Incremental Approximate Maximum Flow on Undirected Graphs in Subpolynomial Update Time*

Jan van den Brand† Li Chen‡ Rasmus Kyng§ Yang P. Liu¶ Richard Peng‖
Maximilian Probst Gutenberg** Sushant Sachdeva†† Aaron Sidford‡‡

Abstract

We provide an algorithm which, with high probability, maintains a $(1-\epsilon)$-approximate maximum flow on an undirected graph undergoing m-edge additions in amortized $m^{o(1)}\epsilon^{-3}$ time per update. To obtain this result, we provide a more general algorithm that solves what we call the *incremental, thresholded, p-norm flow problem* that asks to determine the first edge-insertion in an undirected graph that causes the minimum ℓ_p-norm flow to decrease below a given threshold in value. Since we solve this thresholded problem, our data structure succeeds against an adaptive adversary that can only see the data structure's output. Furthermore, since our algorithm holds for $p = 2$, we obtain improved algorithms for dynamically maintaining the effective resistance between a pair of vertices in an undirected graph undergoing edge insertions.

Our algorithm builds upon previous dynamic algorithms for approximately solving the minimum-ratio cycle problem that underlie previous advances on the maximum flow problem [Chen-Kyng-Liu-Peng-Probst Gutenberg-Sachdeva, FOCS '22] as well as recent dynamic maximum flow algorithms [v.d.Brand-Liu-Sidford, STOC '23]. Instead of using interior point methods, which were a key component of these recent advances, our algorithm uses an optimization method based on ℓ_p-norm iterative refinement and the multiplicative weight update method. This ensures a monotonicity property in the minimum-ratio cycle subproblems that allows us to apply known data structures and bypass issues arising from adaptive queries.

1 Introduction

The design and analysis of dynamic graph algorithms is a rich, well-studied research area. Researchers have studied dynamic variations of fundamental graph problems such as minimum spanning tree, shortest path, and bipartite matching. Recent progress on dynamic matching has been widely celebrated [10, 23], and dynamic graph algorithms play a key role in recent advances in static graph algorithms, yielding the first nearly-linear time algorithms for bipartite matching and maximum flow (maxflow) in dense graphs [30, 29] and almost-linear time algorithm for maxflow [36]. (See Section 1.2, for additional related work.)

Despite the many successes of dynamic graph algorithms, *dynamic maxflow*, another class of core problems, has seen relatively little progress. This is perhaps due in part to strong conditional hardness results. In the incremental and decremental settings, where, respectively, edges are only added or only removed, exactly maintaining the s-t flow value requires $\Omega(n)$ time per update, even in directed unit capacity graphs, assuming the Online-Matrix-vector (OMv)-conjecture. This was shown by [40], building on earlier work by [60] that introduced the OMv-conjecture and showed a $O(\sqrt{m})$ time per update lower bound for the same problems.

*The full version of the paper can be accessed at https://arxiv.org/abs/2311.03174

†Georgia Tech. vdbrand@gatech.edu

‡Georgia Tech. lichen@gatech.edu. Li Chen was supported by NSF Grant CCF-2106444.

§ETH Zurich. kyng@inf.ethz.ch. The research leading to these results has received funding from the grant "Algorithms and complexity for high-accuracy flows and convex optimization" (no. 200021 204787) of the Swiss National Science Foundation.

¶Stanford University. yangpliu@stanford.edu. Yang P. Liu was supported by NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, and a Google Research Fellowship.

‖University of Waterloo. y5peng@uwaterloo.ca. Richard Peng was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2022-03207

**ETH Zurich. maxprobst@ethz.ch.

††University of Toronto. sachdeva@cs.toronto.edu. Sushant Sachdeva was supported by an NSERC Discovery Grant RGPIN-2018-06398, an Ontario Early Researcher Award (ERA) ER21-16-284, and a Sloan Research Fellowship.

‡‡Stanford University. sidford@stanford.edu. Aaron Sidford was supported by a Microsoft Research Faculty Fellowship, NSF CAREER Award CCF-1844855, NSF Grant CCF-1955039, a PayPal research award, and a Sloan Research Fellowship.

Moving to approximate solutions opens the possibility of faster algorithms. Recently, [53] gave an $(1-\epsilon)$-approximation algorithm for incremental unit capacity maxflow with amortized update time $\widehat{O}(\epsilon^{-1/2}\sqrt{m})$[1], an improvement for sparse graphs in the low accuracy regime. Additionally, [91] recently gave an $(1-\epsilon)$-approximation algorithm for incremental maxflow and minimum-cost flow[2] with amortized update time $\widehat{O}(\epsilon^{-1}\sqrt{n})$ by dynamizing and building upon the recent almost-linear time algorithm for maxflow of [36]. For constant ϵ, this runtime is faster than the conditional lower bound for exact incremental maxflow.

In this work, we ask whether *we can build upon this recent progress and give a subpolynomial amortized update time algorithm for the dynamic maxflow problem?* We answer this in the affirmative by developing such an algorithm for approximate incremental undirected maxflow.

Dynamizing Static Maxflow. To obtain our result, we build upon the recent advance of [91] and the work that underlies it, as well as distinct lines of research related to approximate undirected maxflow. [91] essentially dynamizes the almost linear-time algorithm for maxflow [36]. [36] came at the end of a long line of research that focused on solving flow problems by combining graph theoretic tools with interior point methods (IPMs), a class of continuous optimization methods which obtain high-accuracy solutions to convex optimization problems [41, 78, 75, 79, 39, 77, 68, 7, 30, 29, 52, 8, 46, 28, 36]. The [36] IPM relies on solving an ℓ_1 flow update problem known as "(undirected) min-ratio cycle."[3] This problem is solved $m^{1+o(1)}$ times using a data structure with amortized $m^{o(1)}$ time per update. [91], showed how to dynamize this IPM, but obtained an $n^{1/2+o(1)}$ amortized time per update due to the cost of adapting the min-ratio cycle data structures to this setting.

There are two key ideas in [91]. The first is that the ℓ_1-IPM of [36] can be naturally extended to the incremental setting. In particular, they showed how the IPM can be used to solve a threshold variant of the incremental maximum flow problem, i.e., detecting the first update that causes the maximum flow value to increase above a threshold. However, this extension creates a challenge: The [36] data structure for min-ratio cycle does *not* work against an adaptive adversary. Instead, [36] crucially leverages stability properties of their ℓ_1 IPM that ultimately determines the update problems, and uses these stability properties to guarantee that while their data structure may occasionally fail, this will occur infrequently. In the incremental setting, [91] cannot leverage this guarantee to ensure their data structure works. This leads to the second central idea of [91]. They develop a new version of the [36] data structure that works against adaptive adversaries, at the expense of increasing the amortized time per update from $m^{o(1)}$ to $n^{1/2+o(1)}$.

At a high level, our approach is motivated by [91], but ultimately we develop a fundamentally different optimization approach, which yields more tractable update problems. A key observation enabling our algorithms, is that the min-ratio cycle problem succeeds against adaptive adversaries provided that there is a particular monotonicity in the updates. We change the optimization framework to yield subproblems which can be solved by such monotonic updates. Because of this, we manage to show that in our setting, the data structure of [36] can directly solve the update problems in the incremental setting, with only $m^{o(1)}$ amortized time per update.

It is worth mentioning that the authors of this work recently gave a deterministic min-cost flow algorithm [27]. However, [27] still uses a version of the data structure of [36] that does not work against adaptive adversaries, and critically still uses stability properties of the update sequence to argue correctness.

ℓ_p-norm Flow and Approximate Undirected Maxflow. To leverage that the min-ratio cycle data structure succeeds against monotonic adversaries, we turn to an approach motivated by lines of research on algorithms for (static) ℓ_p-norm flow and approximate undirected maxflow. One important line of research yielded undirected approximate maxflow in $\widetilde{O}(\epsilon^{-1}m)$ time [37, 84, 69, 81, 85]. This sequence of works combined first-order continuous optimization methods with graph theoretic tools. A second line of work focused on a broader class of flow problems, namely ℓ_p-norm flows [2, 4, 3, 73, 1], and developed iterative optimization methods tailored to ℓ_p-norm objectives. The problem of ℓ_p-norm flows asks for a flow $\boldsymbol{f}$ that routes a given demand and minimizes its ℓ_p-norm, $\|\boldsymbol{f}\|_p$. This problem is maxflow for $p=\infty$ and setting $p = O(\epsilon^{-1}\log m)$ yields $(1-\epsilon)$-approximate undirected maxflow.

Our Approach to Incremental Flow Problems. We show that the desired monotonicity properties for the subproblems can be achieved in the setting of ℓ_p-norm flows by adapting the optimization framework. Even

[1] In this paper, we use $\widehat{O}(\cdot)$ to suppress subpolynomial $n^{o(1)}$ factors.

[2] In both cases, this is assuming polynomially bounded capacities; we make this assumption throughout the paper.

[3] In this paper we refer to this problem as *min-ratio cycle*, omitting the term "undirected." Elsewhere in the literature, but never in this paper, *min-ratio cycle* may refer to a variant with edge-direction constraints.

though ℓ_p-norm flow is less general than directed maxflow, it has interesting consequences including approximate undirected maxflow and effective resistances [37, 87].

Switching to ℓ_p-norm flows allows us to use the ℓ_p iterative refinement framework developed in [2, 4, 3], and study the smoothed ℓ_p-norm flow problems of [73]. The iterative refinement framework shows that smoothed ℓ_p-norm flow computation can be accomplished by a small number of iterations of a *refinement step*. In our context $\widehat{O}(p)$ steps suffice.

To solve each refinement step problem, we use a ℓ_1 multiplicative weight update method (MWU). Our method lets us solve smoothed ℓ_p-norm flow to $m^{o(1)}$ accuracy by solving a min-ratio cycle problem roughly $m^{1+o(1)}$ times. Crucially, we show that our MWU induces a *monotonicity* property in our min-ratio cycle problems. Concretely, our sequence of approximate min-ratio cycle problems only change by (1) edge insertions and (2) edge lengths increases (see problem 5.1). In this way, we create a more tractable data structure problem than those in [91], and we show that this problem can be solved using data structures from [36] with $m^{o(1)}$ amortized update time.

Finally, combining our monotonic ℓ_1-MWU for computing refinement steps with iterative refinement, we obtain an incremental algorithm for (a decision version of) smoothed ℓ_p-norm flows. From this, we derive an algorithm for approximate incremental undirected maxflow and for incremental electrical flow. Using the incremental electrical flow algorithm, for a fixed pair of vertices s, t, we can detect in an incremental graph the first time the effective resistance between s and t drops below a given threshold.

If we were only focusing on approximate incremental maxflow instead of the more general ℓ_p-norm flows, a similar monotonic data structure problem could also be obtained by using an ℓ_1-oracle MWU to compute each update step of a first-order optimization method such as one used by [84, 69, 81, 85]. Wrapping this inside a first-order ℓ_∞ optimization approach would then yield a similar algorithm for approximate incremental maxflow.

1.1 Results Here we present the main results of the paper. This section leverages a variety of notation, in particular graph theory conventions, all provided later in section 2.

A central result of this paper is an algorithm with subpolynomial update time for the following *undirected incremental approximate maxflow problem* (which we abbreviate as *incremental maxflow* in the remainder of the paper). Incremental maxflow is the dynamic data structure problem of maintaining a $(1+\epsilon)$-(multiplicative) approximate maximum flow in an undirected graph undergoing edge additions (hence the term incremental).

PROBLEM 1.1. (UNDIRECTED INCREMENTAL APPROXIMATE MAXFLOW PROBLEM) *In the* undirected incremental approximate maxflow problem (incremental maxflow) *we are given a finite set of n vertices V, a distinct pair of elements s and t, and a parameter $\epsilon > 0$. There are then a sequence of $m \leq \text{poly}(n)$ edge insertions where starting from $E = \emptyset$ an undirected edge e is added to E with integral capacity $\boldsymbol{u}_e \in [1, U]$. The algorithm must maintain a $(1+\epsilon)$-approximate maximum flow in the capacitated graph $G = (V, E, \boldsymbol{u})$ before and after each edge addition.*

The main result of this paper is a randomized algorithm for the incremental maxflow problem that succeeds with high probability in n (whp.) and implements each update in amortized $n^{o(1)}\epsilon^{-3}$ time. This is the first subpolynomial update time for an incremental maxflow problem which achieves even constant approximation.

Before stating our result, we comment on the adversary model. In our algorithms, we assume that the adversary can see the output flow, but not the internal randomness or information stored in the data structure. We call this an *adaptive adversary*. We refer to the stronger adversary which can also see the internal randomness as a *non-oblivious adversary*.

THEOREM 1.1. (INCREMENTAL MAXFLOW) *There is an algorithm which solves incremental maxflow (Problem 1.1) whp. in amortized $n^{o(1)}\epsilon^{-3}$ time per update against adaptive adversaries.*

To obtain this result, we develop dynamic algorithms for the problem of computing smoothed ℓ_p-norm flows [73] for $p \geq 2$. The *smoothed ℓ_p-norm flow problem* asks to find a flow routing given vertex demands while minimizing a linear plus quadratic plus p^{th} power objective on the flow. This problem generalizes both the popular and prevalent problems of computing electric flows (and therefore solving Laplacian systems) [88, 71, 38, 66] as well as computing approximate maximum flows on undirected graphs [37, 84, 69, 81, 85].

PROBLEM 1.2. (SMOOTHED ℓ_p-NORM FLOW) *Given an undirected graph $G = (V, E)$, gradient vector $\boldsymbol{g}^G \in \mathbb{R}^E$, edge resistances and weights $\boldsymbol{r}^G, \boldsymbol{w}^G \in \mathbb{R}^E_+$, and demand vector $\boldsymbol{d} \in \mathbb{R}^V$ the* smoothed ℓ_p-norm flow problem *asks to solve the following optimization problem*

$$OPT = \min_{\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{d}} \mathcal{E}(\boldsymbol{f}) \quad \textit{for} \quad \mathcal{E}(\boldsymbol{f}) \stackrel{\text{def}}{=} \langle \boldsymbol{g}^G, \boldsymbol{f} \rangle + \left\| \mathbf{R}^G \boldsymbol{f} \right\|_2^2 + \left\| \mathbf{W}^G \boldsymbol{f} \right\|_p^p . \tag{1.1}$$

$f \in \mathbb{R}^E$ is said to be feasible *or* routes the demands *if $\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{d}$, $\mathcal{E}(\boldsymbol{f})$ is called the* energy *or* smoothed objective value *of $\boldsymbol{f}$, and a solution to (1.1) is called a* smoothed ℓ_p-norm flow.

For short, we refer to smoothed ℓ_p-norm flows as simply *ℓ_p-norm flows* throughout. Throughout we also assume that there is a feasible flow $\boldsymbol{f}^{(0)}$ on the initial graph. This can be ensured by determining the first instance that $\boldsymbol{d}$ is feasible using a simple union-data data structure. This incurs only an additive $\widetilde{O}(1)$ cost in our data structures.

Our main technical result is a high-accuracy algorithm with subpolynomial update time for the following *incremental thresholded ℓ_p-norm flow* problem, or *incremental ℓ_p-norm flow* for short. The incremental ℓ_p-norm flow data structure detects the earliest moment when the optimal value to (1.1) drops below a given threshold F.

PROBLEM 1.3. (INCREMENTAL THRESHOLDED ℓ_p-NORM FLOW) *Consider a dynamic instance of a ℓ_p-Norm Flow $(G, \boldsymbol{g}^G, \boldsymbol{r}^G, \boldsymbol{w}^G, \boldsymbol{d})$ that is subject to edge insertion. Let m be the final number of edges in G. Given an objective threshold $F \in \mathbb{R}$, and an error parameter $\epsilon > 0$, the problem of* Incremental Threshold ℓ_p-Norm Flow *asks, after the initialization or each edge insertion, to either*

1. *correctly certify that $OPT > F$, or*
2. *output a feasible flow $\boldsymbol{f}$ with $\mathcal{E}(\boldsymbol{f}) \leq F + \epsilon$.*

Combining an ℓ_1-MWU with the dynamic min-ratio data structure of [36], we can solve problem 1.3 in almost linear time whp. against an adaptive adversary.

THEOREM 1.2. *There is a randomized algorithm for problem 1.3 that given an initial flow $\boldsymbol{f}^{(0)}$ and inputs $\epsilon, \boldsymbol{g}^G, \boldsymbol{r}^G, \boldsymbol{w}^G$ and $\boldsymbol{d}$ with sizes bounded by $\widetilde{O}(1)$ in fixed point arithmetic., runs in $p^2 m^{1+o(1)} \log(\frac{\mathcal{E}(\boldsymbol{f}^{(0)}) - F}{\epsilon})$ time and succeeds whp. against adaptive adversaries.*

As a result, taking $p = 2$ yields an algorithm for $(1+\epsilon)$-approximate incremental electrical flow with subpolynomial update time. This is the first subpolynomial time algorithm for constant accuracy incremental electrical flows.

We show how theorem 1.2 leads to the incremental maxflow algorithm.

Proof. [Proof of Theorem 1.1] The algorithm proceeds in about $\widetilde{O}(\epsilon^{-1})$ phases. In each phase, we determine when the congestion of the optimal flow has decreased by at least a $(1-\epsilon)$ factor. At the start of such a phase, we find the optimal maxflow $\boldsymbol{f}$ using the almost-linear time algorithm of [36]. Let C be the congestion of $\boldsymbol{f}$. We wish to determine when the optimal congestion is smaller than $e^{-\epsilon}C$. From the previous discussion, any flow of congestion less than $e^{-\epsilon}C$ must have its ℓ_p-norm at most $m(e^{-\epsilon}C)^p$. Our flow $\boldsymbol{f}$, on the other hand, has p-norm at most mC^p. So we apply theorem 1.2 with threshold $F = m(e^{-\epsilon}C)^p$ and error $m(e^{-\epsilon}C)^p$ as well. When the data structure certifies that every flow has ℓ_p-norm at least $m(e^{-\epsilon}C)^p$, we know that $\boldsymbol{f}$ is still a $(1-\epsilon)$-approximate maxflow. When the data structure outputs a new feasible flow $\boldsymbol{f}'$ s.t. $\mathcal{E}(\boldsymbol{f}') \leq F + m(e^{-\epsilon}C)^p = 2m(e^{-\epsilon}C)^p$, we know the congestion of $\boldsymbol{f}'$ is at most $(2m(e^{-\epsilon}C)^p)^{1/p} \leq e^{-\epsilon/2}C$. This means that the optimal congestion drops at least by a factor of $(1+\epsilon/2)$ and we restart the whole algorithm with a newly computed maximum flow. It restarts at most $\widetilde{O}(1/\epsilon)$ times because the congestion is between $[\exp(-\widetilde{O}(1)), \exp(\widetilde{O}(1))]$. ∎

Bit Complexity. The number of exact arithmetic operations performed in the algorithm for theorem 1.2 is only $pm^{1+o(1)} \log(\frac{\mathcal{E}(\boldsymbol{f}^{(0)}) - F}{\epsilon})$. The additional p dependency arises as our algorithms manipulate numbers of size $\widetilde{O}(p)$ due to the p-th power taken in the objective (1.1) and we are using fixed point arithmetic. A potential way to improve the dependency on p is via floating point arithmetic and crude approximations. That is, during the course of the algorithm for theorem 1.2, we only use $(1 + \exp(-\widetilde{O}(1)))$-multiplicative approximation to the numbers we encountered and use an additional $\widetilde{O}(p)$ bits to represent their scales. This way, the numbers encountered would be of size $\widetilde{O}(1)$ and we could shave one factor of p from the runtime in theorem 1.2. This could also improve the amortized update time for incremental approximate maximum flow to $n^{o(1)}\epsilon^{-2}$.

1.2 Additional Related Work Dynamic Flows and Matching. Exact dynamic maxflow on unit capacity graphs can be maintained in $O(m)$ time per update by performing one augmentation per update, or in $O(n)$ amortized time in the incremental setting [58, 72]. On planar graphs, it can be maintained in the fully dynamic setting in $\widetilde{O}(n^{2/3})$ update and query time [64] and in the incremental setting with $\widetilde{O}(\sqrt{n})$ update and query time [42]. Beyond that, there is more recent work on the approximate setting, with $(1-\epsilon)$-approximate incremental algorithms [53, 91], as discussed earlier. Finally, in the fully dynamic setting, algorithms with super-constant (i.e. polylog or subpolynomial) approximation ratios and sublinear amortized update time [35] and for uncapacitated graphs with subpolynomial worst-case update time [55] are known.

Dynamic bipartite matching (which is a special case of directed maxflow) has also received significant attention in the approximate setting [59, 57, 26, 17, 18, 9, 80, 86, 19, 5, 33, 14, 16, 24, 11, 34, 92, 15, 13, 20, 70, 12, 74, 56, 82, 6, 23, 21, 25]. In the $(1-\epsilon)$-approximate regime, the current state-of-the-art is $\text{poly}(1/\epsilon)$ update time for the incremental setting [25], $\text{poly}(\log(n)/\epsilon)$ for the decremental setting [22], and $O(\sqrt{m}^{1-\Omega_\epsilon(1)})$ for the fully dynamic setting [21]. For fully dynamic exact bipartite matching, the fastest update time is $O(n^{1.406})$ [83, 31].

Finally, it is worth mentioning that recent works have leveraged numerical optimization methods based on entropy-regularized optimal transport and MWU to design dynamic algorithms for partially dynamic bipartite matching and positive linear programs [65, 22].

Edge Connectivity. The k-edge connectivity between two vertices s, t can be seen as a maxflow of value up to k. Dynamic k-edge st-connectivity has been studied for small constant values of $k \leq 5$ [50, 51, 49, 93, 43, 44, 61, 48, 45, 89, 62, 63]. For super-constant k, [67] presents a fully dynamic algorithm with $n^{o(1)}$ update time for $k = (\log n)^{o(1)}$. [32] give an offline fully dynamic algorithm with $\widehat{O}(k^{O(k)})$ query time. These results all require small k to be efficient, whereas our result has no such restrictions. We also point out the work in [90] which gives an algorithm to dynamically maintain the (value of) the global min-cut with $\widetilde{O}(\sqrt{n})$ worst-case update time.

Dynamic Electric Flows. For $p = 2$, our incremental ℓ_p-norm flow algorithm can maintain a $(1-\epsilon)$-approximate electric flow between two fixed vertices $s, t \in V$ subject to edge insertions. Dynamic electric flows have previously been studied in [54, 47]. Such dynamic electric flow algorithms were also studied for the purpose of accelerating static maxflow and mincost flow algorithms [52, 28]. The closely related concept of dynamic effective resistances has also been studied in the online dynamic [35] and offline dynamic setting [76].

1.3 Paper Organization In the remainder of the paper, we provide preliminaries in section 2 and then give a more technical overview of our approach in section 3. We then we prove our main result, theorem 1.2, via iterative refinement in section 4. The incremental algorithm for the ℓ_p-norm residual problem is then presented in section 5. Finally, we argue that the approximate data structure [36] can be used to implement the incremental multiplicative weight method in section 6.

2 Preliminaries

In this section, we introduce notations we use throughout the paper.

General notation. We denote vectors by boldface lowercase letters. We use uppercase boldface to denote matrices. Often, we use uppercase matrices to denote the diagonal matrices corresponding to lowercase vectors, such as $\mathbf{L} = \text{diag}(\boldsymbol{\ell})$. For vectors $\boldsymbol{x}, \boldsymbol{y}$ we define the vector $\boldsymbol{x} \circ \boldsymbol{y}$ as the entrywise product, i.e., $(\boldsymbol{x} \circ \boldsymbol{y})_i = \boldsymbol{x}_i \boldsymbol{y}_i$. We also define the entrywise absolute value of a vector $|\boldsymbol{x}|$ as $|\boldsymbol{x}|_i = |\boldsymbol{x}_i|$. We use $\langle \cdot, \cdot \rangle$ as the vector inner product: $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{y} = \sum_i \boldsymbol{x}_i \boldsymbol{y}_i$. We elect to use this notation when $\boldsymbol{x}, \boldsymbol{y}$ have superscripts (such as time indices) to avoid cluttering. For positive real numbers a, b we write $a \approx_\alpha b$ for some $\alpha > 1$ if $\alpha^{-1} b \leq a \leq \alpha b$. For positive vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}_+^{[n]}$, we say $\boldsymbol{x} \approx_\alpha \boldsymbol{y}$ if $\boldsymbol{x}_i \approx_\alpha \boldsymbol{y}_i$ for all $i \in [n]$.

Graphs. In this paper, we consider multi-graphs G with edge set $E(G)$ and vertex set $V(G)$. When the graph is clear from context, we use the shorthands E for $E(G)$, V for $V(G)$, $m = |E|$, and $n = |V|$. We assume that each edge $e \in E$ has an implicit direction, used to define its edge-vertex incidence matrix $\mathbf{B} \in \mathbb{R}^{E \times V}$, i.e., $\mathbf{B}_{e,u} = -1$, $\mathbf{B}_{e,v} = +1$, and zero elsewhere for the row corresponding to the edge $e = (u, v)$. Abusing notation slightly, we often write $e = (u, v) \in E$ where e is an edge in E and u and v are the tail and head of e respectively (note that technically multi-graphs do not allow for edges to be specified by their endpoints).

A vector $\boldsymbol{d} \in \mathbb{R}^V$ is a demand vector if it is orthogonal to the all-ones vector, i.e., $\sum_{v \in V} \boldsymbol{d}_v = 0$. We say

a flow $\boldsymbol{f} \in \mathbb{R}^E$ routes a demand $\boldsymbol{d} \in \mathbb{R}^V$ if $\mathbf{B}^\top \boldsymbol{f} = \boldsymbol{d}$. We say a flow $\boldsymbol{f}$ is a circulation if it routes an all-zeros demand, i.e., each vertex has zero net flow. For an edge $e = (u, v) \in G$ we let $\boldsymbol{b}_e \in \mathbb{R}^V$ denote the demand vector of routing one unit from u to v.

Dynamic Algorithms. We say G is a *dynamic* graph, if it undergoes *batches* $U^{(1)}, U^{(2)}, \ldots$ of updates consisting of edge insertions/deletions that are applied to G. We use $|U^{(t)}|$ to denote the number of updates contained in the batch $U^{(t)}$. The results on dynamic graphs in this article often only consider a subset of the update types and we therefore often state for each dynamic graph which updates are allowed. We say that a dynamic graph G is incremental (and decremental) if it only undergoes edge insertions (and edge deletions respectively). Additionally, we say that the graph G, after applying the first t update batches $U^{(1)}, U^{(2)}, \ldots, U^{(t)}$, is at *stage* t and denote the graph at this stage by $G^{(t)}$. Additionally, when G is clear, we often denote the value of a variable x at the end of stage t of G by $x^{(t)}$, or a vector $\boldsymbol{x}$ at the end of stage t of G by $\boldsymbol{x}^{(t)}$.

3 Technical Overview

Here we provide a technical overview of the approach we take to obtain the results outlined in Section 1.1. In Section 3.1, we briefly review a variety of previous tools which we leverage and obstacles that we overcome to obtain our results. In Section 3.2, we then elaborate on our central insight about dynamic data structures for the minimum ratio cycle problem that fuels our results. In Section 3.3, we then discuss the dynamic optimization frameworks we use to leverage this data structure. Finally, in Section 3.4, we discuss how we put these pieces together to obtain our results.

3.1 Previous Tools and Obstacles.

We begin by describing the key tools of [36] and [91] which solved maximum flow in almost linear time and obtained an $\widehat{O}(\epsilon^{-1}\sqrt{n})$ time incremental algorithm for directed maximum flow. We elaborate on these tools and the obstacles for using them to achieve $n^{o(1)}\epsilon^{-O(1)}$ update time for undirected maximum flow.

Minimum Ratio Cycle. Both [36] and [91] are built upon efficient data structures for approximately solving the min-ratio cycle problem on a fully-dynamic graph, i.e., finding a cycle that approximately minimizes

$$\min_{\boldsymbol{c} \in \mathbb{R}^E \text{ is a cycle}} \frac{\langle \boldsymbol{g}, \boldsymbol{c} \rangle}{\|\mathbf{L}\boldsymbol{c}\|_1} \tag{3.2}$$

where $\boldsymbol{g} \in \mathbb{R}^E$ and $\boldsymbol{\ell} \in \mathbb{R}^E_+$ are called *edge gradients* and *lengths* respectively.

Both data structures are based on similar ideas. They maintain a d-level hierarchy of vertex and edge sparsification data structures. For vertex sparsification, they use *dynamic low stretch decompositions* which is studied in the context of dynamic shortest paths [35]. For edge sparsification, [36] proposed a data structure that maintains graph spanners, which are sparse graphs that preserves all-pairs distances, under edge updates as well as vertex splits. The original min-ratio cycle data structure has $m^{o(1)}$ update time and outputs $m^{o(1)}$-approximate solutions against oblivious adversaries. Additionally, the data structure succeeds against adaptive adversaries, as long as the inputs satisfy some additional "stability properties". We elaborate on this later in section 6. In [91], they make the data structure adaptive at a higher update time of $n^{1/2+o(1)}$.

Interior Point Methods. The static algorithm of [36] uses an IPM potential $\Phi(\boldsymbol{f})$ to find the maximum flow. It starts at some initial flow where $\Phi(\boldsymbol{f}) = \widetilde{O}(m)$ and iteratively makes progress until the potential is $\Phi(\boldsymbol{f}) < -\widetilde{O}(m)$. At that point, an optimal flow is obtained by standard rounding techniques. When the current flow is $\boldsymbol{f}$, the algorithm finds a $m^{o(1)}$-approximate cycle $\boldsymbol{c}$ to an instance of min-ratio cycle (3.2) with $\boldsymbol{g} \approx \nabla\Phi(\boldsymbol{f})$ and $\boldsymbol{\ell} \approx \sqrt{\nabla^2\Phi(\boldsymbol{f})}$. One can show that augmenting $\boldsymbol{f}$ with a multiple of $\boldsymbol{c}$ decreases the potential by at least $m^{-o(1)}$. After $m^{1+o(1)}$ iterations, the algorithm reaches a flow $\boldsymbol{f}$ whose potential value is at most $-\widetilde{O}(m)$.

The incremental maximum flow algorithm of [91] dynamizes the potential reduction procedure of [36]. To handle an edge insertions, [91] keeps augmenting the current flow with an $m^{o(1)}$-approximate min-ratio cycle until the output cycle cannot make enough, $m^{-o(1)}$, progress. The analysis of this leverages that adding an edge does not affect the feasibility of the current flow and only increases the potential by a constant amount. Additionally, once a flow of cost at most the given threshold appears in the graph, any $m^{o(1)}$-approximate min-ratio cycle decreases the potential by at least $m^{-o(1)}$ as long as our current flow has its cost larger than the threshold by $\exp(-\widetilde{O}(1))$. Since the potential value starts at $\widetilde{O}(m)$, over the course of the incremental algorithm, there are at

most $m^{1+o(1)}$ approximate min-ratio cycle queries. This yields a $n^{1/2+o(1)}$-update time using their adaptive data structure for answering min-ratio cycle queries.

Obstacles. In the static case, [36] manages to apply an oblivious data structure for iteratively minimizing the IPM potential due to (a) the existance an optimal flow before initializing the data structure and (b) the stability of the gradient and Hessian of the IPM potential (which are the inputs to the min-ratio cycle data structure). Consequently, whenever the data structure fails to find a cycle good enough to make progress, it must be the case that some part of the data structure is broken, and we need to fix it actively. However, (a) does not hold in the incremental case. That is, whenever the oblivious data structure fails, we cannot distinguish between the case that (1) the data structure fails due to the obliviousness, or (2) the graph does not support the optimal flow because every feasible flow has a large cost. The issue occurs even when all incremental updates come from an oblivious adversary.

Our Approach. To obtain our results we depart from prior work in both how we reason about adaptive adversaries in the minimum ratio cycle problem and in what optimization method we use for this dynamic data structure. We elaborate on each of these in the next section 3.2 and section 3.3 and then discuss how they are put together to obtain our results in section 3.4.

3.2 Adaptive Adversaries and Monotonic Dynamic Min-Ratio Cycle.

We manage to use the oblivious min-ratio cycle data structure by ensuring that edge lengths are *mostly increasing*. That is, with a different numerical method, we can divide the entire incremental algorithm into $m^{o(1)}$ phases and within each phase, we solve a sequence of slowly changing min-ratio cycle problems with the same gradient $\boldsymbol{g}$ and monotonically increasing lengths $\boldsymbol{\ell}$. In section 6 which presents the min-ratio cycle data structure, we work with an adversary where we can assume that all incremental updates are determined before the initialization. That is, the adversary does not have access to the internal state of our algorithm when deciding which edges to insert next. Using these facts, we show, in section 6, that the updates to the dynamic min-ratio cycle data structure satisfy a weaker form of the *hidden stable-flow chasing* property, which was the critical property that [36] leveraged to show correctness of their min-ratio cycle data structure. While [36] had to periodically rebuild layers of their data structure when the "lengths" of flows at those layers may have decreased, and this prevented its application to incremental directed maxflow in [91], our monotonicity property allows us to avoid this issue. In particular, whenever the data structure fails to output a good cycle, we know for certain that every feasible flow has large congestion.

3.3 From IPMs to Iterative Refinement and MWU

This paper focuses on solving *incremental thresholded ℓ_p-norm flow* for $p \geq 2$. This immediately gives an incremental $(1+\epsilon)$-approximate maximum flow due to the choice of p. Unlike IPM-based maxflow algorithms, ℓ_p-norm flows can be reduced to approximately solving $m^{o(1)}$ smoothed ℓ_p-norm flow *residual problems* to $m^{o(1)}$-approximation factors. To approximately solve each residual problem, we use a multiplicative weight update method (MWU) to reduce the problem to a sequence of $m^{1+o(1)}$ slowly changing ℓ_1-regression sub-problems, which are equivalent to min-ratio cycles in our case. The nature of MWU ensures that the ℓ_1 weights are non-decreasing. These constraints on how the sequence of min-ratio cycle instances change enable us to use the data structure of [36] to achieve the almost-linear runtime.

The reduction to residual problem is achieved via the iterative refinement framework of [2]. Each residual problem asks to find a circulation $\boldsymbol{\Delta}$ such that

$$\|\mathbf{R}\boldsymbol{\Delta}\|_2 \leq m^{o(1)},\ \|\mathbf{W}\boldsymbol{\Delta}\|_p \leq m^{o(1)}, \text{ and } \langle \boldsymbol{g}, \boldsymbol{\Delta} \rangle = -1 \tag{3.3}$$

where $\boldsymbol{r}, \boldsymbol{w} \in \mathbb{R}^E_+$ are edge weights derived from the current solution to problem 1.2, and we are guaranteed the existence of a circulation whose both norms are at most 1.

To find a feasible residual solution to Problem eq. (3.3), we use a MWU to reduce the problem to a sequence of $m^{1+o(1)}$ ℓ_1 sub-problems of finding a circulation $\boldsymbol{c}$ such that

$$\|\mathbf{L}\boldsymbol{c}\|_1 \leq m^{o(1)} \|\boldsymbol{\ell}\|_1 \text{ and } \langle \boldsymbol{g}, \boldsymbol{c} \rangle = -1\,. \tag{3.4}$$

This is equivalent to finding a min-ratio cycle (3.2) up to scaling the solution so that $\langle \boldsymbol{g}, \boldsymbol{c} \rangle = -1$. If we can solve each ℓ_1 sub-problem to $m^{o(1)}$-approximation, we obtain a feasible residual solution. Furthermore, MWU ensures that the ℓ_1 weights are non-decreasing across all $m^{1+o(1)}$ instances.

3.4 Putting it Altogether To dynamize the approach mentioned in section 3.3, we make the following observation:

> If $OPT \leq F$, there is a feasible solution $\boldsymbol{\Delta}^*$ to (3.3) with $\|\mathbf{R}\boldsymbol{\Delta}\|_2, \|\mathbf{W}\boldsymbol{\Delta}\|_p \leq 1$ that satisfies (3.4), i.e., $\|\mathbf{L}\boldsymbol{\Delta}^*\|_1 \leq \|\boldsymbol{\ell}\|_1$ for any $\boldsymbol{\ell}$ encountered in the algorithm.

We leverage that $\boldsymbol{\Delta}^*$ is fixed with respect to $\boldsymbol{r}, \boldsymbol{w}$ and $\boldsymbol{g}$. Thus, whenever the $m^{o(1)}$-approximate min-ratio cycle data structure cannot find a solution to (3.4), we certify that $OPT > F$. Otherwise, if $OPT \leq F$, the MWU method, as well as the iterative refinement procedure, would proceed as desired and output a flow $\boldsymbol{f}$ of ℓ_p-norm energy at most $\mathcal{E}(\boldsymbol{f}) \leq F + \epsilon$ in $p^2 m^{1+o(1)} \log(1/\epsilon)$-time.

4 Incremental p-Norm Iterative Refinement

In this section, we discuss the iterative refinement approach to solving p-norm flows and how reduce them to incremental MWUs. The iterative refinement framework of [2] reduces p-norm flows to approximately solving a small number of residual problems. The original framework requires an estimate on the optimal residual value, which is obtained via binary search. In our setting, a target objective value is given and we can directly use it to estimate the residual value. This requires a slight modification to the convergence analysis via measuring the progress towards the target value.

The goal of this section is to prove the following theorem:

THEOREM 4.1. *There is a randomized algorithm for problem 1.3 that given an initial flow $\boldsymbol{f}^{(0)}$ and inputs $\epsilon, \boldsymbol{g}^G, \boldsymbol{r}^G, \boldsymbol{w}^G$ and $\boldsymbol{d}$ with sizes bounded by $\widetilde{O}(1)$ in fixed point arithmetic., runs in $p^2 m^{1+o(1)} \log(\frac{\mathcal{E}(\boldsymbol{f}^{(0)})-F}{\epsilon})$ time and succeeds whp. against adaptive adversaries.*

We first define the residual problem, $\mathcal{R}_{\boldsymbol{f}}(\boldsymbol{x})$, that we consider to approximate $\mathcal{E}(\boldsymbol{f}+\boldsymbol{x}) - \mathcal{E}(\boldsymbol{f})$ (Problem 4.1) and provide a known lemma about its approximation quality (Lemma 4.2).

PROBLEM 4.1. (RESIDUAL PROBLEM) *For feasible flow $\boldsymbol{f}$ in p-Norm flow instance $(G, \boldsymbol{g}^G, \boldsymbol{r}^G, \boldsymbol{w}^G, \boldsymbol{d})$, we define its* residual problem, *$\mathcal{R}_{\boldsymbol{f}}(\boldsymbol{x})$, (that approximates) as follows:*

$$\mathcal{R}_{\boldsymbol{f}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \langle \boldsymbol{g}, \boldsymbol{x} \rangle + \|\mathbf{R}\boldsymbol{x}\|_2^2 + \|\mathbf{W}\boldsymbol{x}\|_p^p,$$

where $\boldsymbol{g} \stackrel{\text{def}}{=} \boldsymbol{g}^G + 2(\mathbf{R}^G)^2\boldsymbol{f} + p(\mathbf{W}^G)^p|\boldsymbol{f}|^{p-2}\boldsymbol{f}, \boldsymbol{r} \stackrel{\text{def}}{=} \sqrt{(\boldsymbol{r}^G)^2 + 2p^2(\mathbf{W}^G)^p|\boldsymbol{f}|^{p-2}}, \boldsymbol{w} \stackrel{\text{def}}{=} p\boldsymbol{w}^G$.

We often ignore the subscript $\boldsymbol{f}$ when it is clear from the context.

LEMMA 4.2. (ITERATIVE REFINEMENT, [2, 3, 4]) *For any $\boldsymbol{f}$ and $\boldsymbol{x}$, we have*

$$\mathcal{E}(\boldsymbol{f}+\boldsymbol{x}) - \mathcal{E}(\boldsymbol{f}) \leq \mathcal{R}_f(\boldsymbol{x}), \text{ and}$$
$$\mathcal{E}(\boldsymbol{f}+\lambda\boldsymbol{x}) - \mathcal{E}(\boldsymbol{f}) \geq \lambda\mathcal{R}_f(\boldsymbol{x}), \text{ for some } \lambda = O(p)$$

Using lemma 4.2, we can relate the optimal residual objective value to the gap between the current feasible flow $\boldsymbol{f}$ and the target threshold F.

LEMMA 4.3. (THRESHOLD CERTIFICATION BY RESIDUAL VALUE) *If feasible flow $\boldsymbol{f}$ in a p-Norm flow instance $(G, \boldsymbol{g}^G, \boldsymbol{r}^G, \boldsymbol{w}^G, \boldsymbol{d})$ satisfies $OPT \leq F$, then there is a circulation $\boldsymbol{c}^*$ with $\mathcal{R}(\boldsymbol{c}^*) \leq (F - \mathcal{E}(\boldsymbol{f}))/\lambda$.*

Proof. Let $\boldsymbol{f}^*$ be any feasible flow such that $\mathcal{E}(\boldsymbol{f}^*) \leq F$. lemma 4.2 yields that

$$F - \mathcal{E}(\boldsymbol{f}) \geq \mathcal{E}(\boldsymbol{f}^*) - \mathcal{E}(\boldsymbol{f}) \geq \lambda\mathcal{R}\left(\frac{\boldsymbol{f}^* - \boldsymbol{f}}{\lambda}\right).$$

The conclusion follows because $\boldsymbol{f}^* - \boldsymbol{f}$ is a circulation. ∎

Let $R \stackrel{\text{def}}{=} (\mathcal{E}(\boldsymbol{f}) - F)/\lambda > 0$ be the residual threshold. Our goal now is to find a circulation $\boldsymbol{c}$ such that $\mathcal{R}(\boldsymbol{c}) \leq -R/K$ for some $K = m^{o(1)}$. We show that we can compute this by solving the following *incremental residual problem*.

PROBLEM 4.4. (INCREMENTAL K-APPROXIMATE RESIDUAL PROBLEM) *Consider an incremental graph $G = (V, E)$ with at most m edges, a gradient vector $\boldsymbol{g} \in \mathbb{R}^E$, ℓ_2 and ℓ_p edge weights $\boldsymbol{r}, \boldsymbol{w} \in \mathbb{R}^E_+$ and a approximation factor $K > 0$. The* Incremental K-Approximate Residual Problem *asks, after the initialization or each edge insertion, to either*

1. *certify that there is no circulation $\boldsymbol{c}^*$ with $\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle = -1$, $\|\mathbf{R}\boldsymbol{c}^*\|_2 \leq 1$, and $\|\mathbf{W}\boldsymbol{c}^*\|_p \leq 1$, or*
2. *output a circulation $\boldsymbol{c}$ such that $\langle \boldsymbol{g}, \boldsymbol{c} \rangle = -1$, $\|\mathbf{R}\boldsymbol{c}\|_2 \leq K$, and $\|\mathbf{W}\boldsymbol{c}\|_p \leq K$.*

In section 5, we will present an almost-linear time algorithm for problem 4.4.

LEMMA 4.5. *For some $K = m^{o(1)}$, there is a randomized algorithm for problem 4.4, denoted as $\mathcal{A}^{(4.5)}$, that runs in $m^{1+o(1)}$-time and succeeds with high probability against an adaptive adversary.*

Here, recall that the adversary is only adaptive against the yes/no output of the algorithm, and cannot see the internal randomness, including the flow that is being stored internally. Obviously, our algorithm can output a flow when we get a "no" output, by computing a high-accuracy p-norm minimizing flow, but this is different from the flow being stored internally.

At first glance, the solution guarantee of problem 4.4 has nothing to do with the residual problem (problem 4.1) and R. However, since we only need a solution whose objective is better than $m^{-o(1)}(F - \mathcal{E}(\boldsymbol{f}))/\lambda$, we can reduce incremental p-norm regression to the form of problem 4.4 using the following pair of lemmas. The first is a straightforward scaling argument.

LEMMA 4.6. *If there's some $\boldsymbol{c}^*$ such that $\mathcal{R}(\boldsymbol{c}^*) \leq -R$ for some threshold $R > 0$ then,*

$$\frac{\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle}{\|\mathbf{R}\boldsymbol{c}^*\|_2} \leq -2\sqrt{R} \text{ and } \frac{\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle}{\|\mathbf{W}\boldsymbol{c}^*\|_p} \leq -R^{(p-1)/p}$$

Proof. From the bound that $\mathcal{R}(\boldsymbol{c}^*) \leq -R$, we know

$$\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle + \|\mathbf{R}\boldsymbol{c}^*\|_2^2 \leq -R$$

Thus we know that

$$\frac{\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle}{\|\mathbf{R}\boldsymbol{c}^*\|_2} \leq -\frac{R}{\|\mathbf{R}\boldsymbol{c}^*\|_2} - \|\mathbf{R}\boldsymbol{c}^*\|_2 \leq -2\sqrt{R},$$

by the AM-GM inequality. Similarly,

$$\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle + \|\mathbf{W}\boldsymbol{c}^*\|_p^p \leq -R,$$

and hence

$$\frac{\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle}{\|\mathbf{W}\boldsymbol{c}^*\|_p} \leq -\frac{R}{\|\mathbf{W}\boldsymbol{c}^*\|_p} - \|\mathbf{W}\boldsymbol{c}^*\|_p^{p-1} \leq -R^{\frac{p-1}{p}},$$

where we used the (weighted) AM-GM inequality again. ∎

The second, given below, is used to argue that the solution quality of problem 4.4 translates to a bound on the residual objective value.

LEMMA 4.7. *Given a residual threshold $R > 0$ and the current residual problem $\mathcal{R}(\boldsymbol{c})$ with gradient $\boldsymbol{g}$ and ℓ_2 and ℓ_p edge weights $\boldsymbol{r}$ and $\boldsymbol{w}$, consider a circulation $\boldsymbol{c}$ such that $\langle \boldsymbol{g}, \boldsymbol{c} \rangle = -1$, $\|2\sqrt{R}\mathbf{R}\boldsymbol{c}\|_2 \leq K$, and $\|R^{(p-1)/p}\mathbf{W}\boldsymbol{c}\|_p \leq K$ for some $K \geq 1$. We have $\mathcal{R}((R/(2K^2))\boldsymbol{c}) \leq -R/(6K^2)$.*

Proof. From the assumptions of the lemma, we know that

$$\langle \boldsymbol{g}, \boldsymbol{c} \rangle = -1, \ \|\mathbf{R}\boldsymbol{c}\|_2 \leq \frac{K}{2\sqrt{R}}, \text{ and } \ \|\mathbf{W}\boldsymbol{c}\|_p \leq KR^{-(p-1)/p}$$

Plugging these into the definition of the residual problem yields

$$\begin{aligned}\mathcal{R}\left(\frac{R}{2K^2}\boldsymbol{c}\right) &= \frac{R}{2K^2}\langle \boldsymbol{g}, \boldsymbol{c}\rangle + \frac{R^2}{4K^4}\left\|\mathbf{R}\boldsymbol{c}\right\|_2^2 + \frac{R^p}{2^p K^{2p}}\left\|\mathbf{W}\boldsymbol{c}\right\|_p^p \\ &\leq -\frac{R}{2K^2} + \frac{R^2}{4K^4}\frac{K^2}{4R} + \frac{R^p}{2^p K^{2p}} K^p R^{-(p-1)} \\ &= -\frac{R}{2K^2} + \frac{R}{16K^2} + \frac{R}{2^p K^p} \leq \frac{-R}{5K^2}\,.\end{aligned}$$

□

Now, we are ready to state algorithm 1 which we use to prove theorem 1.2.

Algorithm 1: Incremental Thresholded p-Norm Flow

global variables
- p: the norm considered in problem 1.3.
- $\lambda \leftarrow O(p)$: the scaling factor in lemma 4.2.
- $K \leftarrow m^{o(1)}$: the approximation factor in lemma 4.5.

procedure IncrementalPNorm($G, \boldsymbol{d}, \boldsymbol{g}^G, \boldsymbol{r}^G, \boldsymbol{w}^G, F, \epsilon$)
- Initialize $\boldsymbol{f}^{(0)}$ to be the optimal p-norm flow on G.
- $S \stackrel{\text{def}}{=} O(pK^2 \log(\frac{\mathcal{E}(\boldsymbol{f}^{(0)})-F}{\epsilon}))$
- **for** *step* $t = 0, 1, \ldots, S$ **do**
  - Construct residual problem $(\boldsymbol{g}^{(t)}, \boldsymbol{r}^{(t)}, \boldsymbol{w}^{(t)})$ with respect to $\boldsymbol{f}^{(t)}$.
  - Define residual threshold $R \stackrel{\text{def}}{=} (\mathcal{E}(\boldsymbol{f}^{(t)}) - F)/\lambda$.
  - Initialize and run the incremental approximate residual algorithm
  - **while** $\mathcal{A}^{(4.5)}$ *cannot find a circulation satisfying Lemma 4.5 item 2* **do**
    - Claim that $OPT > F$.
    - Wait for a new edge e.
    - Insert the new edge e to $\mathcal{A}^{(4.5)}$ with

$$\boldsymbol{g}_e^{(t)} \stackrel{\text{def}}{=} \boldsymbol{g}_e^G,\ \boldsymbol{r}_e^{(t)} \stackrel{\text{def}}{=} 2\sqrt{R}\boldsymbol{r}_e^G, \text{ and } \boldsymbol{w}_e^{(t)} \stackrel{\text{def}}{=} R^{(p-1)/p}\boldsymbol{w}_e^G$$

  - $\mathcal{A}^{(4.5)}$ finds a circulation $\boldsymbol{c}$ such that

$$\langle \boldsymbol{g}^{(t)}\ , \boldsymbol{c}\rangle = -1, \|2\sqrt{R}\mathbf{R}^{(t)}\boldsymbol{c}\|_2 \leq K, \text{ and } \|R^{(p-1)/p}\mathbf{W}^{(t)}\boldsymbol{c}\|_p \leq K\,.$$

  - Set $\boldsymbol{f}^{(t+1)} \leftarrow \boldsymbol{f}^{(t)} + \frac{R}{2K^2}\boldsymbol{c}$
- **return** $\boldsymbol{f}^{(S)}$

To show correctness, we first prove that each step makes an exponential progress. Leveraging this lemma we then prove theorem 1.2.

Lemma 4.8. (Iterative Refinement Convergence Rate) *At the end of any step t in algorithm 1,*

$$\mathcal{E}(\boldsymbol{f}^{(t+1)}) - F \leq \left(1 - \frac{1}{6K^2\lambda}\right)\left(\mathcal{E}(\boldsymbol{f}^{(t)}) - F\right)$$

Proof. At the end of step t, lemma 4.7 ensures that $\mathcal{R}(\frac{R}{2K^2}\boldsymbol{c}) \leq -\frac{R}{6K^2}$. lemma 4.2 then yields

$$\mathcal{E}(\boldsymbol{f}^{(t+1)}) - \mathcal{E}(\boldsymbol{f}^{(t)}) \leq \mathcal{R}\left(\frac{R}{2K^2}\boldsymbol{c}\right) \leq \frac{-1}{6K^2\lambda}\left(\mathcal{E}(\boldsymbol{f}^{(t)}) - F\right)$$

Thus, we have

$$\begin{aligned}\mathcal{E}(\boldsymbol{f}^{(t+1)}) - F &= \mathcal{E}(\boldsymbol{f}^{(t+1)}) - \mathcal{E}(\boldsymbol{f}^{(t)}) + \mathcal{E}(\boldsymbol{f}^{(t)}) - F \\ &\leq \left(1 - \frac{1}{6K^2\lambda}\right)\left(\mathcal{E}(\boldsymbol{f}^{(t)}) - F\right)\end{aligned}$$

□

Proof. [Proof of theorem 1.2] We first show that whenever the algorithm outputs that $OPT > F$, this is indeed the case. Let t be an arbitrary step in which the algorithm outputs that $OPT > F$, let $\boldsymbol{f}^{(t)}$ be the flow maintained at that step, and let $\mathcal{R}(\boldsymbol{x})$ be the corresponding residual problem instance (problem 4.1). Correctness of lemma 4.5 ensures that no circulation $\boldsymbol{c}^*$ satisfies

$$\langle \boldsymbol{g}^{(t)}, \boldsymbol{c}^* \rangle = -1, \|2\sqrt{R}\mathbf{R}^{(t)}\boldsymbol{c}^*\|_2 \leq 1, \|R^{(p-1)/p}\mathbf{W}^{(t)}\boldsymbol{c}^*\|_p \leq 1$$

This fact and lemma 4.6 then implies that every circulation $\boldsymbol{c}^*$ must have $\mathcal{R}(\boldsymbol{c}^*) > -R = (F - \mathcal{E}(\boldsymbol{f}^{(t)}))/\lambda$. lemma 4.3 implies that $OPT > F$.

Now, we analyze the quality of the output, $\boldsymbol{f}^{(S)}$. Applying lemma 4.8 inductively on steps t yields that

$$\mathcal{E}(\boldsymbol{f}^{(S)}) - F \leq \left(1 - \frac{1}{6K^2\lambda}\right)^S \left(\mathcal{E}(\boldsymbol{f}^{(0)}) - F\right) \leq \epsilon$$

by definitions of S and λ (lemma 4.2).

Finally, the runtime comes directly from $S = m^{o(1)}p\log(\frac{1}{\epsilon})$ applications of lemma 4.5. (Recall that there is an extra factor of p due to bit complexity.) □

5 Incremental Multiplicative Weight Updates

In this section, we present a MWU-based algorithm that solves problem 4.4 and proves lemma 4.5. We use algorithm 2 to prove lemma 4.5. At a high level, the algorithm outputs the final circulation as an average of $T = \widetilde{O}(m)$ circulations. At each iteration i, we compute a set of ℓ_1 edge weights $\boldsymbol{\ell} \in \mathbb{R}^E_+$ and compute an approximate min ratio cycle that minimizes $\langle \boldsymbol{g}, \boldsymbol{c}\rangle / \|\mathbf{L}\boldsymbol{c}\|_1$. As is too costly to compute each cycle from scratc, we instead apply a dynamic data structure simplified from [36] to compute such cycle efficiently.

Problem 5.1. (Approximate Monotonic Min Ratio Cycle (MRC)) *Consider a target ratio $\alpha > 0$ and a dynamic graph $G = (V, E)$ with edge gradients $\boldsymbol{g} \in \mathbb{R}^E$, and lengths $\boldsymbol{\ell} \in \mathbb{R}^E_+$ under edge insertions and length increases. In addition, there is a hidden circulation $\boldsymbol{c}^*$ not necessarily supported on G, i.e., there is possibly some edge not in G with $\boldsymbol{c}^*_e \neq 0$. The problem of* κ-approximate monotonic min ratio cycle *asks, after the initialization and each edge update,*

1. *If $\boldsymbol{c}^*$ is not supported on G: output any circulation.*
2. *If $\boldsymbol{c}^*$ is supported on G: output a circulation $\boldsymbol{c}$ such that*

$$\frac{\langle \boldsymbol{g}, \boldsymbol{c}\rangle}{\|\mathbf{L}\boldsymbol{c}\|_1} \leq \frac{-\alpha}{\kappa},$$

given that $\frac{\langle \boldsymbol{g}, \boldsymbol{c}^*\rangle}{\|\mathbf{L}\boldsymbol{c}^*\|_1} \leq -\alpha$.

We can solve problem 5.1 in almost-linear time by using a data structure from [36]. The guarantee of this data structure is stated below and the lemma is proved in Section 6.

Lemma 5.2. (Approximate Monotonic MRC Data Structure) *For some $\kappa = m^{o(1)}$, there is a randomized algorithm, denoted $\mathcal{A}^{(5.2)}$, that maintains a collection of $s = m^{o(1)}$ spanning trees $T_1, T_2, \ldots, T_s$ and solves the κ-approximate monotonic MRC problem (problem 5.1) in $(m+Q)m^{o(1)}$-time where Q is the total number of updates. In particular, it always output a cycle $\boldsymbol{\Delta}$ represented by $m^{o(1)}$ off-tree edges and paths on a spanning tree T_i for some $i \in [s]$.*

We now state algorithm 2 which shows lemma 4.5. The algorithm runs a MWU algorithm with ℓ_1-norm subproblems for $\widetilde{O}(m)$ iterations. This is motivated by the ℓ_1-IPM of [36] for solving min-cost flow, and contrasts with ℓ_2-norm MWUs of [37, 2] for approximate maxflow and p-norm regression. These ℓ_1-norm subproblems are min-ratio cycle problems, which we solve by calling a min-ratio cycle data structure. If the output has sufficiently good ratio, the algorithm proceeds to the next iteration. Otherwise, we conclude that G does not support a valid circulation yet, and ask for the next edge insertion.

Algorithm 2: Incremental K-Approximate Residual Algorithm

global variables
- p: the norm considered in problem 4.4.
- $q \leftarrow \min\{\log_2 m, p\}$: the norm which the algorithm handles.
- $\kappa \leftarrow m^{o(1)}$: the approximation factor in lemma 5.2.
- $K \leftarrow 100q\kappa$: the approximation factor of this algorithm.
- $\alpha \leftarrow K^{1-q}/(10q)$: the target ratio given in lemma 5.7.

procedure IncrementalMWU($G, \boldsymbol{g}, \boldsymbol{r}, \boldsymbol{w}$)
- Initialize $\boldsymbol{c}^{(0)} = \mathbf{0}$, $\boldsymbol{a}^{(0)} = Km^{-1/2}\boldsymbol{r}^{-1}$, and $\boldsymbol{b}^{(0)} = Km^{-1/q}\boldsymbol{w}^{-1}$.
- Initialize edge lengths $\boldsymbol{\ell}^{(0)} \stackrel{\text{def}}{=} K^{q-2}\boldsymbol{r}^2\boldsymbol{a}^{(0)} + \boldsymbol{w}^q(\boldsymbol{b}^{(0)})^{q-1}$.
- Initialize a κ-approximate MRC algorithm $\mathcal{A}^{(5.2)}$ on $(G, \boldsymbol{g}, \boldsymbol{\ell})$ and target ratio α.
- Set $T \leftarrow 100qm$
- **for** *iteration* $i = 0, 1, \ldots, T$ **do**
  - Define the edge length $\boldsymbol{\ell}^{(i)} \stackrel{\text{def}}{=} K^{q-2}\boldsymbol{r}^2\boldsymbol{a}^{(i)} + \boldsymbol{w}^q(\boldsymbol{b}^{(i)})^{q-1}$.
  - Maintain an estimation $\widetilde{\boldsymbol{\ell}} \in [\boldsymbol{\ell}^{(i)}, 2\boldsymbol{\ell}^{(i)}]$.
  - Feed updates to the estimation $\widetilde{\boldsymbol{\ell}}$ to $\mathcal{A}^{(5.2)}$.
  - **while** $\mathcal{A}^{(5.2)}$ *outputs a cycle of ratio* $> -\alpha/\kappa$ **do**
    - Claim that there's no circulation $\boldsymbol{c}^*$ s.t.
    $$\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle = -1, \ \|\mathbf{R}\boldsymbol{c}^*\|_2 \le 1, \text{ and } \|\mathbf{W}\boldsymbol{c}^*\|_p \le 1$$
    - Wait for a new edge e.
    - Set $\boldsymbol{a}_e^{(i)} = Km^{-1/2}\boldsymbol{r}_e^{-1}$ and $\boldsymbol{b}_e^{(i)} = Km^{-1/q}\boldsymbol{w}_e^{-1}$.
    - Insert the new edge e to $\mathcal{A}^{(5.2)}$ with gradient $\boldsymbol{g}_e$ and length $\widetilde{\boldsymbol{\ell}}_e = \boldsymbol{\ell}_e^{(i)} = K^{q-2}\boldsymbol{r}_e^2\boldsymbol{a}_e^{(i)} + \boldsymbol{w}_e^q(\boldsymbol{b}_e^{(i)})^{q-1}$.
  - $\mathcal{A}^{(5.2)}$ outputs a cycle $\boldsymbol{\Delta}^{(i)}$ s.t. $\langle \boldsymbol{g}, \boldsymbol{\Delta}^{(i)} \rangle / \|\widetilde{\mathbf{L}}\boldsymbol{\Delta}^{(i)}\|_1 \le -\alpha/\kappa$.
  - Scale $\boldsymbol{\Delta}^{(i)}$ such that $\langle \boldsymbol{g}, \boldsymbol{\Delta}^{(i)} \rangle = -1$ and $\|\widetilde{\mathbf{L}}\boldsymbol{\Delta}^{(i)}\|_1 \le \kappa/\alpha \le K^q$.
  - Update $\boldsymbol{c}^{(i+1)} \stackrel{\text{def}}{=} \boldsymbol{c}^{(i)} + \frac{1}{T}\boldsymbol{\Delta}^{(i)}$, $\boldsymbol{a}^{(i+1)} \stackrel{\text{def}}{=} \boldsymbol{a}^{(i)} + \frac{1}{T}|\boldsymbol{\Delta}^{(i)}|$, and $\boldsymbol{b}^{(i+1)} \stackrel{\text{def}}{=} \boldsymbol{b}^{(i)} + \frac{1}{T}|\boldsymbol{\Delta}^{(i)}|$
- **return** $\boldsymbol{c}^{(T)}$

To analyze algorithm 2, we keep track of the following two potentials:

$$\Phi^{(i)} \stackrel{\text{def}}{=} \left\|\mathbf{R}\boldsymbol{a}^{(i)}\right\|_2^2 \tag{5.5}$$

$$\Psi^{(i)} \stackrel{\text{def}}{=} \left\|\mathbf{W}\boldsymbol{b}^{(i)}\right\|_q^q \tag{5.6}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are defined as

$$\boldsymbol{a}^{(i)} \stackrel{\text{def}}{=} \frac{K}{\sqrt{m}}\boldsymbol{r}^{-1} + \frac{1}{T}\sum_{j=0}^{i-1}|\boldsymbol{\Delta}^{(j)}| \tag{5.7}$$

$$\boldsymbol{b}^{(i)} \stackrel{\text{def}}{=} \frac{K}{m^{1/q}}\boldsymbol{w}^{-1} + \frac{1}{T}\sum_{j=0}^{i-1}|\boldsymbol{\Delta}^{(j)}| \tag{5.8}$$

Because of the definition of $\boldsymbol{a}^{(i)}$ and $\boldsymbol{b}^{(i)}$, we know $\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} \geq |\boldsymbol{c}^{(i)}|$ after any iteration i and thus both $\Phi^{(i)} \leq \|\mathbf{R}\boldsymbol{c}^{(i)}\|_2^2$ and $\Psi^{(i)} \leq \|\mathbf{W}\boldsymbol{c}^{(i)}\|_q^q$.

LEMMA 5.3. (INITIAL POTENTIAL) *Initially,* $\Phi^{(0)} = K^2$ *and* $\Psi^{(0)} = K^q$.

LEMMA 5.4. (INCREASE IN Φ) *At any iteration* i*, we have* $\Phi^{(i+1)} \leq \Phi^{(i)} + 3K^2/T$.

Proof. For simplicity in the presentation, we ignore superscripts (i) in the proof. Let $\boldsymbol{\Delta}$ be the cycle output in line 21 such that $\|\widetilde{\mathbf{L}}\boldsymbol{\Delta}\|_1 \leq K^q$ and Φ' be the new potential values after updating $\boldsymbol{a}' \stackrel{\text{def}}{=} \boldsymbol{a} + \frac{1}{T}|\boldsymbol{\Delta}|$. For any edge e, we know

$$K^{q-2}\boldsymbol{r}_e^2\boldsymbol{a}_e|\boldsymbol{\Delta}_e| \leq \boldsymbol{\ell}_e|\boldsymbol{\Delta}_e| \leq \|\mathbf{L}\boldsymbol{\Delta}\|_1 \leq \|\widetilde{\mathbf{L}}\boldsymbol{\Delta}\|_1 \leq K^q$$

Rearrangement and the monotonicity of $\boldsymbol{a}$ yields

$$\frac{K^2}{m}\frac{|\boldsymbol{\Delta}_e|}{\boldsymbol{a}_e} \leq \boldsymbol{r}_e^2\boldsymbol{a}_e^2\frac{|\boldsymbol{\Delta}_e|}{\boldsymbol{a}_e} = \boldsymbol{r}_e^2\boldsymbol{a}_e|\boldsymbol{\Delta}_e| \leq K^2$$

and therefore $|\boldsymbol{\Delta}_e| \leq m\boldsymbol{a}_e \leq T\boldsymbol{a}_e/3$. Applying the definition of Φ' yields

$$\begin{aligned}\Phi' &= \sum_e \boldsymbol{r}_e^2(\boldsymbol{a}_e + \frac{1}{T}|\boldsymbol{\Delta}_e|)^2 \\ &\leq \sum_e \boldsymbol{r}_e^2(\boldsymbol{a}_e^2 + \frac{3}{T}\boldsymbol{a}_e|\boldsymbol{\Delta}_e|) = \Phi + \frac{3}{T}\sum_e \boldsymbol{r}_e^2\boldsymbol{a}_e|\boldsymbol{\Delta}_e| \\ &\leq \Phi + \frac{3}{T}K^{2-q}\|\mathbf{L}\boldsymbol{\Delta}\|_1 \leq \Phi + \frac{3K^2}{T}\end{aligned}$$

where we use $(1+x)^2 \leq 1+3x$ for $x \in [0,1]$. ∎

LEMMA 5.5. (INCREASE IN Ψ) *At any iteration* i*, we have* $\Psi^{(i+1)} \leq \Psi^{(i)} + 4qK^q/T$.

Proof. For simplicity in the presentation, we ignore superscripts (i) in the proof. Let $\boldsymbol{\Delta}$ be the cycle output in line 21 such that $\|\widetilde{\mathbf{L}}\boldsymbol{\Delta}\|_1 \leq K^q$ and Ψ' be the new potential values after updating $\boldsymbol{b}' \stackrel{\text{def}}{=} \boldsymbol{b} + \frac{1}{T}|\boldsymbol{\Delta}|$. For any edge e, we know

$$\boldsymbol{w}_e^q\boldsymbol{b}_e^{q-1}|\boldsymbol{\Delta}_e| \leq \boldsymbol{\ell}_e|\boldsymbol{\Delta}_e| \leq \|\mathbf{L}\boldsymbol{\Delta}\|_1 \leq \|\widetilde{\mathbf{L}}\boldsymbol{\Delta}\|_1 \leq K^q$$

Rearrangement and the monotonicity of $\boldsymbol{b}$ yields

$$\frac{K^q}{m}\frac{|\boldsymbol{\Delta}_e|}{\boldsymbol{b}_e} \leq \boldsymbol{w}_e^q\boldsymbol{b}_e^q\frac{|\boldsymbol{\Delta}_e|}{\boldsymbol{b}_e} = \boldsymbol{w}_e^q\boldsymbol{b}_e^{q-1}|\boldsymbol{\Delta}_e| \leq K^q$$

and therefore $|\boldsymbol{\Delta}_e| \leq m\boldsymbol{b}_e \leq \frac{T\boldsymbol{b}_e}{100q}$. Applying the definition of Ψ' yields

$$\begin{aligned}\Psi' &= \sum_e \boldsymbol{w}_e^q(\boldsymbol{b}_e + \frac{1}{T}|\boldsymbol{\Delta}_e|)^q \\ &\leq \sum_e \boldsymbol{w}_e^q(\boldsymbol{b}_e^q + \frac{4q}{T}\boldsymbol{b}_e^{q-1}|\boldsymbol{\Delta}_e|) = \Psi + \frac{4q}{T}\sum_e \boldsymbol{w}_e^q\boldsymbol{b}_e^{q-1}|\boldsymbol{\Delta}_e| \\ &\leq \Psi + \frac{4q}{T}\|\mathbf{L}\boldsymbol{\Delta}\|_1 \leq \Psi + \frac{4qK^q}{T}\end{aligned}$$

where we use $(1+x)^q \leq 1+4qx$ for $x \in [0, 1/100q]$. ∎

LEMMA 5.6. (FINAL POTENTIAL) *When Algorithm 2 returns $\boldsymbol{c}^{(T)}$, we have $\Phi^{(T)} \leq 4K^2$ and $\Psi^{(T)} \leq 5qK^q$. Therefore, we have $\|\mathbf{R}\boldsymbol{c}^{(T)}\|_2 \leq 2K$ and $\|\mathbf{W}\boldsymbol{c}^{(T)}\|_p \leq \|\mathbf{W}\boldsymbol{c}^{(T)}\|_q \leq 2K$.*

Proof. This follows directly from lemma 5.3, lemma 5.4, and lemma 5.5. ∎

LEMMA 5.7. (EXISTENCE OF A GOOD ℓ_1 SOLUTION) *If there is a circulation $\boldsymbol{c}^*$ such that $\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle = -1, \|\mathbf{R}\boldsymbol{c}^*\|_2 \leq 1$, and $\|\mathbf{W}\boldsymbol{c}^*\|_p \leq 1$, we have $\|\mathbf{L}^{(i)}\boldsymbol{c}^*\|_1 \leq 20qK^{q-1}$ at any iteration i.*

Proof. In this proof, we consider an arbitrary iteration i and ignore the superscripts (i) for simplicity. By definition and the bounds on Φ and Ψ (lemma 5.6), we have by the Cauchy-Schwarz inequality and Hölder inquality that,

$$\begin{aligned}
\|\mathbf{L}\boldsymbol{c}^*\|_1 &= K^{q-2} \sum_e \boldsymbol{r}_e^2 \boldsymbol{a}_e |\boldsymbol{c}_e^*| + \sum_e \boldsymbol{w}_e^q \boldsymbol{b}_e^{q-1} |\boldsymbol{c}_e^*| \\
&\leq K^{q-2} \|\mathbf{R}\boldsymbol{a}\|_2 \|\mathbf{R}\boldsymbol{c}^*\|_2 + \left\|\mathbf{W}^{q-1}\boldsymbol{b}^{q-1}\right\|_{q/(q-1)} \|\mathbf{W}\boldsymbol{c}^*\|_q \\
&= K^{q-2} \|\mathbf{R}\boldsymbol{a}\|_2 \|\mathbf{R}\boldsymbol{c}^*\|_2 + \|\mathbf{W}\boldsymbol{b}\|_q^{q-1} \|\mathbf{W}\boldsymbol{c}^*\|_q \\
&\leq K^{q-2}(2K) + (5qK^q)^{(q-1)/q} \cdot 2 \\
&\leq 2K^{q-1} + 10qK^{q-1} \leq 20qK^{q-1}
\end{aligned}$$

where the second inequality comes from $\|\mathbf{W}\boldsymbol{c}^*\|_q \leq m^{1/q-1/p}\|\mathbf{W}\boldsymbol{c}^*\|_p \leq m^{1/\log_2 m} = 2$. ∎

Proof. [Proof of Lemma 4.5] We first show that if line 16 of Algorithm 1 occurs, then the claim about $\boldsymbol{c}^*$ in the following line 17 is true. The correctness of lemma 5.2 says that there's no circulation $\boldsymbol{c}^*$ such that $\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle = -1$ and $\|\widetilde{\mathbf{L}}\boldsymbol{c}^*\|_1 \leq 1/\alpha = 20qK^{q-1}$. Correctness then follows from lemma 5.7.

Next, the quality of the output $\boldsymbol{c}^{(T)}$ is established by lemma 5.6.

Finally, the runtime follows from lemma 5.2 and the standard technique of maintaining $\boldsymbol{a}, \boldsymbol{b}$ and $\widetilde{\boldsymbol{\ell}}$ using dynamic tree data structures such as link-cut trees. In particular, $\boldsymbol{\ell}$ is monotone and bounded by $m^{O(p)}$ on each edge, so we can maintain a $\widetilde{\boldsymbol{\ell}} \approx_2 \boldsymbol{\ell}$ with $\widetilde{O}(p)$ changes per change. ∎

6 Dynamic Min-Ratio Cycle Data Structures

In this section, we prove lemma 5.2. We use the oblivious min-ratio cycle data structure from [36]. In particular, the data structure can handle adversaries stronger than oblivious ones as long as the update sequence satisfies what is called the *hidden stable-flow chasing* property. In the problem of approximate monotonic min-ratio cycles, we show that the update sequences satisfy this property and the monotonicity in edge lengths allows us to use a special case of the data structure in [36].

DEFINITION 6.1. (HIDDEN STABLE-FLOW CHASING UPDATES, DEFINITION 6.1 [36]) *Consider a dynamic graph $G^{(t)}$ undergoing batches of updates $U^{(1)}, \ldots, U^{(t)}, \ldots$ consisting of edge insertions and deletions. We say the sequences $\boldsymbol{g}^{(t)}, \boldsymbol{\ell}^{(t)}$, and $U^{(t)}$ satisfy the* hidden stable-flow chasing *property if there are hidden dynamic circulations $\boldsymbol{c}^{(t)}$ and hidden dynamic upper bounds $\boldsymbol{w}^{(t)}$ such that the following holds at all stages t:*

1. *$\boldsymbol{c}^{(t)}$ is a circulation: $\mathbf{B}_{G^{(t)}}^\top \boldsymbol{c}^{(t)} = 0$.*
2. *$\boldsymbol{w}^{(t)}$ upper bounds the length of $\boldsymbol{c}^{(t)}$: $|\boldsymbol{\ell}_e^{(t)} \boldsymbol{c}_e^{(t)}| \leq \boldsymbol{w}_e^{(t)}$ for all $e \in E(G^{(t)})$.*
3. *For any edge e in the current graph $G^{(t)}$, and any stage $t' \leq t$, if the edge e was already present in $G^{(t')}$, i.e. $e \in G^{(t)} \setminus \bigcup_{s=t'+1}^{t} U^{(s)}$, then $\boldsymbol{w}_e^{(t)} \leq 2\boldsymbol{w}_e^{(t')}$.*

THEOREM 6.1. (SIMPLER VERSION OF [36, THEOREM 7.1]) *Let $G = (V, E)$ be a dynamic graph undergoing τ batches of updates $U^{(1)}, \ldots, U^{(\tau)}$ containing* only *edge insertions and deletions with edge gradient $\boldsymbol{g}^{(t)}$ and length $\boldsymbol{\ell}^{(t)}$ such that the update sequence satisfies the hidden stable-flow chasing property (definition 6.1) with hidden dynamic circulation $\boldsymbol{c}^{(t)}$ and width $\boldsymbol{w}^{(t)}$. In addition, $\|\boldsymbol{w}^{(t)}\|_1$ is non-decreasing in t, i.e., $\|\boldsymbol{w}^{(t)}\|_1 \leq \|\boldsymbol{w}^{(t+1)}\|_1$ for all steps t.*

There is an algorithm on G that for $d = (\log m)^{1/8}$ maintains a collection of $s = O(\log m)^d$ spanning trees $T_1, T_2, \ldots, T_s$ and after each update outputs a circulation $\boldsymbol{\Delta}$ represented by

$\exp(O(\log^{7/8} m \log\log m))$ *off-tree edges and paths on some* $T_i, i \in [s]$. *The output circulation* $\boldsymbol{\Delta}$ *satisfies* $\mathbf{B}^\top \boldsymbol{\Delta} = 0$ *and for some* $\kappa = \exp(-O(\log^{7/8} m \log\log m))$

$$\frac{\langle \boldsymbol{g}^{(t)}, \boldsymbol{\Delta} \rangle}{\left\| \mathbf{L}^{(t)} \boldsymbol{\Delta} \right\|_1} \leq \kappa \frac{\langle \boldsymbol{g}^{(t)}, \boldsymbol{c}^{(t)} \rangle}{(d+1)\|\boldsymbol{w}^{(t)}\|_1}$$

The algorithm succeeds whp. with total runtime $(m+Q)m^{o(1)}$ *for* $Q \stackrel{\text{def}}{=} \sum_{t=1}^{\tau} \left|U^{(t)}\right| \leq \text{poly}(n)$.

This follows from [36, Theorem 7.1] because the terms $\|\boldsymbol{w}^{(\mathsf{prev}_i^{(t)})}\|_1 \leq \|\boldsymbol{w}^{(t)}\|_1$ by the increasing property, and $\mathsf{prev}_i^{(t)} \leq t$.

Proof. [Proof of lemma 5.2] We apply the dynamic algorithm of theorem 6.1 to prove the lemma. In order to do so, we need to show that there is a hidden circulation $\boldsymbol{c}^{(t)}$ and width $\boldsymbol{w}^{(t)}$ that satisfies the hidden stable-flow chasing property (definition 6.1). At any stage t, if the monotonic MRC problem instance (problem 5.1) inserts a new edge e with gradient $\boldsymbol{g}_e$ and length $\boldsymbol{\ell}_e$, we create an update batch $U^{(t)}$ consists of a single edge insertion $(e, \boldsymbol{g}_e, \boldsymbol{\ell}_e)$. Otherwise, if the problem instance increases the length of an edge from $\boldsymbol{\ell}_e$ to $\boldsymbol{\ell}'_e$, we create an update batch $U^{(t)}$ consists of an edge deletion e followed by an edge insertion $(e, \boldsymbol{g}_e, \boldsymbol{\ell}'_e)$.

At any stage t (including stage 0, the moment after the initialization), we define the hidden circulation $\boldsymbol{c}^{(t)}$ to be $\boldsymbol{c}^*$ if every edge in the circulation $\boldsymbol{c}^*$ appears in $G^{(t)}$. Otherwise, we set $\boldsymbol{c}^{(t)}$ to be the all zero circulation. We also define the hidden width $\boldsymbol{w}^{(t)}$ on each edge e as

$$\boldsymbol{w}_e^{(t)} \stackrel{\text{def}}{=} \begin{cases} \left|\boldsymbol{\ell}_e^{(t)} \boldsymbol{c}_e^*\right| & \text{, if } e \in \text{supp}(\boldsymbol{c}^*) \\ 0 & \text{, otherwise.} \end{cases}$$

Now, we verify each condition of definition 6.1 one at a time. item 1 holds at any stage t because both $\boldsymbol{c}^*$ and $\mathbf{0}$ are circulations. For any edge $e \in \text{supp}(\boldsymbol{c}^*)$, $\boldsymbol{c}_e^{(t)}$ is either 0 or $\boldsymbol{c}_e^*$ and $\boldsymbol{w}_e^{(t)} \geq |\boldsymbol{\ell}^{(t)} \boldsymbol{c}_e^{(t)}|$ holds. For any other edge $e \notin \text{supp}(\boldsymbol{c}^*)$, both $\boldsymbol{c}_e^{(t)}$ and $\boldsymbol{w}_e^{(t)}$ are always 0 and item 2 follows. item 3 follows from the fact that $\boldsymbol{c}^*$ is fixed beforehand and that an edge is updated when its length doubles. Also, $\|\boldsymbol{w}^{(t)}\|_1$ is non-decreasing in t because edge lengths are non-decreasing.

Finally, we need to show that the output after each update is always valid. At stage t, if $\text{supp}(\boldsymbol{c}^*) \not\subseteq G^{(t)}$, problem 5.1 does not care what we output. If $\text{supp}(\boldsymbol{c}^*) \subseteq G^{(t)}$, we know $\boldsymbol{g}^{(t)} = \boldsymbol{g}$, $\boldsymbol{c}^{(t)} = \boldsymbol{c}^*$ and $\|\boldsymbol{w}^{(t)}\|_1 = \|\mathbf{L}^{(t)} \boldsymbol{c}^*\|_1$, and the output circulation $\boldsymbol{\Delta}$ satisfies

$$\frac{\langle \boldsymbol{g}, \boldsymbol{\Delta} \rangle}{\left\| \mathbf{L}^{(t)} \boldsymbol{\Delta} \right\|_1} \leq \kappa \frac{\langle \boldsymbol{g}, \boldsymbol{c}^{(t)} \rangle}{(d+1)\|\boldsymbol{w}^{(t)}\|_1} = \frac{\kappa}{d+1} \frac{\langle \boldsymbol{g}, \boldsymbol{c}^* \rangle}{\|\mathbf{L}^{(t)} \boldsymbol{c}^*\|_1}$$

from theorem 6.1. The conclusion follows because $d = (\log m)^{1/8}$. ∎

References

[1] Deeksha Adil, Brian Bullins, Rasmus Kyng, and Sushant Sachdeva. Almost-linear-time weighted ℓ_p-norm solvers in slightly dense graphs via sparsification. *arXiv preprint arXiv:2102.06977*, 2021.

[2] Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for ℓ_p-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1405–1424. SIAM, 2019.

[3] Deeksha Adil, Richard Peng, and Sushant Sachdeva. Fast, provably convergent irls algorithm for p-norm linear regression. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] Deeksha Adil and Sushant Sachdeva. Faster p-norm minimizing flows, via smoothed q-norm problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 892–910. SIAM, 2020.

[5] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *ICALP*, volume 107 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[6] Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. *CoRR*, abs/2207.09354, 2022.

[7] Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–104. IEEE, 2020.
[8] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. In *FOCS*, pages 528–539. IEEE, 2021.
[9] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in o(log n) update time. *SIAM J. Comput.*, 44(1):88–113, 2015.
[10] Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 129–162. SIAM, 2023.
[11] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, pages 382–405. IEEE Computer Society, 2019.
[12] Soheil Behnezhad and Sanjeev Khanna. New trade-offs for fully dynamic matching via hierarchical EDCS. In *SODA*, pages 3529–3566. SIAM, 2022.
[13] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully dynamic matching: Beating 2-approximation in δ^ϵ update time. In *SODA*, pages 2492–2508. SIAM, 2020.
[14] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. *ACM Trans. Algorithms*, 17(4):29:1–29:51, 2021.
[15] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134. IEEE, 2020.
[16] Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. *J. ACM*, 66(5):37:1–37:23, 2019.
[17] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015.
[18] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *SODA*, pages 692–711. SIAM, 2016.
[19] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, pages 398–411. ACM, 2016.
[20] Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *ICALP*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
[21] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time. *arXiv preprint arXiv:2302.05030*, 2023.
[22] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–47. SIAM, 2023.
[23] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 100–128. SIAM, 2023.
[24] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $o(1/\epsilon^2)$ amortized update time. In *SODA*, pages 1872–1885. SIAM, 2019.
[25] Joakim Blikstad and Peter Kiss. Incremental (1-ϵ)-approximate dynamic matching in o(poly$(1/\epsilon)$) update time. *CoRR*, abs/2302.08432, 2023.
[26] Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *FOCS*, pages 384–393. IEEE Computer Society, 2014.
[27] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2022 IEEE 64rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2023.
[28] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *STOC*, pages 543–556. ACM, 2022.
[29] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *STOC*, pages 859–869. ACM, 2021.
[30] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020.
[31] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *FOCS*, pages 456–480. IEEE Computer Society, 2019.
[32] Parinya Chalermsook, Syamantak Das, Yunbum Kook, Bundit Laekhanukit, Yang P Liu, Richard Peng, Mark Sellke,

and Daniel Vaz. Vertex sparsification for edge connectivity. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1206–1225. SIAM, 2021.

[33] Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *ICALP*, volume 107 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[34] Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *FOCS*, pages 370–381. IEEE Computer Society, 2019.

[35] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1135–1146. IEEE, 2020.

[36] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022. https://arxiv.org/abs/2203.00671.

[37] Paul Christiano, Jonathan A. Kelner, Aleksander Mądry, Daniel A. Spielman, and Shang-Hua 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*, pages 273–282. ACM, 2011. Available at https://arxiv.org/abs/1010.2921.

[38] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 343–352, 2014.

[39] Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7})$ time (extended abstract). In *SODA*, pages 752–771. SIAM, 2017.

[40] Soren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *ICALP*, volume 55 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[41] Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 451–460, 2008.

[42] Debarati Das, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. A near-optimal offline algorithm for dynamic all-pairs shortest paths in planar digraphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3482–3495. SIAM, 2022.

[43] Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In *STOC*, pages 716–725. ACM, 1994.

[44] Yefim Dinitz and Alek Vainshtein. Locally orientable graphs, cell structures, and a new algorithm for the incremental maintenance of connectivity carcasses. In *SODA*, pages 302–311. ACM/SIAM, 1995.

[45] Yefim Dinitz and Jeffery R. Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20(3):242–276, 1998.

[46] Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 124–153, 2022.

[47] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. In *STOC*, pages 914–925. ACM, 2019.

[48] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[49] Greg N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. In *FOCS*, pages 632–641. IEEE Computer Society, 1991.

[50] Zvi Galil and Giuseppe F. Italiano. Fully dynamic algorithms for edge-connectivity problems (extended abstract). In *STOC*, pages 317–327. ACM, 1991.

[51] Zvi Galil and Giuseppe F. Italiano. Fully dynamic algorithms for edge-connectivity problems (extended abstract). In *STOC*, pages 317–327. ACM, 1991.

[52] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *FOCS*, pages 516–527. IEEE, 2021.

[53] Gramoz Goranci and Monika Henzinger. Efficient data structures for incremental exact and approximate maximum flow. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[54] Gramoz Goranci, Monika Henzinger, and Pan Peng. The power of vertex sparsifiers in dynamic graph algorithms. In *ESA*, volume 87 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[55] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *SODA*, pages 2212–2228. SIAM, 2021.

[56] Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. Maintaining an EDCS in general graphs:

Simpler, density-sensitive and with worst-case time bounds. In *SOSA*, pages 12–23. SIAM, 2022.

[57] Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, volume 29 of *LIPIcs*, pages 227–239. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

[58] Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set, maximum flow and maximum matching. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 86–91. SIAM, 2021.

[59] Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *FOCS*, pages 548–557. IEEE Computer Society, 2013.

[60] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015.

[61] Monika Rauch Henzinger and Valerie King. Fully dynamic 2-edge connectivity algorithm in polylogarithmic time per operation. *SRC Technical Note*, 4, 1997.

[62] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[63] Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\tilde{O}(\log^2 n)$ amortized time. In *SODA*, pages 35–52. SIAM, 2018.

[64] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *STOC*, pages 313–322. ACM, 2011.

[65] Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Regularized box-simplex games and dynamic decremental bipartite matching. In *ICALP*, volume 229 of *LIPIcs*, pages 77:1–77:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[66] Arun Jambulapati and Aaron Sidford. Ultrasparse ultrasparsifiers and faster laplacian system solvers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 540–559. SIAM, 2021.

[67] Wenyu Jin and Xiaorui Sun. Fully dynamic s-t edge connectivity in subpolynomial time (extended abstract). In *FOCS*, pages 861–872. IEEE, 2021.

[68] Tarun Kathuria, Yang P. Liu, and Aaron 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*, pages 119–130. IEEE, 2020.

[69] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226. SIAM, 2014.

[70] Peter Kiss. Improving update times of dynamic matching algorithms from amortized to worst case. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, page 94, 2022.

[71] Ioannis Koutis, Gary L Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE, 2011.

[72] S. Kumar and P. Gupta. An incremental algorithm for the maximum flow problem. *J. Math. Model. Algorithms*, 2(1):1–16, 2003.

[73] Rasmus Kyng, Richard Peng, Sushant Sachdeva, and Di Wang. Flows in almost linear time via adaptive preconditioning. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 902–913, 2019.

[74] Hung Le, Lazar Milenkovic, Shay Solomon, and Virginia Vassilevska Williams. Dynamic matching algorithms under vertex updates. In *ITCS*, volume 215 of *LIPIcs*, pages 96:1–96:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[75] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\widetilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow. In *FOCS*, pages 424–433. IEEE Computer Society, 2014.

[76] Huan Li, Stacy Patterson, Yuhao Yi, and Zhongzhi Zhang. Maximizing the number of spanning trees in a connected graph. *IEEE Trans. Inf. Theory*, 66(2):1248–1260, 2020.

[77] Yang P Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020.

[78] Aleksander Mądry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

[79] Aleksander Mądry. 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*, pages 593–602. IEEE Computer Society, 2016. Available at https://arxiv.org/abs/1608.06016.

[80] David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *SODA*, pages 712–729. SIAM, 2016.

[81] Richard 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*, pages 1862–1867. SIAM, 2016.
[82] Mohammad Roghani, Amin Saberi, and David Wajc. Beating the folklore algorithm for dynamic matching. In *ITCS*, volume 215 of *LIPIcs*, pages 111:1–111:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
[83] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126. SIAM, 2007.
[84] Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269. IEEE Computer Society, 2013.
[85] Jonah Sherman. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017.
[86] Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, pages 325–334. IEEE Computer Society, 2016.
[87] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 563–568, 2008.
[88] Daniel A. Spielman and Shang-Hua 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*, pages 81–90, 2004. Available at `https://arxiv.org/abs/0809.3232`, `https://arxiv.org/abs/0808.4134`, `https://arxiv.org/abs/cs/0607105`.
[89] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC*, pages 343–350. ACM, 2000.
[90] Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.
[91] Jan van den Brand, Yang P Liu, and Aaron Sidford. Dynamic maxflow via dynamic interior point methods. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1215–1228, 2023.
[92] David Wajc. Rounding dynamic matchings against an adaptive adversary. In *STOC*, pages 194–207. ACM, 2020.
[93] Jeffery R. Westbrook and Robert Endre Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(5&6):433–464, 1992.