$ maintains the approximate Schur complement ignoring edge updates, but still tracking terminal additions. It is useful for our dynamic $\text{Evaluator}$ and $\text{Locator}$ data structures in Section 5. We use the notation $\text{SC}_H$ for the approximation as it eventually gets used to approximately compute a harmonic extension $H$.

Theorem 4.3 (Dynamic Schur complement). There is a data structure $\text{DynamicSC}$ that supports the following operations against oblivious adversaries given a graph $G = (V, E)$ with dynamic edge conductances $w \in \mathbb{R}^E$ and parameters $\beta < \epsilon^2 < 1$.

- $\text{Initialize}(G, w, e, \beta)$. Initializes the data structure with accuracy parameter $\epsilon$, and chooses a set of $O(1/m)$ terminals $C, \overline{C}$ initialized as $w$. Runtime: $O(m^{2/3} - \epsilon^2)$.
- $\text{AddTerminal}(v)$. Makes $v$ a terminal, i.e. $C \leftarrow C \cup \{v\}$. Runtime: amortized $O(\beta^{-2} \epsilon^2)$.
- $\text{Update}(e, w_{\text{new}})$. Under the guarantee that both endpoints of $e$ are terminals in $C$, updates $w_e \leftarrow w_{\text{new}}$. Runtime: amortized $O(1)$.
- $\text{SC}()$. Returns a Laplacian $\text{SC}(L(\overline{C}), C)$ with $O(\beta^{-m} \epsilon^2)$ edges in time $O(\beta^{-m} \epsilon^2)$.
- $\text{InitializeSC}()$. Returns a Laplacian $\text{SC}_H$ with $O(\beta^{-m} \epsilon^2)$ edges in time $O(\beta^{-m} \epsilon^2)$. Let $Z$ be the set of edges which were input to $\text{Update}$ after initialization. Define $w_{\text{new}}$ as $w_{\text{new}}(e) = 0$ for $e \in Z$ and $w_{\text{new}}(e) = w_e$ otherwise. Then $\text{SC}_H$ satisfies

$$\text{SC}(L(w), C) - e \text{SC}(L(\overline{C}), C) \leq \text{SC}_H,$$

$$\text{SC}_H \leq \text{SC}(L(w), C) + e \text{SC}(L(\overline{C}), C).$$

All outputs and runtimes are correct w.h.p. if $|C| = O(1/m)$ at all times and there are at most $O(1/m)$ total calls to $\text{Update}$.

The proof of Theorem 4.3 is deferred to the full version.

### 5 DATA STRUCTURES FOR DYNAMIC ELECTRICAL FLOWS

The goal of this section is to apply the dynamic Schur complement data structure of Theorem 4.3 to give algorithms that dynamically maintain electric potentials and edges with large electric energies in dynamic electrical flows. In Section 5.1, we will introduce the harmonic extension and use it to decompose the energy vector we need to maintain for the outer IPM. In Section 5.2, we show how to maintain a potential vector which is a key component for the following subsections. In Section 5.3, we build the $\text{Evaluator}$ that estimates the energy of any edge. In Section 5.4, we build the $\text{Locator}$ that returns a superset of edges with large energies.

#### 5.1 Harmonic Extension

A key notion we use throughout is the harmonic extension, which is a linear operator that maps the potentials restricted to a terminal set to the full electric potentials $\phi$. We use $T$ to denote the projection orthogonal to the all-ones vector.

**Definition 5.1 (Harmonic extension).** For a graph $G = (V, E)$ with edge conductances $w \in \mathbb{R}^E$ and $C \subseteq V(G)$, define the harmonic extension operator $H_C \in \mathbb{R}^{E \times C}$ as

$$H_C \overset{\text{def}}{=} \left[ -L(w)^{-1}L(w)F_C T \right].$$

Note that the harmonic extension does not depend on edges with both endpoints in $C$. In this section, we use $\overline{w}$ to denote modified conductances and $w$ to denote initial conductances.

#### 5.2 Dynamic Potential Maintenance

In this section, we show how to maintain a vector $\bar{\phi}$ that approximates the potential vector $\phi$. This data structure can also be used for the $\text{Locator}$.

**Lemma 5.2 (Dynamic Potential).** For a graph $G = (V, E)$ with dynamic edge conductances $\overline{w} \in \mathbb{R}^E$ and a dynamic vector $\overline{v} \in \mathbb{R}^V$ for some constant $C$, there is a data structure that supports the following operations against an oblivious adversary for parameters $\beta < \epsilon^2 < 1$.

- $\text{Initialize}(G, w, \alpha^{(\text{init})}, \beta, \epsilon)$. Initializes the data structure in time $O(1/m^{2/3} - \epsilon^2)$ with an empty set $Z \leftarrow \emptyset$ of marked edges. Initialize $\overline{w}$ as $w$ and $\overline{v}$ as $\alpha^{(\text{init})}$.
- $\text{Update}(e, \overline{w}_{\text{new}})$. Updates $\overline{w}_{\text{new}} \leftarrow \overline{w}_{\text{new}}(e) \in O(\beta^{-2} \epsilon^2)$ time.
- $\text{QueryPotential}(e, \overline{v}_{\text{new}})$. Updates $\overline{v}_{\text{new}} \leftarrow \overline{v}_{\text{new}}(e) \in O(\beta^{-2} \epsilon^2)$ time.

Runtimes and output correctness hold w.h.p. if there are at most $O(1/m)$ calls to $\text{Update}$ and $\text{Update}$ in total.

The proof is deferred to the full version.
5.3 Dynamic Evaluator

For the following sections, we define the electric projection matrix
\[
P_w \defeq W^{1/2} B (T W B) B^T W^{1/2} = W^{1/2} B (T_w B) B^T W^{1/2}.
\]

**Theorem 5.3 (Dynamic Evaluator).** For a graph \( G = (V, E) \) with dynamic edge conductances \( w \in \mathbb{R}_{\geq 0}^{|E|} \) and a dynamic vector \( \overline{v} \in \mathbb{R}^{|E|} \), there is a data structure **Evaluator** that supports the following operations against an oblivious adversary for parameters \( \beta < \epsilon^2 < 1 \):

- **Initialize** \( (G, w, \overline{v}^{(\text{init})}, \beta, \epsilon) \). Initializes the data structure in time \( \tilde{O}(nm\beta^2\epsilon^2) \) with an empty set \( Z \leftarrow \emptyset \) of marked edges. Initializes \( \overline{w} \) as \( \overline{v} \) as \( \overline{v}^{(\text{init})} \).
- **Update** \( (e, \overline{v}^{\text{new}}) \). Updates \( \overline{v} \leftarrow \overline{v}^{\text{new}} \) in \( \tilde{O}(\beta^2\epsilon^2) \) time.
- **Update** \( (e, \overline{w}^{\text{new}}) \). Updates \( \overline{w} \leftarrow \overline{w}^{\text{new}} \) in \( \tilde{O}(\beta^2\epsilon^2) \) time.
- **Query**. Returns a vector \( u \in \mathbb{R}^{|E|} \) satisfying \( ||u - \overline{P}_w \overline{v}||_2 \leq \epsilon \|\overline{P}\|_2 + \epsilon \|\overline{v}\|_2 \) in time \( O(\beta \text{fme}^{-2}) \).

Runtimes and output correctness hold w.h.p. if there are at most \( O(\beta m) \) calls to UpdateV, UpdateW in total.

The proof is deferred to the full version.

5.4 Dynamic Locator

**Theorem 5.4 (Dynamic Locator).** For a graph \( G = (V, E) \) with dynamic edge conductances \( \overline{w} \in \mathbb{R}_{\geq 0}^{|E|} \) and a dynamic vector \( \overline{\sigma} \in \mathbb{R}^{|E|} \), there is a data structure **Locator** that supports the following operations against an oblivious adversary for parameters \( \beta < \epsilon^2 < 1 \):

- **Initialize** \( (G, \overline{w}, \overline{v}^{(\text{init})}, \beta, \epsilon) \). Initializes the data structure in time \( \tilde{O}(nm\beta^2\epsilon^2) \) and sets \( \overline{w} \leftarrow \overline{w} \) and \( \overline{v} \leftarrow \overline{v}^{(\text{init})} \).
- **Update** \( (e, \overline{\sigma}^{\text{new}}) \). Updates \( \overline{v} \leftarrow \overline{\sigma}^{\text{new}} \) in \( \tilde{O}(\beta^2\epsilon^2) \) time.
- **Update** \( (e, \overline{v}^{\text{new}}) \). Updates \( \overline{v} \leftarrow \overline{v}^{\text{new}} \) in \( \tilde{O}(\beta^2\epsilon^2) \) time.
- **Locate**. Returns in time \( \tilde{O}(\text{fme}^{-2}) \) a set \( S \subseteq \mathcal{E}(G) \) with \( |S| \leq O(\epsilon^{-2}) \) containing all edges \( e \) with \( ||P_w \overline{v}_e||_1 \geq \epsilon \|P\|_2 \) w.h.p.

Runtimes and output correctness hold w.h.p. if there are at most \( O(\beta m) \) calls to UpdateV, UpdateW in total.

The proof is deferred to the full version.

6 REDUCING ADAPTIVE TO OBLIVIOUS ADVERSARIES

In this section we show a blackbox reduction that is able to transform any dynamic algorithm that maintains some sequence of vectors \( (\overline{v}^t)_{t \geq 1} \) against oblivious adversaries to one that can maintain the vectors against adaptive adversaries. We formalize the requirements of the dynamic algorithm via Definition 6.1. Roughly, Definition 6.1 states that the dynamic algorithm must support two operations: (i) find the entries of the current vector \( \overline{v}^t \) with large absolute value, and (ii) query some set of the entries approximately.

**Definition 6.1.** We call a dynamic algorithm an \( \epsilon \)-approximate \((L, S)\)-locator for an online* sequence of vectors \( (\overline{v}^t)_{t \geq 1} \), if in each iteration \( t \geq 1 \) the dynamic algorithm returns a set \( S \subseteq [n] \) of size at most \( S \) containing all \( i \) with \( |\overline{v}_i^t| > \epsilon \) in \( L \) time.

*The sequence may depend on outputs of the data structures.

We call a dynamic algorithm an \( \epsilon \)-approximate \( C \)-evaluator, if it supports a query operation that, given some \( i \subseteq [n] \), returns all \( \overline{w}_i \) for \( i \in \mathcal{I} \) in \( O(|\mathcal{I}|) \) time for some \( \overline{w}_i \in \mathbb{R}^m \) with \( \|\overline{w} - \overline{w}^t\|_2 \leq \epsilon \).

We show that, given a locator (a dynamic algorithm that can tell us the large entries), and locators (dynamic algorithms that tells us the entries of the vectors) with different accuracies, we can combine these dynamic algorithms to work against an adaptive adversary. The more accurate locators will be used less frequently, resulting in an expected time complexity faster than the most accurate locator.

**Theorem 6.2.** Assume we have \( \epsilon \)-accurate \((L, S)\)-locator and \( (\epsilon/2^k)\)-accurate \( C_k \)-evaluators for \( k = 0, \ldots, K \). Both dynamic algorithms hold against an oblivious adversary. Also assume there is an \( \epsilon/2^k \)-accurate \( T \)-evaluator against an adaptive adversary.

Then there exists a dynamic algorithm against an adaptive adversary that in each iteration returns w.h.p. some \( \overline{w} \) with \( \|\overline{w} - \overline{w}^t\|_2 \leq O(\epsilon \log^2 n) \). Each iteration takes expected time

\[
O(SK + \frac{T(S)}{2^K} + L + \sum_{i=0}^{K-1} C_i(S))
\]

The proof of Theorem 6.2 is deferred to the full version.

7 INTERIOR POINT METHOD

In this section we provide the machinery we use to reduce minimum cost flow to dynamic graph data structure problems. In Section 7.1 we introduce the data structures, subroutines, and bounds that we develop in this paper to implement this framework efficiently and in Section 7.2 we combine these pieces to give the efficient IPM. The proof of Theorem 7.2 is given in Section 8, and remaining proofs are deferred to the full version.

7.1 Robust IPM Tools

Here we discuss the key tools we develop in this paper to efficiently implement a robust IPM. A key advance of this paper is efficient procedures for approximately maintaining the iterates of a robust IPM algorithm, i.e. approximating the result of approximate projection steps. We formalize this maintenance problem as a data structure problem defined below.

**Definition 7.1 (Solution Maintainer).** We call a data structure a \((T_{\text{init}}, T_{\text{phase}})\)-solution maintainer if it supports the following operations against an adaptive adversary with high probability:

- **Initialize** \( B \in \mathbb{R}^{mxn}, \overline{w}^{(0)} \in \mathbb{R}^m, \overline{x}^{(0)} \in \mathbb{R}^m, \overline{s}^{(0)} \in \mathbb{R}^m, \alpha, C_r, C_z \)\): Given input constraint matrix \( B \), weight vector \( \overline{w}^{(0)} \), iterate \( (\overline{x}^{(0)}, \overline{s}^{(0)}) \), accuracy parameter \( \alpha \), weight range \( r \), phase length \( k \), sparsity of changes \( z \), initialize the data structure with \( \overline{w} := \overline{w}^{(0)}, \overline{x} := \overline{x}^{(0)} \), and \( \overline{s} := \overline{s}^{(0)} \) in time \( O(T_{\text{init}}) \) with \( T_{\text{init}} = \Omega(m) \).

- **Start Phase** \( (\overline{x} \in \mathbb{R}^m, \overline{s} \in \mathbb{R}^m) \): Given input iterate \((\overline{x}, \overline{s}) \) with \( ||W^{-1/2} (\overline{x} - \overline{x}_k)||_1 \leq 1 \) and \( ||W^{-1/2} (\overline{s} - \overline{s}_k)||_2 \leq 1 \), update \( \overline{x} \leftarrow \overline{x} \) and \( \overline{s} \leftarrow \overline{s} \) in amortized \( O(T_{\text{phase}}) \) time with \( T_{\text{phase}} = \Omega(m) \).

- **Move** \( (\overline{w}^{(j)}) \in \mathbb{R}^m, \overline{w}^{(j)} \in \mathbb{R}^m, k^{(j)} \in \mathbb{R} \rightarrow \mathbb{R}^m \times \mathbb{R}^m \): In the \( j \)-th call to Move, given input weights \( \overline{w}^{(j)} \), direction
... and step size \( h^{(j)} \) with \( h^{(j)} \| \phi^{(j)} \|_2 \leq 1 \), Move updates
\[ w \leftarrow w^{(j)}, \]
\[ x \leftarrow x + h^{(j)} W_j^{1/2} (I - P_j) \phi^{(j)}, \]
and \( s \leftarrow s + h^{(j)} W_j^{1/2} P_j \phi^{(j)}, \)
where \( W_j \equiv \text{diag}(w^{(j)}) \) and \( P_j \equiv W_j^{1/2} B (B^T W_j B)^{-1} B^T W_j^{1/2} \) and move outputs \((\tilde{x}^{(j)}, \tilde{s}^{(j)})\) in \( \mathbb{R}^n \times \mathbb{R}^m \) with
\[ \|(W^{(j)})^{-1/2} (\tilde{x}^{(j)} - x)\|_\infty \leq \alpha, \|(W^{(j)})^{1/2} (\tilde{s}^{(j)} - s)\|_\infty \leq \alpha, \]
and the number of coordinates changed from the previous output bounded by \( O(2^{2L} \alpha^2 \log^4 m + S_j) \) where
\[ S_j \equiv \left| \left\{ i \in \{m\} : w^{(j)}_i \neq w^{(j-1)}_i, \tilde{x}^{(j-1)}_i = \tilde{x}^{(j-2)}_i, \text{ and } \tilde{s}^{(j-1)}_i = \tilde{s}^{(j-2)}_i \right\} \right|. \]
The input \( w^{(j)} \) and \( \phi^{(j)} \) and output \((\tilde{x}^{(j)}, \tilde{s}^{(j)})\) are given implicitly as a list of changes to the previous input and output of Move.

Furthermore, the above operations need only be supported under the following assumptions:

1. **Phase length**: \texttt{StartPhase} is called at least every \( k \) calls to \texttt{Move} and at most twice in a row.
2. **Number of changes**: for all \( j \geq 1 \) there are at most \( |C z 2^{j/2} / m| \) coordinates changed in \( w^{(j)}, \phi^{(j)} \) from \( w^{(j-1)}, \phi^{(j-1)} \) where \( \ell_j \) is the largest integer with \( \ell_j \) with \( j \equiv 0 \pmod{2^j} \).
3. **Magnitude of changes**: for any \( |z_j - z_{j-1}| \leq L \), we have
\[ \sqrt{w^{(j)}_i / w^{(j-1)}_i} \leq C_r L^2 \]
for all \( i \in \{m\} \).

Our algorithm actually always has \( S_j = 0 \), but we state Definition 7.1 with possibly nonzero \( S_j \) for more generality.

In the particular case of graphs, one of the key results of this paper is the following efficient solution maintenance data structure in the particular case of graphs (shown in Section 8.1).

**Theorem 7.2 (Graph Solution Maintenance).** In the special case that \( B \) is the incidence matrix of a \( m \)-edge, \( n \)-node graph, if \( C_r, C_z = \tilde{O}(1) \) and \( c_1 = \tilde{O}(1) \), there is a \((T_{\text{init}}, T_{\text{phase}})\)-solution maintainer (Definition 7.1) with \( T_{\text{init}} = \tilde{O}(m) \) and \( T_{\text{phase}} = \tilde{O}(m + m^{15/16} k^{29/8}) \).

Note that in the solution maintenance data structure problem it is required that \texttt{StartPhase} be called at least every \( k \) calls to \texttt{Move}. Consequently, to apply this data structure to implement the robust IPM framework the input \( \tilde{x} \) and \( \tilde{s} \) to \texttt{StartPhase}, i.e. weighted \( \ell_1 \) approximations to \( (x,s) \), need to be computed efficiently. We formalize this problem below.

**Definition 7.3 (Solution Approximation).** We call a procedure \( T_{\text{approx}} \)-approximator if given \( \mu \)-feasible \((x, s)\), weights \( w^{(1)}, \ldots, w^{(k)} \in \mathbb{R}^m \), directions \( \phi^{(1)}, \ldots, \phi^{(k)} \in \mathbb{R}^m \), and step sizes \( h^{(1)}, \ldots, h^{(k)} \) such that
- \( h^{(j)} \| \phi^{(j)} \|_2 \leq 1 \),
- all the changes in \( w \) and \( \phi \) are supported on \( z \) many edges and the input is given as these changes,
- \[ \frac{1}{\gamma} \leq \sqrt{w_i^{(1)} / w_i^{(j)}} \leq r \]
for all \( i, j \in [k] \) and \( t \in [m] \),
with high probability, we can compute \( \mu \)-feasible \((\tilde{x}, \tilde{s})\)
such that
\[ \left\| \tilde{x} - x - \sum_{i\in[k]} h^{(i)} W_i^{1/2} (I - P_i) \phi^{(i)} \right\|_{W_k^{-1}} \leq \epsilon \]
\[ \text{and} \]
\[ \left\| \tilde{s} - s - \sum_{i\in[k]} h^{(i)} W_i^{1/2} P_i \phi^{(i)} \right\|_{W_k^{-1}} \leq \epsilon \]
in \( O(T_{\text{approx}}) \) time where \( W_j \equiv \text{diag}(w^{(j)}) \).

In the particular case of graphs, we have a more efficient solution approximator by using Schur complements. Its proof is deferred to the full version.

**Theorem 7.4.** In the special case that \( B \) is the incidence matrix of a \( m \)-edge, \( n \)-node graph there is \( T_{\text{approx}} \)-approximator with \( T_{\text{approx}} = \tilde{O}(m + z k^3 r^2 \ell^{-2}) \).

### 7.2 Robust IPM Implementation

We state our main theorem on the efficiency of a robust IPM.

**Theorem 7.5 (Informal, See Full Version for a Formal Statement).** Given access to a \((T_{\text{init}}, T_{\text{phase}})\)-solution maintainer, there is an algorithm that outputs a high accuracy solution \((\tilde{x}', \tilde{s}')\) with in time \( \tilde{O}((T_{\text{init}} + \sqrt{m}) (T_{\text{phase}} + T_{\text{approx}})) \).

The proof is deferred to the full version.

### 8 FINAL RUNTIME BOUND

In this section we show Theorem 7.2 which describes how efficiently the data structures we developed in Sections 4 to 6 can implement an IPM step. Our final runtime is then achieved via Theorem 7.5. Finally, we cite previous work to explain how to get an initial point for the IPM, and how to get a mincost flow after running \( \tilde{O}(\sqrt{m}) \) IPM iterations.

### 8.1 Efficient Solution Maintenance

**Theorem 7.2 (Graph Solution Maintenance).** In the special case that \( B \) is the incidence matrix of a \( m \)-edge, \( n \)-node graph, if \( C_r, C_z = \tilde{O}(1) \) and \( \alpha = \tilde{O}(1) \), there is a \((T_{\text{init}}, T_{\text{phase}})\)-solution maintainer (Definition 7.1) with \( T_{\text{init}} = \tilde{O}(m) \) and \( T_{\text{phase}} = \tilde{O}(m + m^{15/16} k^{29/8}) \).

We first make several useful definitions. We let \( x^{(j)}, s^{(j)} \) be the true iterates after the \( j \)-th call to \texttt{Move}. Our algorithm will explicitly approximate iterates \( \tilde{x}^{(j)}, \tilde{s}^{(j)} \). Using these approximate iterates, the algorithm will output \( \tilde{x}^{(j)}, \tilde{s}^{(j)} \) satisfying the desired update schedule, i.e. at most \( \tilde{O}(2^{2L} \alpha^2) \) coordinates are updated after the \( j \)-th call to \texttt{Move}. Additionally, call an index \( j \) corresponding to the \( j \)-th \texttt{Move} operation \texttt{special} if it occurs immediately following a \texttt{StartPhase} operation. The formal construction of \( \tilde{x}^{(j)} \) and \( \tilde{s}^{(j)} \) is given in the following definition.

**Definition 8.1 (Approximate iterates).** Say there have been \( j \) \texttt{Move} operations so far. If the next operation is \texttt{StartPhase}(\( \tilde{x}, \tilde{s} \)), then set \( \tilde{x}^{(j+1)} \leftarrow \tilde{x} \) and \( \tilde{s}^{(j+1)} \leftarrow \tilde{s} \). If the next operation is \texttt{Move}
operation \((j + 1)\), then let \(\delta_x, \delta_s\) satisfy
\[
\delta_x - h^{(j+1)}(1 - P_{j+1})d^{(j+1)} \leq \epsilon \quad \text{and} \quad \delta_s - h^{(j+1)}P_{j+1}d^{(j+1)} \leq \epsilon,
\]
where \(\delta_x, \delta_s\) supported on \(O(\epsilon^{-2})\) coordinates. If the previous operation was Move, define and let \(\tilde{x}^{(j)} \leftarrow x^{(j)} + W_{j+1}^{(j+1)}\delta_x\) and \(\tilde{z}^{(j+1)} \leftarrow s^{(j)} + W_{j+1}^{(j+1)}\delta_s\). Otherwise, if the previous operation was StartPhase define \(\tilde{x}^{(j)} \leftarrow x^{(j)} + W_{j+1}^{(j+1)}\delta_x\) and \(\tilde{z}^{(j+1)} \leftarrow s^{(j+1)} + W_{j+1}^{(j+1)}\delta_s\) (so that we redefine \(\tilde{x}^{(j+1)}, \tilde{z}^{(j+1)}\)).

We show that the \(\tilde{x}^{(j)}\) are slowly changing, except potentially at special indices. This is because \(\|\delta_x\|_2 \leq O(1)\) as it is supported on \(O(\epsilon^{-2})\) nonzeros and \(h^{(j+1)}\|\delta_s^{(j+1)}\|_2 \leq 1\).

We now argue that \(\delta_x, \delta_s\) can be computed efficiently.

**Lemma 8.2 (Computation of \(\delta_x, \delta_s\)).** In the context of Theorem 7.2, there is an operation that computes \(\delta_x, \delta_s\) satisfying (7) in average amortized time \(\tilde{O}(m^{15/16}e^{-7/8})\) and succeeds with high probability against adaptive adversaries.

**Proof.** We first write \(h^{(j+1)}(1 - P_{j+1})d^{(j+1)} = h^{(j+1)}P_{j+1}d^{(j+1)}\), and handle both parts separately up to error \(\epsilon/2\). The first part can be trivially handled, as it can be explicitly maintained in time proportional to the number of changes in \(h^{(j)}\), and \(\|h^{(j+1)}P_{j+1}d^{(j+1)}\|_2 \leq 1\). For the second part, we first call the dynamic Locator (Theorem 5.4) to get a set \(S\) of size \(O(\epsilon^{-2})\). Then we call the dynamic Evaluators (Theorem 5.3) wrapped inside Theorem 6.2 with \(\epsilon \leftarrow \epsilon/(C\log^2 n)\) on \(S\) by calling Query() on \(S\). The algorithm for \(\delta_x\) follows exactly as the second term. Also, \(\delta_x, \delta_s\) are supported on \(|S| \sim O(\epsilon^{-2})\) coordinates by Theorem 6.2.

Correctness follows directly from the guarantees of Theorems 5.3, 5.4 and 6.2. It suffices to analyze the amortized runtime. We focus on the cost of applying Theorem 5.3 inside Theorem 6.2, as the cost of Theorem 5.4 is less. Let \(\delta_x \leftarrow \tilde{\epsilon}^{-2}\) so that the \(j\)-th Evaluators is run with accuracy \(\epsilon_x \leftarrow \delta_x \epsilon\) in Theorem 6.2. Let \(\beta_i\) be the terminal size parameter for the \(i\)-th Evaluators.

There are two possible ways to run the \(i\)-th Evaluators. Either it pays \(\tilde{O}(m)\) time per call to solve a Laplacian exactly (while this algorithm is randomized, we can hide randomness by adding polynomially small noise that is larger than the error we solve the Laplacian to [54]) or applies Theorem 5.3. Let us calculate the runtime of the latter approach. After \(\beta_i m\) edge updates or marking, the data structure must re-initialize. Thus, after \(T\) Move updates, because \(C_2 \sim O(1)\), there are at most \(O(T^2 + T\epsilon^{-2})\) total edges we have queried or updated. \(\tilde{O}(T^2)\) from updates, and \(\tilde{O}(T\epsilon^{-2})\) from the set returned by Locator. We assume for now that the \(\tilde{O}(T^2)\) term dominates – thus the data structure must reinitialize every \(\sqrt{\beta_i m}\) iterations, where each initialization costs \(O(\beta_i m^{1/2} \epsilon^{-2})\) time. Thus the amortized reinitialization time per Move is \(\tilde{O}(m^{1/2} \epsilon^{-2}/\sqrt{\beta_i m}) = \tilde{O}(\sqrt{\beta_i m^{1/2} \epsilon^{-2}})\).

By Theorem 6.2, the \(i\)-th Evaluators is queried with probability \(O(\delta_i)\), hence the expected query time is \(\tilde{O}(\beta_i m^{1/2} \epsilon^{-2}) = \tilde{O}(\beta_i m^{1/2} \epsilon^{-2})\).

By Theorem 5.3 Query(), or \(\tilde{O}(\delta_i m)\) if Evaluators simply solves a Laplacian every iteration. Thus, the amortized runtime for the \(i\)-th Evaluators is
\[
\tilde{O}\left(\min\left\{\delta_i m, \beta_i m^{1/2} \epsilon^{-2} \delta_i^{-1}\right\} + \sqrt{\beta_i m^{1/2} \epsilon^{-2} \delta_i^{-2}}\right).
\]

For the choice \(\beta_i = m^{-1/2} \epsilon^{-2}\), this becomes
\[
\tilde{O}\left(\min\left\{\delta_i m, m^{6/7} \epsilon^{-2} \delta_i^{-2}\right\}\right).
\]

This is maximized when the two expressions are equal at \(\delta_i = m^{-1/16} \epsilon^{-7/8}\), yielding a runtime of \(\tilde{O}(m^{15/16} \epsilon^{-7/8})\) as desired. Finally, note that this means that \(\epsilon \gtrsim m^{-1/4}\) or the previous runtime is trivial. All \(\beta_i \gtrsim m^{-1/7}\), so \(T \epsilon^{-2} \lesssim T^2\) for the choice \(T = \sqrt{\beta_i m} \gtrsim \epsilon^{-2}\), so the \(\tilde{O}(T^2)\) term dominated earlier, as desired.

We now show that \(\tilde{x}^{(j)}\) and \(\tilde{z}^{(j)}\) are close to \(x^{(j)}, s^{(j)}\).

**Lemma 8.3.** For \(\epsilon = \frac{a}{10\log n}\) and \(\tilde{x}^{(j)}, \tilde{z}^{(j)}\) defined in Definition 8.1,
\[
\left\|W_j^{(j)} \left(\tilde{x}^{(j)} - x^{(j)}\right)\right\|_\infty \leq \alpha/10 \quad \text{and} \quad \left\|W_j^{(j)} \left(\tilde{z}^{(j)} - s^{(j)}\right)\right\|_\infty \leq \alpha/10.
\]

**Proof.** It suffices to analyze \(j\) between StartPhases, as \(\tilde{x}^{(j)}, \tilde{z}^{(j)}\) and \(x^{(j)}, s^{(j)}\) are both set to \(\tilde{x}, \tilde{z}\) during a StartPhase. For \(L\) steps between StartPhases (from \(j_1\) to \(j_2 = j_1 + L\)), we have
\[
\left\|W_j^{(j)} \left(\tilde{x}^{(j)} - x^{(j)}\right)\right\|_\infty \leq \max_{j \in [j_1, j_2]} \left\|W_j^{(j)} \left(\tilde{x}^{(j)} - x^{(j)}\right)\right\|_\infty \leq \frac{a}{10L^2} \epsilon C L^3 \leq \alpha/10,
\]
where \((i)\) follows from the guarantee that of Definition 7.1 that \(\|w_{j_1}\|_{\infty} \leq rL^2\) and (7). The bound on the error for \(s^{(j)}\) follows similarly.

**Proof of Theorem 7.2.** We show this by carefully defining \(\tilde{x}^{(j)}\) and \(\tilde{z}^{(j)}\) given \(x^{(j)}, s^{(j)}\). We mimic the approach based on binary expansions given in previous works on robust IPMs, for example [16, Theorem 8]. Precisely, we first calculate \(\sum_{j' = j-2^{d_j}}^{j} g_j^{(j')}\), i.e. the sum of errors in the last \(2^{d_j}\) steps. If this exceeds \(\frac{a}{10\log n}\), then we set \(\tilde{x}^{(j)} \leftarrow \tilde{x}^{(j)}\), otherwise we set \(\tilde{x}^{(j)} \leftarrow \tilde{x}^{(j-1)}\) (no change). We do the same for \(\tilde{z}^{(j)}\). Now, the bounds on number of changes follows from the bounds \(\|\delta_x\|_2 \leq O(1)\) and \(\|W_j^{(j)}(x - \tilde{x})\|_2 \leq 1\) in StartPhase. More precisely, over \(2^d\) steps, only \(O(2^{d_0} a^{-2} \log^2 n)\) could change by \(\frac{a}{10\log n}\), because every step satisfies \(\|\delta_x\|_2 \leq 1\) or \(\|W_j^{(j)}(x - \tilde{x})\|_2 \leq 1\). This completes the proof of the number of changes.

We now claim that \(\tilde{x}^{(j)} - x^{(j)}\) is close to \(\tilde{x}^{(j)} - x^{(j)}\)
by combining with Lemma 8.3 for \( \epsilon = \alpha / (100 \log n) \) and \( \gamma = \Theta(k^2) \) as \( C_{\gamma} = \Theta(1/k^3) \) as \( C_{\gamma} = \Theta(1) \). This claim follows from the same argument as [16, Theorem 8]: each interval \( [j, j + 1] \) can be split into \( \log n \) intervals contained in intervals \( [j - 2^i, j] \) for \( j \leq j \). Each of these has at most \( \alpha/(100 \log n) \) error, so the total error is at most \( \alpha/(100 \log n) \cdot \log n \leq \alpha/10 . \)

Finally, we must calculate the runtime of \( T_{\text{phase}} \). The first cost is \( \tilde{O}(m) \) (eg. for reading \( \tilde{x}, \tilde{s} \)). The second cost is calling Lemma 8.2 \( k \) times (as there are at most \( k \) Move operations between START-PHASE). For our choice \( \epsilon = \Theta(1/k^3) \), the total time for this is \( \tilde{O}(m^{15}/(16 k^{29/8})) \) as desired. \( \square \)

8.2 Initial Point, Final Point, and Proof of Main Theorem

It is standard to get an initial point \((x, s)\) for a polynomially bounded path parameter, i.e. at most \((mU)^{O(1)} \). Additionally, given a feasible pair for small path parameter at most \((mU)^{-O(1)} \) we can recover a high-accuracy mincost flow (and hence round to an exact solution).

**Lemma 8.4** [14, Lemma 7.5, Lemma 7.8]. Given a graph \( G = (V, E) \) and mincost flow instance with demand \( d \in [-U, \ldots, U]^V \), costs \( c \in [-U, \ldots, U]^E \), and capacities \( \ell, u \in [-U, \ldots, U]^E \), we can build a mincost flow instance on a graph \( G' \) with at most \( O(m) \) edges with demands, costs, and capacities bounded by poly\((mU)\). Additionally, we can construct an initial flow, and given a 1/poly\((mU)\)-accurate mincost flow on \( G' \) we can recover an exact mincost flow on \( G \) in time \( \tilde{O}(m \log U) \).

**Proof of Theorem 1.1.** We apply Lemma 8.4 to get an initial point for the algorithm. Then, we run the algorithm of Theorem 7.5 and round to an exact mincost flow using Lemma 8.4. This succeeds by Theorem 7.5 in time

\[
\tilde{O}\left( \sqrt{m/k} \left( T_{\text{phase}} + T_{\text{approx}} \right) \log m \right)
\]

It suffices to plug in the values of \( T_{\text{init}}, T_{\text{phase}}, T_{\text{approx}} \) from Theorem 7.2 and 7.4.

We take \( k = m^{1/58} \) so \( T_{\text{phase}} = \tilde{O}(m^{15/16 k^{29/8}} + m) = \tilde{O}(m) \) by Theorem 7.2. Also by Theorem 7.4, \( T_{\text{approx}} = \tilde{O}(m + k^{23}/k^2) = \tilde{O}(m + k^{13}) = \tilde{O}(m) \) for \( m \leq m^{1/58} \). Thus, the expression in (9) evaluates to \( \tilde{O}(m^{35/2 - 1/58} \log U) \) as desired. \( \square \)

**Acknowledgments**

Aaron Sidford is supported in part by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship. Yang P. Liu is supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship, and NSF CAREER Award CCF-1844855 and NSF Grant CCF-1955039. Yin Tat Lee is supported in part by NSF awards CCF-1749609, DMS-1839116, DMS-2023166, CCF-2105772, a Microsoft Research Faculty Fellowship, a Sloan Research Fellowship, and a Packard Fellowship. Jan van den Brand is funded by ONR BRC grant N00014-18-1-2562 and by the Simons Institute for the Theory of Computing through a Simons-Berkeley Postdoctoral Fellowship. Part of this work was done while Jan van den Brand, Yang P. Liu and Aaron Sidford were visiting the Discrete Optimization Trimester Program at the Hausdorff Research Institute for Mathematics.

**References**

[1] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. 2019. Iterative refinement for \( \ell_p \)-norm regression. In Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 1405–1424.

[2] Hilal Asi, Yair Carmon, Arun Jambulapati, Yuja Jin, and Aaron Sidford. 2021. Stochastic Bias-Reduced Gradient Methods. Advances in Neural Information Processing Systems 34 (2021).

[3] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. 2020. Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs. In FOCS. IEEE, 93–104.

[4] Amos Beimel, Haim Kaplan, Yishay Mansour, Kobi Nissim, Thachphol Saranurak, and Uri Stemmer. 2021. Dynamic Algorithms Against an Adaptive Adversary: Generic Constructions and Lower Bounds. CoRR abs/2101.03980 (2021).

[5] Omri Ben-Eliezer, Talya Eden, and Krzysztof Onak. 2021. Adversarially Robust Streaming via Dense-Sparse Trade-offs. CoRR abs/2109.03785 (2021).

[6] Aaron Bernstein, Maximilian Probst Gutenberg, Danupon Nanongkai, Thachphol Saranurak, Aaron Sidford, and He Sun. 2020. Fully-Dynamic Graph Sparsifiers Against an Adaptive Adversary. CoRR abs/2004.08432 (2020). Available at https://arxiv.org/abs/2004.08432.

[7] Aaron Bernstein and Shiri Chechik. 2016. Deterministic decremental single source shortest paths: beyond the \( O(m) \) bound. In Proceedings of the Forty-eighth annual ACM symposium on Theory of Computing. 389–397.

[8] Aaron Bernstein and Shiri Chechik. 2017. Deterministic partially dynamic single source shortest paths for sparse graphs. In Proceedings of the Twenty-Eighth Annual ACM-SIAM symposium on Discrete algorithms. SIAM, 453–469.

[9] Aaron Bernstein, Maximilian Probst Gutenberg, and Thachphol Saranurak. 2021. Deterministic Decremental SSSP and Approximate Min-Cost Flow in Almost-Linear Time. arXiv preprint arXiv:2101.07149 (2021).

[10] Jose H Blanchet and Peter W Glynn. 2015. Unbiased Monte Carlo for optimization and functions of expectations via multi-level randomization. In 2015 Winter Simulation Conference (WSC). IEEE, 3656–3667.

[11] Jan van den Brand. 2020. A Deterministic Linear Program Solver in Current Matrix Multiplication Time. In SOCG. SIAM, 259–278.

[12] Jan van den Brand. 2021. Unifying Matrix Data Structures: Simplifying and Speeding up Iterative Algorithms. In SOSA. SIAM, 1–13.

[13] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thachphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2021. Minimum cost flows, MDPs, and \( \ell_1 \)-regression in nearly linear time for dense instances. In STOC. ACM, 859–869.

[14] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thachphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2020. 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. 919–930. https://doi.org/10.1109/FOCS46700.2020.00090

[15] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. 2011. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In Proceedings of the 53rd Annual ACM-SIAM Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020. ACM, 775–788. Available at https://arxiv.org/abs/2002.02304.

[16] Julia Chuzhoy, Shiri Chechik, and Uri Stemmer. 2021. Dynamic Algorithms Against an Adaptive Adversary: Generic Constructions and Lower Bounds. In Proceedings of the 53rd Annual ACM Symposium on Theory of Computing, STOC 2021, San Jose, CA, USA, June 6-8 2021. ACM, 273–282. Available at https://arxiv.org/abs/1039.2921.

[17] Julia Chuzhoy. 2021. Decentralized all-pairs shortest paths in deterministic near-linear time. In Proceedings of the 53rd Annual ACM Symposium on Theory of Computing, 626–639.

[18] Julia Chuzhoy, Yin Tat Lee, Jason Li, Danupon Nanongkai, Richard Peng, and Thachphol Saranurak. 2020. 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, 1158–1167. https://doi.org/10.1109/FOCS46700.2020.00111. Available at https://arxiv.org/abs/1910.08025.

[19] Julia Chuzhoy and Sanjeev Khanna. 2019. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In Proceedings of the 31st Annual ACM-SIAM Symposium on Theory of Computing. 389–400.
Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. 2017. Dynamic Minimum Spanning Forest with Subpolynomial Worst-Case Update Time. In 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017. IEEE Computer Society, 950–961. Available at https://arxiv.org/abs/1708.03962.

Jelani Nelson and Huacheng Yu. 2020. Optimal bounds for approximate counting. arXiv preprint arXiv:2010.02116 (2020).

Richard Peng. 2016. Approximate Undirected Maximum Flows in O(m 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, 1862–1867. https://doi.org/10.1137/1.9781611974331.ch130

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

Thatchaphol Saranurak and Di Wang. 2019. Expander Decomposition and Pruning: Faster, Stronger, and Simpler. In Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019. SIAM, 2616–2635. Available at https://arxiv.org/abs/1812.08958.

Aaron Schild. 2018. 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, 214–227. Available at https://arxiv.org/abs/1711.06455.

Jonah Sherman. 2017. Area-convexity, ℓ∞ regularization, and undirected multi-commodity flow. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, 452–460.

Aaron Sidford and Kevin Tian. 2018. 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. 922–933. Available at https://arxiv.org/abs/1808.01278.

Daniel A. Spielman and Shang-Hua Teng. 2004. 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, 452–460.

Pravin M. Vaidya. 1989. 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, 332–337. https://doi.org/10.1109/SFCS.1989.63499

Christian Wulff-Nilsen. 2017. 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. 1130–1143. Available at https://arxiv.org/abs/1611.02864.