Almost-Linear Time Algorithms for Decremental Graphs: Min-Cost Flow and More via Duality

Jan van den Brand
School of Computer Science
Georgia Tech
Atlanta, USA
vdbrand@gatech.edu

Li Chen
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
lichenntu@gmail.com

Rasmus Kyng
Department of Computer Science
ETH Zurich
Zurich, Switzerland
kyng@inf.ethz.ch

Yang P. Liu
School of Mathematics
Institute for Advanced Study
Princeton, USA
yangpliu@ias.edu

Simon Meierhans
Department of Computer Science
ETH Zurich
Zurich, Switzerland
mesimon@inf.ethz.ch

Maximilian Probst Gutenberg
Department of Computer Science
ETH Zurich
Zurich, Switzerland
maximilian.probst@inf.ethz.ch

Sushant Sachdeva
Department of Computer Science
University of Toronto
Toronto, Canada
sachdeva@cs.toronto.edu

2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS) | 979-8-3315-1674-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/FOCS61266.2024.00120

***Abstract*—We give the first almost-linear total time algorithm for deciding if a flow of cost at most F still exists in a directed graph, with edge costs and capacities, undergoing decremental updates, i.e., edge deletions, capacity decreases, and cost increases. This implies almost-linear time algorithms for approximating the minimum-cost flow value and s-t distance on such *decremental* graphs. Our framework additionally allows us to maintain decremental strongly connected components in almost-linear time deterministically. These algorithms also improve over the current best known runtimes for *statically* computing minimum-cost flow, in both the randomized and deterministic settings.**

We obtain our algorithms by taking the dual perspective, which yields cut-based algorithms. More precisely, our algorithm computes the flow via a sequence of $m^{1+o(1)}$ dynamic *min-ratio cut* problems, the dual analog of the dynamic min-ratio cycle problem that underlies recent fast algorithms for minimum-cost flow. Our main technical contribution is a new data structure that returns an approximately optimal min-ratio cut in amortized $m^{o(1)}$ time by maintaining a *tree-cut sparsifier*. This is achieved by devising a new algorithm to maintain the dynamic expander hierarchy of [Goranci-Räcke-Saranurak-Tan, SODA 2021] that also works in *capacitated* graphs. All our algorithms are deterministc, though they can be sped up further using randomized techniques while still working against an adaptive adversary.

Jan van den Brand was supported by NSF Grant CCF-2338816. Li Chen was supported by NSF Grant CCF-2330255. The research of Rasmus Kyng, Simon Meierhans and Maximilian Probst Gutenberg leading to these results has received funding from grant no. 200021 204787 of the Swiss National Science Foundation. Yang P. Liu acknowledges that this material is based upon work supported by the National Science Foundation under Grant No. DMS-1926686. Sushant Sachdeva's research is supported by an Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2018-06398, an Ontario Early Researcher Award (ERA) ER21-16-283, and a Sloan Research Fellowship.

***Index Terms*—Maximum flow, Minimum cost flow, Data structures, Interior point methods, Convex optimization**

I. Introduction

The study of *dynamic graph algorithms* involves designing efficient algorithms for graphs undergoing edge updates. In this paper, we focus on solving the challenging *minimum-cost flow* problem on directed graphs in the *decremental* setting, where the graph undergoes updates that guarantee that the optimal cost is non-decreasing. Henceforth, decremental updates consist of edge deletions, cost increases, and capacity decreases. The minimum-cost flow problem generalizes the s-t shortest path and the more general single-source shortest-path (SSSP) problem that have received significant attention in the decremental setting [1]–[6], which are all not known to admit almost-linear-time algorithms, even against oblivious adversaries. In this paper, we give almost-linear-time algorithms for several problems in decremental graphs, which are primarily derived by solving the more general problem of *decremental thresholded min-cost flow*.

Definition I.1. *The thresholded min-cost flow problem is defined on a directed graph $G = (V, E)$ with capacities $\boldsymbol{u}$ and costs $\mathbf{c}$, undergoing decremental updates (edge deletions, edge capacity decreases, and cost increases) along with a threshold F and demands $\boldsymbol{d} \in \mathbb{R}^V$. A dynamic algorithm solves the problem if, after each update, the algorithm outputs whether there is a feasible flow $\boldsymbol{f}$ routing demand $\boldsymbol{d}$ with cost $\mathbf{c}^\top \boldsymbol{f}$ at most F, or answers that no such flow exists.*

2575-8454/24/$31.00 ©2024 IEEE
DOI 10.1109/FOCS61266.2024.00120

The thresholded min-cost flow problem in *incremental* graphs (undergoing edge insertions) was recently shown to have an almost-linear-time algorithm in [7]. In this paper, we show that the decremental version can also be solved in almost-linear-time (see Theorem I.6 for a formal statement).

Informal Theorem I.2. *There is a deterministic algorithm that solves the decremental thresholded min-cost flow problem on graphs with m edges initially, undergoing Q updates in total time $(m+Q)m^{o(1)}$, provided that costs, capacities, and demands are integral and polynomially bounded in m.*

This result and its extensions give almost-linear-time deterministic algorithms for decremental approximate min-cost flow value, single-source reachability, strongly connected component maintenance, and s-t distance.

Towards proving this result, let us recall the approach of [7], which builds on the almost-linear-time min-cost flow algorithm of [8]. The algorithm of [8] used an ℓ_1-based interior point method (IPM) to solve min-cost flow via a sequence of dynamic *min-ratio cycle* problems, with approximation quality $\alpha = m^{o(1)}$. Later, [9] showed that giving an algorithm with amortized $m^{o(1)}$ update time for approximate dynamic min-ratio cycle against adaptive adversaries (which was not achieved in [8]) suffices for incremental thresholded min-cost flow. Such a data structure for dynamic min-ratio cycle was developed in [7].

There is one key difference between the incremental and decremental settings: a feasible flow $\boldsymbol{f}$ continues to be feasible under edge insertions, but not under edge deletions. To handle this, we instead work with a dual version of the min-cost flow problem. More precisely, there is a standard reduction between min-cost flow and transshipment: $\min_{\boldsymbol{B}^\top \boldsymbol{f}=\boldsymbol{d}, \boldsymbol{f}\geq 0} \boldsymbol{c}^\top \boldsymbol{f}$, where $\boldsymbol{B}$ is the edge-vertex incidence matrix of the underlying graph G. The dual of this problem, computed via strong duality, is

$$\max_{\boldsymbol{c}-\boldsymbol{B}\boldsymbol{y}\geq 0} \boldsymbol{d}^\top \boldsymbol{y}. \tag{1}$$

Note that in this dual formulation, if a solution $\boldsymbol{y}$ is feasible, i.e., $\boldsymbol{c}-\boldsymbol{B}\boldsymbol{y}\geq 0$, then it continues to be feasible after an edge deletion or cost increase. It turns out that $\langle \boldsymbol{d}, \boldsymbol{y}\rangle$ is also monotone increasing in the transshipment instance under capacity decreases. Thus, it is natural to work with the dual problem in the decremental setting.

To give an almost-linear-time algorithm for solving (1), we broadly follow the approach set forth by [8]. We first design an ℓ_1-IPM which solves (1) via a sequence of $m^{1+o(1)}$ dynamic *min-ratio cut* problems (see Section VI), defined below.

Definition I.3 (α-approximate dynamic min-ratio cut)**.** *The dynamic min-ratio cut problem is defined on an undirected graph G with capacities $\boldsymbol{u}\in\mathbb{R}^E_{\geq 0}$, and vertex gradient $\boldsymbol{g}\in\mathbb{R}^V$. At each time step, the gradient of a single vertex or the length of a single edge may be updated, in a fully-dynamic manner. We let $\boldsymbol{B}$ be the edge-vertex incidence matrix of G after assigning an arbitrary orientation to each edge.*

A dynamic algorithm solves the problem, if after the i-th update, it identifies a cut $\boldsymbol{z}\in\{0,1\}^V$, such that

$$\frac{\langle \boldsymbol{g}, \boldsymbol{z}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{z}\|_1} \leq \frac{1}{\alpha}\min_{\boldsymbol{\phi}\neq 0}\frac{\langle \boldsymbol{g}, \boldsymbol{\phi}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\phi}\|_1}.$$

It is worth pointing out that $\min_{\boldsymbol{\phi}\neq 0}\frac{\langle \boldsymbol{g},\boldsymbol{\phi}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\phi}\|_1}$ is a non-positive quantity by symmetry. We also stress that the min-ratio cut problem does not depend on the orientations chosen for edges when defining the edge-vertex incidence matrix $\boldsymbol{B}$. Our main technical contribution is an algorithm that solves dynamic min-ratio cut in amortized $m^{o(1)}$ time with $\alpha = m^{o(1)}$ approximation, under fully-dynamic updates against adaptive adversaries.

To solve the min-ratio cut problem approximately, we fully-dynamically maintain an ℓ_∞-oblivious routing for G which is realized by a single tree T, often referred to as a *tree cut sparsifier*. We then show that on this tree, we can solve the min-ratio cut problem, and these cuts are good approximate solutions to the min-ratio cut problem on G. We give a formal definition of these tree cut sparsifiers, since they are crucial to our result.

Definition I.4 (Tree Cut Sparsifier)**.** *Given graph $G=(V,E,\boldsymbol{u})$, a* tree cut sparsifier *$T=(V',E',\boldsymbol{u}')$ of quality q is a tree graph with $V\subseteq V'$ such that for every pair of disjoint sets $A,B\subseteq V$, we have that* $\mathrm{mincut}_G(A,B)\leq \mathrm{mincut}_T(A,B)\leq q\cdot \mathrm{mincut}_G(A,B)$.

Tree cut sparsifiers, in turn, are associated with dynamic expander hierarchies as introduced in [10]. Loosely speaking, an expander hierarchy computes an expander decomposition of an underlying graph, contracts each expander piece to a single vertex, and recursively computes more expander decompositions, contractions, etc. This naturally induces a tree structure, which [10] proves is a tree cut-sparsifier of quality $q=m^{o(1)}$ in unit capacity graphs. We give the first non-trivial algorithm for maintaining dynamic expander hierarchies and thus tree cut sparsifiers in *capacitated* graphs. In fact, our algorithm is optimal up to subpolynomial factors.

Informal Theorem I.5. *Given an m-edge graph $G=(V,E,\boldsymbol{u})$ with polynomially bounded capacities that undergoes $\widetilde{O}(m)$ edge insertions/deletions, then there is a deterministic algorithm that maintains a tree cut sparsifier T of quality $m^{o(1)}$ in total update time $m^{1+o(1)}$.*

A. Comparison to Earlier Minimum-Cost Flow Algorithms

Our results build on the ℓ_1-interior point method introduced in the first almost-linear time minimum-cost flow algorithm [8]. The primal ℓ_1-IPM of [8] and later works [7], [9], [11] requires solving a dynamic min-ratio cycle problem. This problem is solved using data structures that fundamentally center around distance approximation in undirected graphs. Our dual ℓ_1-IPM requires us to solve a dynamic min-ratio cut problem, which instead requires cut approximation in undirected graphs. The dual perspective turns out to be very natural in retrospect, and has two striking consequences: Firstly, our

dual approach enables us to solve decremental graph problems, similar to how incremental graph problems were solved in [7], [9], [11] using primal algorithms, essentially because dual solutions stay feasible under edge deletions while primal solutions stay feasible under edge insertions. Secondly, the dual approach yields methods based on cut geometry instead of distance geometry, motivating us to develop fully-dynamic tree cut sparsifiers for weighted graphs, a powerful data structure for answering cut queries. Notably, our cut data structures are substantially simpler than earlier approaches. We expand on this comparison in Section I-B below.

In seeking to develop our dynamic cut approximation data structures, we encounter challenges that are morally similar to those of [7], [8], [12] which developed extensive new machinery for maintaining fully-dynamic low-stretch trees and ℓ_1-oblivious routings in weighted graphs. While fast dynamic algorithms to maintain LSSTs for unit capacity graphs existed previously [13]–[15], a central technical contribution in each of [7], [8], [12] is a dynamic algorithm to maintain LSSTs or ℓ_1-oblivious routings respectively in capacitated graphs. This turns out to require a very different set of tools, and the resulting algorithms deviate heavily from algorithms designed for unit capacity graphs, and instead build on ideas from [16]–[19].

We are faced with a similar challenge in constructing fully-dynamic tree cut sparsifiers. A striking prototype data structure for the unit capacity case was built via the expander hierarchy in [10], but their methods face major obstacles in extending to the weighted case. Our construction is motivated by their result but takes as its starting point a later generation of expander decomposition methods [20], [21]. These methods yield particularly clean expander hierarchies in unweighted decremental graphs, and we show how to extend these methods to weighted graphs using a new reduction from weighted to unweighted graphs in this setting. Using *core graph* techniques motivated by [7], [8], [22], we finally reduce the fully-dynamic tree cut sparsifier problem to the decremental case.

B. Applications

a) Application #1: Faster static min-cost flow.: Somewhat surprisingly, the expander hierarchy data structure has fewer recursive levels than those for min-ratio cycle. This results in a faster runtime for static min-cost flow for both randomized and deterministic algorithms. In particular, we give a randomized algorithm that statically solves exact min-cost flow on graphs with polynomially bounded costs and capacities in time $m \cdot e^{O((\log m)^{3/4} \log\log m)}$, and a deterministic version that runs in time $m \cdot e^{O((\log m)^{5/6} \log\log m)}$. This should be compared to randomized $m \cdot e^{O((\log m)^{7/8} \log\log m)}$ time [8], and deterministic $m \cdot e^{O((\log m)^{17/18} \log\log m)}$ time [12] respectively.

Our algorithm initially solves the dual (1) and is able to use the final IPM dual solution to extract an optimal flow (see the full version). Our approach is arguably the simplest almost-linear time algorithm for computing minimum-cost flows yet: Our data structure only needs one main component, namely a fully-dynamic tree cut sparsifier, obtained from our dynamic expander hierarchy. In the randomized setting, the tools required to implement this expander hierarchy primarily involve a direct reduction from capacitated expander decomposition to the unit capacity setting. Finally, on top of this, we build a data structure for detecting the best tree cut, using standard techniques. In contrast, the data structure for solving min-ratio cycle in the first almost-linear time min-cost flow algorithm of [8] is quite involved. It relies on complex fully-dynamic spanners, core graphs, all-pairs shortest path data structures, and a delicate restarting procedure to manage the interaction between a non-fully-adaptive data structure and its 'adversary' coming from the interior point method. This need for reasoning about the interaction between a data structure and an adversary was removed in [7], which gave a (deterministic) fully-adaptive data structure for solving min-ratio cycle problems, but introduced other complexities by using extensive machinery to maintain fully-dynamic ℓ_1-oblivious routings using dynamic terminal spanners and low-diameter trees [23].

b) Application #2: Decremental min-cost flow: By designing an ℓ_1-IPM for (1), and an efficient data structure for min-ratio cuts (Definition I.3), we show the following.

Theorem I.6. *There is a randomized algorithm that given a decremental graph $G = (V, E, \boldsymbol{u}, \boldsymbol{c})$ with integer capacities $\boldsymbol{u}$ in $[1, U]$ and integer costs $\boldsymbol{c}$ in $[-C, C]$, with $U, C \leq m^{O(1)}$, where m is the initial number of edges in G, a demand $\boldsymbol{d} \in \mathbb{Z}^V$, and parameter $F \in \mathbb{R}$ reports after each edge deletion, capacity decrease, or cost increase, whether there is a feasible flow $\boldsymbol{f}$ with cost $\boldsymbol{c}^\top \boldsymbol{f}$ at most F. Over Q updates, the algorithm runs in total time $(m+Q) \cdot e^{O((\log m)^{3/4} \log\log m)}$, and can be made deterministic with time $(m+Q) \cdot e^{O((\log m)^{5/6} \log\log m)}$.*

A corollary of Theorem I.6 is an algorithm for approximately maintaining the flow value. Because Theorem I.6 solves a thresholded problem, it succeeds against adaptive adversaries. Because Theorem I.7 reduces to the thresholded problem, it also succeeds against adaptive adversaries.

Theorem I.7. *There is a randomized algorithm that given a decremental graph $G = (V, E, \boldsymbol{u}, \boldsymbol{c})$ with integer capacities $\boldsymbol{u}$ in $[1, U]$ and integer costs $\boldsymbol{c}$ in $[1, C]$, with $U, C \leq m^{O(1)}$, where m is the initial number of edges in G, a demand $\boldsymbol{d} \in \mathbb{Z}^V$, maintains a $(1+\epsilon)$-approximation to the cost of the current min-cost flow. Over Q updates, the algorithm runs in total time $\epsilon^{-1}(m+Q) \cdot e^{O((\log m)^{3/4} \log\log m)}$, and can be made deterministic in time $\epsilon^{-1}(m+Q) \cdot e^{O((\log m)^{5/6} \log\log m)}$.*

Proof. We run a thresholded decremental min-cost flow algorithm (Theorem I.6) for thresholds $F = (1+\epsilon)^i$. Note that the cost the min-cost flow is monotonically increasing because the graph is decremental. The cost is lower bounded by 1, and upper bounded by mCU, so it suffices to set $i = 0, 1, \ldots, O(\epsilon^{-1} \log(mCU))$. The result thus follows from Theorem I.6. □

This also implies approximation and threshold algorithms for maintaining the value of the decremental maximum flow and the size of a weighted bipartite matching. It is worth noting that the dependence on ϵ is optimal under the online matrix-vector (OMv) conjecture [24]. Indeed, exact decremental matching in unweighted graphs requires time at least $mn^{1-o(1)}$ under OMv. Our result should be compared to previous algorithms [6], [25], [26], with runtimes[1] $\widetilde{O}(m\epsilon^{-4})$, $\widetilde{O}(m\epsilon^{-3})$, and $m^{1+o(1)}\epsilon^{-2}$ respectively.

c) Application #3: Deterministic decremental single-source reachability and strongly connected components: A long line of work resulted in near-linear time algorithms for decremental single-source reachability (SSR) and strongly connected components (SCC) [1], [2], [5], [27], [28]. All these algorithms are randomized. They technically work against adaptive adversaries because the SCC decomposition or reachability structure does not reveal any randomness. However, they do not work against "non-oblivious" adversaries that can see all internal randomness of the algorithm. In general, non-oblivious or even deterministic algorithms are often more desirable so that they can be used within an optimization framework (such as IPMs or multiplicative weights). To the best of our knowledge, the current fastest deterministic algorithms for SSR and SCC require time $mn^{1/2+o(1)}$, achieved by [29].[2] We improve this runtime to $m^{1+o(1)}$.

Theorem I.8. *There is a deterministic algorithm that given a directed graph $G = (V, E)$ undergoing edge deletions, explicitly maintains the strongly connected components of G in total time $m \cdot e^{O((\log m)^{5/6} \log\log m)}$.*

There is a simple reduction from SSR to SCCs.

Corollary I.9. *There is a deterministic algorithm that, given a directed graph $G = (V, E)$ undergoing edge deletions and vertex $s \in V$, explicitly maintains the set of vertices reachable from s in G in total time $m \cdot e^{O((\log m)^{5/6} \log\log m)}$.*

Proof. For a graph $G = (V, E)$ and vertex $s \in V$, consider the graph $\widehat{G}$ which contains the edges in E, plus (t, s) for $t \in V$. The SCC containing s in $\widehat{G}$ is exactly the set of vertices reachable from s. If G is decremental, then so is $\widehat{G}$. Thus, the result follows from Theorem I.8. □

It is worth mentioning that the previous deterministic result in [29] allows for querying paths between vertices in the same SCC, in time proportional to the length of the returned path. We do not currently see how to use our methods to achieve this.

d) Application #4: Decremental s-t distance: The s-t shortest path problem is of particular interest, partly because a classic algorithm of Garg and Könemann [30] shows that a data structure which solves approximate s-t shortest path in decremental directed graphs (against an adaptive adversary, with path reporting) can be used to design a high-accuracy maximum flow algorithm with nearly the same runtime. Their algorithm is based on multiplicative weights, and hence initially only achieves a constant factor approximation. However, it works in directed graphs, so one can take the residual graph and repeat the argument to boost to high accuracy. Versions of this MWU framework have been instantiated in several settings [6], [22]. Even though we now know an almost-linear-time maximum flow algorithm, such an approach may provide an alternate algorithm not based on IPMs. Recently Chuzhoy and Khanna achieved a $n^{2+o(1)}$ runtime for bipartite matching via this approach [31], [32], by leveraging specific properties of the residual graphs encountered in a bipartite matching algorithm. Curiously, Theorems I.6 and I.7 indeed give an almost-linear-time algorithm for reporting the distance of the decremental s-t shortest path, though not a witness approximate shortest path itself. However, the algorithm uses an IPM, so even having access to a witness path would not lead to a more "combinatorial" maximum flow algorithm based only on MWU.

Before our result, the previous best-known runtimes against oblivious adversaries were $\widetilde{O}(n^2)$ in dense graphs [4] and $\widetilde{O}(mn^{3/4})$ in general [33], and deterministically/adaptively only runtimes of $n^{2+2/3+o(1)}$ and $O(mn)$ are known [29], [34] Our result is deterministic and hence works against adaptive adversaries. It should be noted that these prior works solve the more general problem of single-source shortest path, i.e., approximate shortest path lengths from a source s to every other vertex. They also support reporting approximate shortest paths. We do not know how to achieve either currently, for reasons similar to the ones discussed above regarding why we cannot report paths for SSR and SCC. However, we are mildly optimistic that it may be achievable with additional insights.

e) Application #5: Dynamic flow algorithms: Our data structure to maintain the expander hierarchy and tree cut sparsifier runs in graphs undergoing edge insertions and deletions with polynomially bounded edge capacities with randomized amortized update time and approximation quality $e^{O(\log^{3/4} \log\log m)}$. We further give a deterministic algorithm that achieves amortized update time and approximation quality $e^{O(\log^{5/6} \log\log m)}$. These bounds match the respective runtimes claimed in [10] but extend their result also to capacitated graphs.

By the same reductions as in [10], we obtain the first algorithm with subpolynomial update time and approximation ratio for various important flow and cut problems.

Theorem I.10. *There is a deterministic algorithm on a capacitated m-edge graph undergoing edge insertions and deletions with amortized update time $m^{o(1)}$ that can return an $m^{o(1)}$-approximation to queries for the following properties:*

- *s-t maximum flow, s-t minimum cut for any input pair $(s, t) \in V^2$;*
- *lowest conductance cut, sparsest cut; and*
- *multi-commodity flow, multi-cut, multi-way cut, and vertex cut sparsifiers.*

[1] We use $\widetilde{O}(\cdot)$ to hide $\text{polylog}(m)$ factors.

[2] [29] claims a runtime of $mn^{2/3+o(1)}$. Using the almost-linear-time deterministic max-flow algorithm from [12] to speed up a routine to embed directed expanders in [29], improves the runtime to $mn^{1/2+o(1)}$.

The former two queries are answered in worst-case time $\widetilde{O}(1)$, the last type of queries are answered in time $\widetilde{O}(k)$ where k is the number of multi-commodity flow pairs; k is the number of required sets in the multi-cut; k is the number of terminals in the multi-way cut; or k is the number of terminal vertices over which the vertex sparsifier is required.

Previously, similar results were obtained in unit capacity graphs by [10]. [35] gave algorithms for the first problem that achieve nearly-logarithmic quality while achieving sub-linear update time $\widetilde{O}(n^{2/3})$ in an n-vertex graph against an oblivious adversary and $\widetilde{O}(m^{3/4})$ time against an adaptive adversary. In [36], a deterministic algorithm with $m^{o(1)}$-approximation quality and update time was given which works in capacitated graphs undergoing edge insertions only.

C. Related Work

a) Dual-based flow algorithms: The work [37] provided a cut toggling alternative to the cycle toggling Laplacian solver of [38]. For the problem of decremental approximate bipartite matching, [25] provided an MWU algorithm on dual vectors that run in nearly-linear time. Additionally, [39] (see also [40]) gave a framework for undirected transshipment that was partially based on adjusting dual variables. The dual perspective is also crucial for the communication complexity of transshipment [41].

b) Previous min-cost flow algorithms: Following a long line of work [6], [17], [18], [42]–[50], the work [8] gave an almost-linear time algorithm for solving minimum-cost flow exactly in graphs with polynomially bounded integral costs and capacities. A series of works since then [7], [9], [11], [12] has made the algorithm deterministic and has given an algorithm for maintaining a minimum-cost flow in incremental graphs [7]. Earlier works primarily used electrical flows to make progress, and recent works use approximate minimum-ratio cycles. Our work provides an alternative approach that instead solves minimum-ratio cut problems. Our algorithm has fewer recursive layers and results in a faster runtime for exact minimum-cost flow in both the randomized and deterministic settings.

c) Decremental graph algorithms: One of the first decremental graph algorithms was given in the 80's, when Even and Shiloach gave an algorithm to maintain decremental BFS trees [34]. Since then, there has been significant work on maintaining fundamental properties of decremental graphs. Decremental s-t or single source shortest path (SSSP) is particularly important problems that have applications such as efficient implementation of numerical methods on graphs [6], [22], [30], [51]. On undirected decremental graphs, it is known how to achieve a $(1+\epsilon)$-approximation ratio in $m^{1+o(1)}$ total time deterministically [3], [6], [23], [52]–[56]. For directed graphs, achieving an almost-linear runtime remains open and the state of the art is either $n^{2+2/3+o(1)}$ deterministically, $\widetilde{O}(m^{3/4}n^{5/4})$ assuming adaptive adversaries, or $\widetilde{O}(m^{2/3}n^{4/3})$ assuming oblivious adversaries [1], [2], [4], [29], [33], [57]. With a large $m^{o(1)}$ approximation factor, deterministic almost-linear total time is achievable for even fully-dynamic all-pair shortest path (APSP) on undirected graphs [6], [56], [58], [59].

Matchings are another graph property that have attracted significant attention from the dynamic graph algorithms community. In decremental graphs, it is possible to maintain $(1+\epsilon)$-approximate maximum (weighted or cardinality) matching on either bipartite [26], [29] or general graphs [60]–[62].

Most of the aforementioned results are purely combinatorial and are focused on maintaining discrete structures such as graph decompositions, neighborhood coverings, and search trees. On the other hand, our work maintains the solution through the lens of feasibility and optimality of a continuous optimization problem. Such idea also appears in some previous works such as incremental matchings [63], decremental matchings [26], [62], partially dynamic LPs [64], and decremental max eigenvectors [65].

d) Dynamic flow algorithms: As a direct implication of the Ford-Fulkerson's maxflow algorithm, one can maintain exact max flow on fully dynamic unweighted graphs with $O(m)$ update time (see [66] for a discussion on the matter). On planar graphs, an improved update time of $\widetilde{O}(n^{2/3})$ can be achieved [67], [68]. However, under the strongly exponential time hypothesis (SETH), there is no sublinear update time algorithm for partially dynamic general graphs [69]. As a result, much attention is devoted to maintaining approximate solutions.

In the fully dynamic case, $m^{o(1)}$-approximation ratio with $m^{o(1)}$ update time can be achieved via expander hierarchy on unit capacity, undirected graphs [10]. [35] shows how to maintain a $\widetilde{O}(1)$-approximation on capacitated graphs. Our dynamic tree cut sparsifier improves these results to $m^{o(1)}$-approximation in $m^{o(1)}$-amortized time on capacitated graphs. In the incremental case, $(1+\epsilon)$-approximate solutions can be maintained with $m^{1+o(1)}\epsilon^{-1}$ total time for undirected p-norm flows as well as directed min-cost flows [7], [9], [11]. For unweighted graphs, a runtime of $m^{3/2+o(1)}\epsilon^{-1/2}$ total time was previously achieved by [70]. The algorithm of [70] can also maintain exact max flows on incremental unweighted graphs in a $n^{5/2+o(1)}$ total time, which corresponds to a sublinear update time when the graph is sufficiently dense.

e) Lower bounds: In this paragraph, we give a more detailed discussion of related lower bounds. Since the current lower bounds do not distinguish between the incremental and decremental settings, we refer the reader to [7] for an analogous and slightly more expansive discussion in the incremental setting.

- Flows and Matchings: Under the online matrix-vector (OMv) conjecture [24], there are bipartite graphs where performing $\Theta(n^2)$ deletions and $\Theta(n)$ size queries requires $\Omega(n^{3-\delta})$ total time for any fixed constant $\delta > 0$ to maintain exact matching size [69]. Therefore, $\Omega(n^{2-\delta})$ amortized update time is necessary for $\Theta(n)$ updates and one size query under OMv. Thus, our dependence on ϵ is optimal for algorithms with sub-polynomial update time because a $(1-\frac{1}{n+1})$-approximate matching is a maximum cardinality matching.

Furthermore, under the strongly exponential time hypothesis (SETH), every decremental algorithm for the weighted and directed exact maximum flow value problem on a sparse graph with n vertices requires $O(n^{1-\delta})$ amortized update time [69].

- SCCs: We discuss the hardness of deciding if a directed graph contains a cycle in the fully-dynamic and worst-case decremental settings. In the fully-dynamic setting $\Theta(n^2)$ updates and $\Theta(n)$ cycle detection queries take $\Omega(n^{3-\delta})$ time for an arbitrary constant $\delta > 0$ under OMv [24].
  By a straightforward reduction to deciding if the s-t shortest path has length 3 or 5, there are graphs for which $\Theta(n)$ edge deletions and a single cycle detection query take total time $\Omega(n^{2-\delta})$ under OMv [24]. This rules out sub-linear worst-case update time for decremental algorithms.
- Decremental s-t Shortest Path: Under the OMv conjecture the exact decremental s-t shortest distance problem requires amortized update time $m^{0.5-\delta}$ on an unweighted graph with n vertices and $m = O(n^2)$ edges for any fixed $\delta > 0$ [24]. We remark that our dependence on ϵ is optimal for algorithms with sub-polynomial update time because a $(1+\frac{1}{n})$-approximate shortest distance on a unweighted graph is an exact shortest distance. Furthermore, algorithms with sub-polynomial worst-case update time are ruled out for obtaining a $3/5-\delta$ approximation under OMv, again via distinguishing s-t distances 3 and 5 [24].

f) Paper Organization: In Section II, we give an overview of our algorithm. Then, we describe our algorithm for maintaining a tree cut sparsifier in Section IV. In Section V, we show that tree cut sparsifers can be used to detect min-ratio cuts. Finally, in Section VI, we show that such a min-ratio cut data-structure suffices to solve decremental threshold min-cost flow.

II. Overview

To convey the workings of our algorithm, it is natural to present the sections in a top-down manner to better highlight and motivate why we need to solve certain subproblems. The later main text will give the formal proof in the bottom-up order, as our proofs build on the precise properties and guarantees of the subroutines derived before.

A. Min-Cost Flow, Transshipment and its Dual

Our algorithm for min-cost flow first reduces the min-cost flow problem to transshipment on a sparse bipartite graph $G = (V, E, \boldsymbol{c})$ with some vertex demands $\boldsymbol{d}$. The transshipment problem

$$\min_{\boldsymbol{B}^\top \boldsymbol{f} = \boldsymbol{d}, \boldsymbol{f} \geq 0} \boldsymbol{c}^\top \boldsymbol{f} \tag{2}$$

is a special case of min-cost flow where all the capacities are unbounded. Because our ultimate goal is to handle edge deletions, the reduction to this form does not address the central issue that arises for algorithms in flow space: Deleting an edge causes the current flow to no-longer route the demands. Therefore, we take the dual of (2) which translates the problem to voltage space (i.e. vertex potentials)

$$\max_{\boldsymbol{c} - \boldsymbol{B}\boldsymbol{y} \geq \boldsymbol{0}} \boldsymbol{d}^\top \boldsymbol{y}. \tag{3}$$

This form is more amenable to edge deletions, since the vertex potentials $\boldsymbol{y}$ remain feasible under edge deletions. Finally, we consider the thresholded version of (3) and simply aim to decide if $\max_{\boldsymbol{c}-\boldsymbol{B}\boldsymbol{y}\geq \boldsymbol{0}} \boldsymbol{d}^\top \boldsymbol{y} \geq F$ instead of maximizing the dual.

We summarize the technical results as the following lemma:

Lemma II.1. *Suppose there is an algorithm $\mathcal{A}$ that, given any (decremental) transshipment instance and some threshold F, outputs either a feasible flow of cost at most $F + \epsilon$ or certifies that the minimum cost is at least F after the initialization and each edge deletion in $T(n, m)$ total time. Then, there is a thresholded min-cost flow algorithm $\mathcal{A}'$ (Definition I.1) that runs in $T(O(m), O(m))$ total time. Furthermore, if $\mathcal{A}$ successes with probability p, so does $\mathcal{A}'$.*

The proof of this is deferred to the full version.

B. Solving the Thresholded Transshipment Dual via Min-Ratio Cuts

In this section, we outline how to solve the transshipment dual problem by repeatedly solving the min-ratio cut problem on a fully dynamic graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$, where $\boldsymbol{u} \in \mathbb{R}^E_{\geq 0}$ are best interpreted as edge capacities (different from the capacities in the original min-cost flow instance) and $\boldsymbol{g} \in \mathbb{R}^V \perp \boldsymbol{1}$ are referred to as vertex gradients. We denote $\boldsymbol{U} = \operatorname{diag}(\boldsymbol{u})$. Then, the min-ratio cut problem is given by

$$\min_{\boldsymbol{\Delta} \in \mathbb{R}^V} \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta} \rangle}{\| \boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta} \|_1}. \tag{4}$$

Notice, that the solution of (4) is always negative and that it therefore maximizes the absolute value of the ratio. We show that there is always a optimal solution $\boldsymbol{\Delta} = \pm \boldsymbol{1}_C$, i.e. there exist optimal $\boldsymbol{\Delta}$ which indicate cuts in the graph. Furthermore, despite being used to solve the transshipment dual on a directed graph, this problem is undirected in that only the signs of the gradients depend on the side of the cut.

To show that (4) can be used to solve the transshipment dual problem (3), we closely follow the ℓ_1-IPM framework introduced by [8] for the first almost-linear time algorithm for minimum-cost flow, adapted to dual space, and apply it to the transshipment dual. Following [8] we introduce a potential $\Phi : \mathbb{R}^V \to \mathbb{R}$

$$\Phi(\boldsymbol{y}) \stackrel{\text{def}}{=} 100m \log(F - \langle \boldsymbol{d}, \boldsymbol{y} \rangle) + \sum_{e=(u,v)\in G} (\boldsymbol{c}(e) - (\boldsymbol{B}\boldsymbol{y})(e))^{-\alpha} \tag{5}$$

for $\alpha \approx 1/\log(mC)$ where m denotes the initial number of edges in G and all costs are integers in the interval $[-C, C]$. If a solution of cost F exists, then the potential $\Phi(\boldsymbol{y})$ is unbounded and goes to $-\infty$ as $\langle \boldsymbol{d}, \boldsymbol{y} \rangle$ approaches F.

The barrier $(\cdot)^{-\alpha}$ can be thought of as the more standard $\log(\cdot)$ barrier to ensure that $\boldsymbol{y}$ remains feasible, but it penalizes approaching the boundary more harshly and thus ensures that $(\boldsymbol{c}(e) - (\boldsymbol{B}\boldsymbol{y})(e)) \geq 1/n^{\widetilde{O}(1)}$ as long as $(\boldsymbol{c}(e) - (\boldsymbol{B}\boldsymbol{y})(e))^{-\alpha} \leq \widetilde{O}(m)$. This ensures that the bit-complexity remains bounded by $\widetilde{O}(1)$, which directly follows from the following description of vertex gradients and edge capacities respectively. We let

$$\boldsymbol{g} \stackrel{\text{def}}{=} \nabla\Phi(\boldsymbol{y}) = \frac{-100m}{F - \langle \boldsymbol{d}, \boldsymbol{y}\rangle}\boldsymbol{d} + \alpha\boldsymbol{B}^\top(\boldsymbol{c} - (\boldsymbol{B}\boldsymbol{y}))^{-1-\alpha}$$

and $\boldsymbol{u}(e) \stackrel{\text{def}}{=} (\boldsymbol{c}(e) - (\boldsymbol{B}\boldsymbol{y})(e))^{-1-\alpha}$ where the $-1-\alpha$ exponent is applied to every element in the vectors separately. The Taylor-expansion

$$\begin{aligned}\Phi(\boldsymbol{y} + \boldsymbol{\Delta}) &\approx \Phi(\boldsymbol{y}) + \langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle + \|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_2^2 \\ &\leq \Phi(\boldsymbol{y}) + \langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle + \|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1^2\end{aligned}$$

implies that solving the min-ratio cut problem to $1/\kappa$ accuracy yields an update reducing the potential by approximately $1/\kappa^2$ if there is a solution to (3) with cost F because the optimum ratio is then ≈ 1.

This can be turned into an algorithm for decremental transshipment with the following observations. First, if there is a feasible $\boldsymbol{y}$ with $\langle \boldsymbol{d}, \boldsymbol{y}\rangle \geq F$, then there exists a solution to the min-ratio cut problem that decreases the potentials by at least $m^{-o(1)}$. Thus if our min-ratio cut algorithm cannot find a good solution, we conclude that $\max_{\boldsymbol{c}-\boldsymbol{B}\boldsymbol{y}\geq 0}\langle \boldsymbol{d}, \boldsymbol{y}\rangle < F$, and continue.

It is not difficult to initialize $\boldsymbol{y}$ so that the potential is initially $\widetilde{O}(m)$, and when $\Phi(\boldsymbol{y}) \leq \widetilde{O}(m)$, one can show that $\langle \boldsymbol{d}, \boldsymbol{y}\rangle \geq F - m^{-O(1)}$. Finally, edge deletions cannot increase the potential, and each edge deletion only causes $O(1)$ updates to the gradients and capacities. Overall, the algorithm only makes $m^{1+o(1)}$ calls to the dynamic min-ratio cut data structure. We refer the reader to Section VI for a detailed description of the interior point method.

C. Min-Ratio Cuts on Trees

Despite its description involving an arbitrary update vector $\boldsymbol{\Delta}$ in (4), the min-ratio cut problem always has a solution that updates along a single cut, i.e., we have

$$\min_{C\subseteq V}\frac{\langle \boldsymbol{g}, \mathbf{1}_C\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_C\|_1} = \min_{\boldsymbol{\Delta}\in\mathbb{R}^V}\frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1}$$

which explains the nomenclature and allows us to focus our efforts on cuts from here on out. We refer the reader to Lemma V.5 for a short proof of this fact.

To describe how we repeatedly solve the min-ratio cut problem approximately on a fully dynamic graph G, we assume that the problem is posed on a dynamic tree T instead. We will later reduce to this case using dynamic tree-cut sparsifiers, the main data structure we develop in this paper.

In an analogue to the cycle decomposition of circulations in flow-space, we next show by induction that it suffices to consider cuts induced by a single tree edge. Consider a min-ratio tree cut C that cuts $k > 1$ tree edges. We show that there is a cut with at most $k-1$ edges achieving at least as good quality. Because the graph is a tree there is at least one connected component C' (after removing the cut edges) that is only incident to a single cut edge. Since shifting the $\boldsymbol{\Delta} = \mathbf{1}_C$ vector does not change its ratio, we may assume that this connected component receives value 1, i.e., $\boldsymbol{\Delta}(v) = 1$ for $v \in C'$. Now notice that $\mathbf{1}_C = \mathbf{1}_{C'} + \mathbf{1}_{C\setminus C'}$ where C' is a cut induced by removing a single edge, and $C \setminus C'$ is a cut induced by removing $k-1$ edges. But then we obtain

$$\begin{aligned}&\min\left(\frac{\langle \boldsymbol{g}, \mathbf{1}_{C'}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C'}\|}, \frac{\langle \boldsymbol{g}, \mathbf{1}_{C\setminus C'}\rangle}{\|\boldsymbol{U}\boldsymbol{B}_{C\setminus C'}\|}\right) \qquad (6)\\ &\leq \frac{\langle \boldsymbol{g}, \mathbf{1}_{C'} + \mathbf{1}_{C\setminus C'}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C'}\| + \|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C\setminus C'}\|} = \frac{\langle \boldsymbol{g}, \mathbf{1}_C\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_C\|}\end{aligned}$$

where the inequality follows from the well known fact that $\min(a/b, c/d) \leq \frac{a+c}{b+d}$ given $b, d > 0$. Iterating (6) directly yields that it suffices to consider cuts induced by single tree edges.

Given this insight, it suffices to maintain the ratio achieved by every tree edge under updates to the tree, edge capacities, and gradients, where we are guaranteed that $\boldsymbol{g} \perp \mathbf{1}$ at all times. It turns out that the tree-cut sparsifiers we maintain have hop diameter bounded by $\widetilde{O}(1)$[3]. This allows us to maintain the quality of each single edge cut in $\widetilde{O}(1)$ time: whenever vertices u and v undergo an update in the form of an edge insertion, deletion, or gradient change, only edges in the path $T[u, v]$ connecting u, v have their qualities change.

We refer the reader to Section V for a detailed description of our min-ratio cut data structure on trees. This section also contains an additional component necessary to our overall algorithm. We must maintain approximations to the true gradient and capacities to know which edges to update in the dynamic min-ratio cut data structure, and this involves detecting edges which have accumulated large potential differences across the cuts we have returned. This can be done with a standard data structure.

D. Reducing to Trees via Tree-Cut Sparsifiers

In this section, we explain our construction of tree-cut sparsifiers T for a dynamic graph $G = (V, E, \boldsymbol{u})$. These are trees on a larger vertex set that capture every cut up to some multiplicative factor q. This allows us to approximate the min-ratio cut in G with a tree cut up to a multiplicative loss q.

Definition II.2 ((Tree/Forest) Cut Sparsifier)**.** *Given graph $G = (V, E, \boldsymbol{u})$, a* cut sparsifier $G' = (V', E', \boldsymbol{u}')$ *of quality q is a graph with $V \subseteq V'$ such that for every pair of disjoint sets $A, B \subseteq V$, we have that* $\text{mincut}_G(A, B) \leq \text{mincut}_{G'}(A, B) \leq q \cdot \text{mincut}_G(A, B)$. *We say that G' is a* forest cut sparsifier *if G' is a cut sparsifier and a forest graph; and we say G' is a* tree cut sparsifier *if G' is a cut sparsifier and a tree graph.*

[3]The hop-diameter of a graph is the diameter of its uncapacitated version.

At a high level, our algorithm wishes to maintain an *expander hierarchy* on a dynamic capacitated graph G, introduced by [10]. Broadly, an expander hierarchy is constructed by first finding an expander decomposition of G. In fact, a stronger notion called *boundary-linkedness* (Definition IV.12) is necessary, but this generally follows from most expander decomposition constructions. Then each expander piece is contracted, and the algorithm then finds an expander decomposition on the contracted graph, and repeats. Note that this naturally leads to a tree structure. [10] proves that this tree is a tree-cut sparsifier of quality $\widetilde{O}(1)^k/\phi$, where $k = O(\log_{1/\phi} m)$ is the number of layers in the expander hierarchy. This is $\widetilde{O}(1)^k/\phi \leq m^{o(1)}$, for $\phi = 2^{-\sqrt{\log m}}$.

The work of [10] showed how to maintain an expander hierarchy in unit capacity graphs. Our first goal is to extend this to capacitated graphs that only undergo edge *deletions*. Later, we show how to construct a tree-cut sparsifier on fully dynamic graphs by using the *core-graph* technique and batching. We start by discussing how to maintain an expander decomposition on a capacitated graph undergoing edge deletions.

a) Capacitated Decremental Expander Decomposition.: An expander decomposition is a partition $\mathcal{X}$ of the vertices in $G = (V, E, \boldsymbol{u})$, such that for every $X \in \mathcal{X}$ the induced subgraph $G[X]$ is a $\phi/\widetilde{O}(1)$ expander with respect to conductance, i.e., $\boldsymbol{u}_G(S, X \setminus S)/\min(\mathrm{vol}_G(S), \mathrm{vol}_G(X \setminus S) \geq \phi$ for all $S \subseteq X$. Futhermore, the total capacity of the crossing edges is bounded with $\widetilde{O}(\phi \cdot U^{\text{total}})$, where we denote the total capacity of all the edges in G with U^{total}.

While this problem has been studied before using more involved techniques [71], we give the simplest imaginable reduction to the uncapacitated setting. This is important for us because we require an additional property of the expander decomposition we maintain: the vertex sets of the expanders refine over time, and the total volume of all edges that are ever cut is bounded by $\widetilde{O}(\phi \cdot U^{\text{total}})$ over all edge deletions.

We first fix a value $U^{\text{cutoff}} = \phi \cdot \frac{U^{\text{total}}}{m}$, and let G' be the sub-graph of G that only contains edges with capacity at least U^{cutoff}, and additionally contains self loops of capacity $\mathrm{vol}_G(v)$ for every vertex v. Notice that computing a weighted expander decomposition for this graph G' suffices, since the same decomposition has at most ϕU^{total} extra crossing edge capacity in G, and we have $\mathrm{vol}_G(S) \leq \mathrm{vol}_{G'}(S)$ for every set $S \subseteq V$ due to the additional self loops.

We now exploit that all the non-self loop edges in G' have high capacity to replace G' by an unweighted multi-graph G''. We simply replace every edge with $\lceil \boldsymbol{u}(e)/U^{\text{cutoff}} \rceil$ uncapacitated multi-edges. Notice that the capacity of every cut in G'' 2-approximates the capacity of the cut in G' (after scaling with U^{cutoff}), and that the volume of G'' is lower bounded by the volume of G'. Furthermore G'' only contains $O(m/\phi)$ edges.

An uncapacitated decremental expander decomposition that refines over time under edge deletions can then be computed using recent works on expander decompositions, specifically [21] adapted using ideas from [20] to enable vertex splits and self-loop insertions. This *refining property* of the expander decomposition then ensures that the total amount of capacity on all edges cut at any point in time is $\widetilde{O}(\phi \cdot U^{total})$.

Overall, we have given an algorithm to maintain an expander hierarchy, and thus a tree-cut sparsifier of quality $2^{O(\sqrt{\log m} \log\log m)}$ in $m^{1+o(1)}$ time in decremental capacitated graphs.

b) Fully Dynamic Tree-Cut Sparsifiers. : Finally, we reduce from the fully dynamic case to the decremental case using batching. To describe the main ideas used in our batching scheme, we consider a current tree-cut sparsifier T of some graph G that receives a batch of insertions I. We show that we can compute a new tree-cut sparsifier of $G \cup I$ in time proportional to $|I|$ without losing too much quality. Batching the updates appropriately then turns such an algorithm into a fully dynamic tree-cut sparsifier data structure.

To compute a tree-cut sparsifier of $G \cup I$, we instead consider the graph $T \cup I$. It is not surprising that $T \cup I$ is a cut-sparsifier of $G \cup I$ given that T is a tree-cut sparsifier of G. Then, we instantiate a set of terminals $B \subseteq V(T)$. Initially, every endpoint of an edge in I is added to B. Then, B is extended to a branch-free set, i.e. a set such that the set of paths $\mathcal{P}$ containing all tree paths $T[a, b]$ for $a, b \in B$ such that it does not intersect any other terminal is edge-disjoint. This extension can be achieved by doubling the size of the terminal set.

Then, we remove the minimum capacity edge from each such path and refer to the trees in the leftover forest as cores. Thereafter, the algorithm contracts the cores (forest pieces) and computes a tree-cut sparsifier on the contracted graph merely containing the identified min-capacity edges and the inserted edges in I. This graph contains approximately $|I|$ edges, and therefore computing the tree-cut sparsifier takes time roughly proportional to $|I|$. Then, this tree is mapped back to a tree on the whole graph via un-contracting the cores.

Since the procedure described above loses an q factor in quality every time it is applied, we make sure that the sequential depth k of this operation in the final batching scheme handling insertions is very low, i.e., $q^k = \widehat{O}(1)$.

See Section IV for a full description of our tree-cut sparsifier data structure.

III. Preliminaries

a) Linear Algebra. : We denote vectors as lower case bold letters $\boldsymbol{a}$, and matrices as upper case bold letters $\boldsymbol{A}$. Given a vector $\boldsymbol{a} \in \mathbb{R}^X$ and a subset $Y \subseteq X$ we let $\boldsymbol{a}[X]$ denote the vector $\boldsymbol{a}$ restricted to the coordinates in X, and we let $\boldsymbol{a}(X) = \sum_{x \in X} \boldsymbol{a}(x)$. For a vector $\boldsymbol{u} \in \mathbb{R}^n$, we let $\mathrm{diag}(\boldsymbol{u}) \in \mathbb{R}^{n \times n}$ denote the diagonal matrix with entries of $\boldsymbol{u}$ on the diagonal.

b) Graphs.: We work with a capacitated input graph $G = (V, E, \boldsymbol{u})$ where $\boldsymbol{u}$ is the function that assigns each edge $e \in E$ a capacity $\boldsymbol{u}(e) \geq 1$. We define $\mathrm{vol}_G(v)$ for every vertex $v \in V$ as the weighted degree, i.e. $\mathrm{vol}_G(v) = \sum_{e \in E, v \in e} \boldsymbol{u}(e)$ and denote by $\deg_G(v)$ the combinatorial degree, i.e. $\deg_G(v) = \sum_{e \in E : v \in e} 1$. We extend this notion to sets where $X \subseteq V$, $\mathrm{vol}_G(X) = \sum_{v \in X} \mathrm{vol}_G(v)$ and $\deg_G(X) = \sum_{v \in X} \deg_G(v)$. For uncapacitated graphs, note that degrees and volumes coincide.

For directed graphs, we let the in-degree of vertex v be equal to the number of edges (w, v) whose head is v, and we let the out-degree of v be the number of edges (v, w) whose tail is v.

We say that a graph G is a ϕ-expander if for every $S \subseteq V$ with $\mathrm{vol}_G(S) \leq \mathrm{vol}_G(V)/2$, we have $\boldsymbol{u}(E(S, V \setminus S)) \geq \phi \cdot \mathrm{vol}_G(S)$.

Given a tree T, we denote with $T[u, v]$ the unique tree path from vertex u to vertex v.

Finally, we define the mincut between two sets of vertices in a graph G.

Definition III.1. *Given a graph $G = (V, E, \boldsymbol{u})$ and two disjoint sets $A, B \subseteq V$, we denote by $\mathrm{mincut}_G(A, B)$ the minimum value $\boldsymbol{u}(E_G(A', V \setminus A'))$ achieved by any set A' with $A \subseteq A' \subseteq V \setminus B$.*

IV. Fully-Dynamic Tree Cut Sparsifiers

The main graph-theoretic object in this paper is the notion of a tree cut sparsifier.

Definition II.2 ((Tree/Forest) Cut Sparsifier)**.** *Given graph $G = (V, E, \boldsymbol{u})$, a* cut sparsifier *$G' = (V', E', \boldsymbol{u}')$ of quality q is a graph with $V \subseteq V'$ such that for every pair of disjoint sets $A, B \subseteq V$, we have that $\mathrm{mincut}_G(A, B) \leq \mathrm{mincut}_{G'}(A, B) \leq q \cdot \mathrm{mincut}_G(A, B)$. We say that G' is a* forest cut sparsifier *if G' is a cut sparsifier and a forest graph; and we say G' is a* tree cut sparsifier *if G' is a cut sparsifier and a tree graph.*

In this section, we show that tree cut sparsifiers can be maintained efficiently in a fully-dynamic graph. Previously, this result was only known for uncapacitated graphs [10]. Our main result is summarized in Theorem IV.1.

Theorem IV.1. *Given an m-edge graph $G = (V, E, \boldsymbol{u})$ where $\boldsymbol{u} \in [1, U = m^{O(1)}]^E$. Let G be undergoing up to $\widetilde{O}(m)$ edge deletions/edge insertions and vertex splits. Then, there is a randomized algorithm that maintains a tree $T = (V', E', c')$ undergoing insertions and deletions of edges and isolated vertices, such that T is a tree cut sparsifier of quality $\gamma_q = 2^{O(\log^{3/4}(m) \log\log(m))}$ with total update time $m \cdot 2^{O(\log^{3/4}(m) \log\log(m))}$. The algorithm succeeds w.h.p.*

We further augment the above theorem to maintain a dependency graph H that allows us to track approximately which edges are in the cut induced by each tree edge of T. This graph H is crucial in our final min-cost flow algorithm as it allows us to implicitly maintain flow and potentials in the IPM.

Definition IV.2. *Given a tree cut sparsifier T of quality q, a directed layer graph $H = (V_0 \cup V_1 \cup \cdots \cup V_k, E_H)$ has k layers where V_0 has a vertex for each edge $e \in E$, and all edges $e_H \in E_H$ have their tail in V_{i+1} and head in V_i for some $0 \leq i < k$, such that every vertex $v \in V(H)$ has in-degree $d = O(\log^{c'} m)$ for some constant $c' > 0$.*

For every edge $e_T \in T$, let E_{e_T} be the set of edges in G that cross the cut induced by $T \setminus \{e_T\}$, i.e. let A, B be the connected components of $T \setminus \{e_T\}$, then $E_{e_T} = E_G(A \cap V, B \cap V)$. Let E'_{e_T} be the set of edges in G whose corresponding vertices in V_0 are reached by the vertex v_{e_T} that represents the edge e_T in the graph H. Then, we have at any time that $E_{e_T} \subseteq E'_{e_T}$ and $\boldsymbol{u}_G(E'_{e_T}) \leq q \cdot \boldsymbol{u}_G(E_{e_T})$.

Lemma IV.3. *The algorithm in Theorem IV.1 can be extended to explicitly maintain a directed layer graph $H = (V_0 \cup V_1 \cup \cdots \cup V_k, E_H)$ where $k = O(\log^{1/4}(m) \log\log(m))$.*

The additional total runtime for maintaining the graph H is again $m \cdot 2^{O(\log^{3/4}(m) \log\log(m))}$. The total number of updates to H consisting of insertions/deletions of edges and isolated vertices is bounded by $m \cdot 2^{O(\log^{3/4}(m) \log\log m)}$.

Finally, we discuss how to derandomize the above result at the cost of obtaining a slightly worse approximation guarantee and runtime.

Theorem IV.4. *Given an m-edge graph $G = (V, E, \boldsymbol{u})$ where $\boldsymbol{u} \in [1, U = m^{O(1)}]^E$. Let G be undergoing up to $\widetilde{O}(m)$ edge deletions/edge insertions and vertex splits. Then, there is a* ***deterministic*** *algorithm that maintains a tree cut sparsifier $T = (V', E', c')$ of quality $\gamma_q = 2^{O(\log^{5/6}(m) \log\log(m))}$ with total update time $m \cdot 2^{O(\log^{5/6}(m) \log\log(m))}$.*

Lemma IV.5. *The deterministic algorithm in Theorem IV.4 can be extended to explicitly maintain a directed layer graph $H = (V_0 \cup V_1 \cup \cdots \cup V_k, E_H)$ where $k = O(\log^{1/6}(m) \log\log(m))$.*

The additional total runtime for maintaining the graph H is again $m \cdot 2^{O(\log^{5/6}(m) \log\log(m))}$. The total number of updates to H consisting of insertions/deletions of edges and isolated vertices is bounded by $m \cdot 2^{O(\log^{5/6}(m) \log\log(m))}$.

Remark IV.6. *The tree cut sparsifers maintained by Theorem IV.1 and Theorem IV.4 have hop diamter $\widetilde{O}(1)$.*

For the rest of the section, we implicitly assume that all (dynamic) graphs G under consideration are connected (at all times). We obtain our main result summarized in Theorem IV.1 in three steps: first, in Section IV-A, we give a reduction that allows us to maintain a decremental expander decomposition of **capacitated** graphs by using existing techniques to maintain an expander decomposition of a decremental, **un-capacitated** graph. We then show that we can maintain a tree cut sparsifier of a decremental graph via expander decompositions in Section IV-B. Finally, we reduce the problem of maintaining a tree cut sparsifier on a dynamic graph to a decremental graph problem in Section IV-C. We then discuss how to derandomize our result to obtain Theorem IV.4 in Section IV-D.

A. Decremental Expander Decompositions for Capacitated Graphs

In this section, we generalize a recent result about the maintenance of expander decompositions to graphs with capacities. We summarize our result in Theorem IV.7 below. We point out that our proof techniques in this section can be used to obtain expander decompositions of directed, capacitated graphs, however, here we focus only on undirected graphs.

Theorem IV.7 (Capacitated Expander Decomposition)**.** *Given a parameter $0 < \phi \leq 1$ and a capacitated m-edge graph $G = (V, E, \boldsymbol{u})$, where $\boldsymbol{u} \in [1, U]^E$ and U being any positive number, undergoing a sequence of $\widetilde{O}(m)$ updates consisting of edge deletions, vertex splits and self-loop insertions.*

There is a randomized algorithm that explicitly maintains tuple $(\mathcal{X}, E^{cut})$ where $\mathcal{X}$ is a partition of the vertex set of G that refines over time and E^{cut} is a monotonically increasing set of intercluster edges with $E^{cut} \subseteq E$ such that:

1) *at any stage, for every* cluster *$X \in \mathcal{X}$, we have that the current graph $G[X]$ is a (ϕ/c_0)-expander for some fixed $c_0 = \widetilde{O}(1)$, and*
2) *at any stage, for every edge e in the current graph G, we have that if its endpoints are not in the same cluster $X \in \mathcal{X}$, then the edge is intercluster and therefore in E^{cut}, and at any time $\boldsymbol{u}(E^{cut}) \leq c_1 \cdot \phi \cdot U^{total}$ where U^{total} is the total capacity of all edges present in G at any point in time and $c_1 = \widetilde{O}(1)$.*

The algorithm takes total time $\widetilde{O}(m/\phi^3)$ and succeeds w.h.p.

To prove Theorem IV.7, we give a reduction to the uncapacitated setting and then use the following result. We point out that the theorem below generalizes the theorem in [21] as it also allows for vertex splits and self-loop insertions. This generalization can be obtained straightforwardly by combining the framework from [21] with standard techniques from [20] to deal with vertex splits and self-loop insertions.

Theorem IV.8 (Expander Decomposition [21])**.** *Given a parameter $0 < \phi \leq 1$ and an **un-capacitated** m-edge (multi-)graph $G = (V, E)$ undergoing a sequence of $\widetilde{O}(m)$ updates consisting of edge deletions, vertex splits and self-loop insertions.*

There is a randomized algorithm that explicitly maintains tuple $(\mathcal{X}, E^{cut})$ where $\mathcal{X}$ is a partition of the vertex set of G that refines over time and E^{cut} is a monotonically increasing set of intercluster edges with $E^{cut} \subseteq E$ such that:

1) *at any stage, for every* cluster *$X \in \mathcal{X}$, we have that the current graph $G[X]$ is a (ϕ/c_0)-expander for $c_0 = \widetilde{O}(1)$, and*
2) *at any stage, for every edge e in the current graph G, we have that if its endpoints are not in the same cluster $X \in \mathcal{X}$, then the edge is intercluster and therefore in E^{cut}, and at any time $|E^{cut}| \leq c_1 \cdot \phi m$ for $c_1 = \widetilde{O}(1)$.*

The algorithm takes total time $\widetilde{O}(m/\phi^2)$ and succeeds w.h.p.

a) The Algorithm.: For the proof of Theorem IV.7, we first assume that the total capacity of all edges inserted since the start of the algorithm is at most equal to the total capacity U^{total} of the initial graph. This is w.l.o.g. as otherwise the algorithm can be restarted with edges in the set E^{cut} removed from the graph and added to the new set of intercluster edges produced.[4]

[4]Because capacities are not polynomially-bounded, the number of restarts could be large, however, using the techniques introduced below, an edge can effectively be ignored if its capacity is below $\phi \cdot U^{\text{total}}/m$ and thus any edge is only considered by the algorithm during $O(\log m)$ restarts.

Then, consider the dynamic graph G' obtained from the graph G by deleting/not inserting all edges with capacity less than $\phi \cdot \frac{U^{\text{total}}}{m}$. Throughout, let G'' be the uncapacitated dynamic graph obtained from graph G' by replacing each edge e of capacity $\boldsymbol{u}(e)$ by $\lceil \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}} \phi} \rceil$ multi-edges between the same endpoints and by additionally having $\lceil \frac{m \cdot \text{vol}_G(v)}{U^{\text{total}} \phi} \rceil$ self-loops at each vertex $v \in V$.

Finally, maintain the tuple $(\mathcal{X}, E'')$ by running the algorithm from Theorem IV.8 on graph G''. Maintain the output tuple $(\mathcal{X}, E^{\text{cut}})$ to have the same partition and let E^{cut} be the union of all edges that appear at any time in $G \setminus G''$ and all edges in E such that a corresponding multi-edge is in E''.

b) Analysis.: We prove the two main properties claimed in Theorem IV.7 and then analyze the remaining properties claimed.

Claim IV.9. *The total number of edges to ever appear in G'' is at most $\widetilde{O}(m/\phi)$. Thus, the total capacity of all edges in G that become intercluster for $\mathcal{X}$ is at most $\widetilde{O}(\phi \cdot U^{total})$.*

Proof. The total capacity of all edges that ever appear in G is by assumption at most $2 \cdot U^{\text{total}}$. Since we replace each edge of capacity $\boldsymbol{u}(e)$ by $\lceil \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}} \phi} \rceil$ multi-edges, we can thus upper bound the number of such multi-edges by $\frac{m \cdot U^{\text{total}}}{U^{\text{total}} \phi} + \widetilde{O}(m) = \widetilde{O}(m/\phi)$ since we can charge each edge e its capacity $\boldsymbol{u}(e)$ and where the second term $\widetilde{O}(m)$ stems from the fact that we are rounding up $\widetilde{O}(m)$ terms.

Let us next bound the number of self-loops added to G''. We have that the total volume at all vertices is at most $4 \cdot U^{\text{total}}$ at any time by assumption, and we have that there are at most $\widetilde{O}(m)$ vertices. Thus, there are again at most $\widetilde{O}(\frac{m \cdot U^{\text{total}}}{U^{\text{total}} \cdot \phi}) + \widetilde{O}(m) = \widetilde{O}(m/\phi)$ self-loops added this way, as desired.

Finally, it suffices to observe that at most a $\widetilde{O}(\phi)$-fraction of the edges in G'' ever become intercluster for the partition $\mathcal{X}$ by Theorem IV.8. But for each edge e in the graph G', we add $\lceil \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}} \phi} \rceil$ corresponding multi-edges to G''. Thus, the total capacity of all edges in G' that becomes intercluster for $\mathcal{X}$ is at most $\widetilde{O}(\phi \cdot U^{\text{total}})$. Further, the capacity of all edges in G that do not appear in G' is at most $\widetilde{O}(m) \cdot \phi \cdot \frac{U^{\text{total}}}{m} = \widetilde{O}(\phi \cdot U^{\text{total}})$ by our construction of G'. □

Claim IV.10. *The partition $\mathcal{X}$ is such that at any time, for any $X \in \mathcal{X}$, we have that $G[X]$ is a $\widetilde{\Omega}(\phi)$-expander.*

Proof. Consider at any time, any cluster $X \in \mathcal{X}$. Let $S \subseteq X$ such that $\text{vol}_{G''[X]}(S) \leq \text{vol}_{G''[X]}(X)/2$. Then, we have from Theorem IV.8, that $|E_{G''[X]}(S, X \setminus S)| = \widetilde{\Omega}(\phi) \cdot \text{vol}_{G''[X]}(S)$.

Since we have a one-to-one correspondence between non-self-loop multi-edges e' of multiplicity a in G'' and edges e in G' such that $\lceil \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}} \phi} \rceil = a$ and since all edges in G' have capacity at least $\phi \cdot \frac{U^{\text{total}}}{m}$, we have that $\lceil \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}} \phi} \rceil = a \leq 2 \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}} \phi}$.

We further have that

$$\begin{aligned}|E_{G''[X]}(S, X \setminus S)| &= \sum_{e \in E_{G'[X]}(S, X \setminus S)} \left\lceil \frac{m \cdot \boldsymbol{u}(e)}{U^{\text{total}}\phi} \right\rceil \\ &\leq 2\boldsymbol{u}(E_{G'[X]}(S, X \setminus S)) \cdot \frac{m}{U^{\text{total}}\phi} \\ &\leq 2\boldsymbol{u}(E_{G[X]}(S, X \setminus S)) \cdot \frac{m}{U^{\text{total}}\phi}\end{aligned}$$

where we use that $G' \subseteq G$ in the last inequality. Thus, we obtain

$$\begin{aligned}\boldsymbol{u}(E_{G[X]}(S, X \setminus S)) &\geq |E_{G''[X]}(S, X \setminus S)| \cdot \frac{U^{\text{total}}\phi}{2m} \\ &= \widetilde{\Omega}(\phi) \cdot \text{vol}_{G''[X]}(S) \cdot \frac{U^{\text{total}}\phi}{2m} \\ &= \widetilde{\Omega}(\phi) \cdot \text{vol}_G(S)\end{aligned}$$

where we use in the last inequality that since each vertex $v \in V$ has at least degree $\lceil \frac{m \cdot \text{vol}(v)}{U^{\text{total}}\phi} \rceil$ in G'' and since we add self-loops, we also have that $\text{vol}_{G''[X]}(S) \geq \frac{m \cdot \text{vol}_G(S)}{U^{\text{total}}\phi}$. □

Given the two claims above, it suffices for a proof of Theorem IV.7 to observe that $\mathcal{X}$ is refining by Theorem IV.8, that E^{cut} is monotonically increasing by adding only edges that become intercluster for $\mathcal{X}$ and that the time to maintain $\mathcal{X}$ on G'' is at most $\widetilde{O}(m/\phi^3)$ by the size upper bound from Claim IV.9 on G'' and again by Theorem IV.8, and that all additional operations take time at most $\widetilde{O}(m/\phi^3)$.

B. Decremental Tree Cut Sparsifiers

In this section, we prove the following result that was previously obtained in [10] for uncapacitated graphs. Our proof follows a similar high-level strategy, however, we require more refined building blocks and arguments to obtain our result.

Theorem IV.11. *Given an m-edge graph $G = (V, E, \boldsymbol{u})$ undergoing up to $\widetilde{O}(m)$ edge deletions, vertex splits and self-loop insertions where $\boldsymbol{u} \in [1, U = m^{O(1)}]^E$.*

Then, there is a randomized algorithm that maintains a tree cut sparsifier $T = (V', E', \boldsymbol{u}')$ of G of quality $\gamma_{quality} = 2^{O(\sqrt{\log m} \log\log m)}$ such that T is a graph consisting of at most $\widetilde{O}(m)$ vertices and undergoing at most $\widetilde{O}(m)$ edge weight decreases and edge un-contractions where the latter is an update that splits a vertex v into vertices v' and v'' and inserts an edge (v', v''). The algorithm takes total time $m \cdot 2^{O(\sqrt{\log m})}$. The algorithm succeeds w.h.p.

Furthermore, the hop diameter of T is at most $O(\log m)$ throughout.

To obtain the above result, we maintain a decremental boundary-linked expander hierarchy as defined in [10].

Definition IV.12 (Dynamic Boundary-Linked Expander Decomposition)**.** *Given a dynamic graph G and parameters $\phi \in (0, 1], \beta > 0, s \geq 1$, we say that a partition $\mathcal{X}$ of the vertex set of G is an (β, ϕ, s) boundary-linked expander decomposition of G if*

1) *at any stage, for every edge e in the current graph G, we have that if its endpoints are not in the same cluster $X \in \mathcal{X}$, then the edge is intercluster and therefore in E^{cut}, and at any time $\boldsymbol{u}(E^{cut}) \leq \beta \cdot \phi \cdot U^{total}$ where U^{total} is the total capacity of all edges present in G at any point in time.*
2) *at any time, for any $X \in \mathcal{X}$, we have that the graph $G_{\mathcal{X}}^{1/(s\beta\phi)}[X]$ is ϕ-expander where we have a one-to-one correspondence between edges $e = (u, v) \in E^{cut}$ and self-loops at u and v of capacity $\frac{1}{s\beta\phi}\boldsymbol{u}(e)$. Here, $G_{\mathcal{X}}^{1/(s\beta\phi)}$ is the graph G plus self-loops of total capacity $\frac{1}{s\beta\phi} \cdot \boldsymbol{u}(E_G(v, V) \cap E^{cut})$ at each vertex $v \in V$.*

We next define the crucial concept of expander hierarchies.

Definition IV.13 (Dynamic Expander Hierarchy)**.** *Given a dynamic graph G and parameters $\phi \in (0, 1], \beta > 0$, $s \geq 1$, we define an (β, ϕ, s)-expander hierarchy recursively to consist of levels $0 \leq i \leq k$ where for each level i, we maintain a dynamic graph G_i and an (β, ϕ, s) boundary-linked expander decomposition $\mathcal{X}_i$ of G_i. We let $G_0 = G$, and for $i \geq 0$, we define G_{i+1} to be the dynamic graph obtained from G_i after contracting all vertices in the same partition set in $\mathcal{X}_i$ into a single super node and removing all self-loops. We let k be the first index such that G_k consists of only a single vertex.*

Remark IV.14. *We point out that the partitions $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_k$ can be extended to partitions of V and it is straightforward to see that the extension of $\mathcal{X}_i$ refines the extension of $\mathcal{X}_{i+1}$ and that G_{i+1} can be obtained from contracting $\mathcal{X}_i$ in G_i or from contracting the extension of $\mathcal{X}_i$ in G. We use these partitions and their extensions interchangeably when the context is clear. Further, again when the context is clear, we refer to the sets $X \in \mathcal{X}_i$ as the vertices of G_{i+1}.*

In our algorithm, for $\phi = 1/2^{\sqrt{\log m}}$, we maintain a $(2c_1, \phi/c_0, 2)$ expander hierarchy for our decremental input graph G as described in Definition IV.13 (the values c_0, c_1 are defined in Theorem IV.7). To maintain the boundary-linked expander decomposition $\mathcal{X}_i$ for each graph G_i, we simply run the algorithm from Theorem IV.7 on the graph $\widetilde{G}_i = (G_i)_{\mathcal{X}_i}^{1/(4c_1 \cdot \beta\phi)}$ where $\beta = 2c_i$ with parameter ϕ.[5] We denote by $E_{\mathcal{X}_i}^{cut}$ the set of cut edges maintained by the algorithm in Theorem IV.7 that is run on $\widetilde{G}_i$.

To obtain a tree cut sparsifier T from our dynamic expander hierarchy, we finally appeal to the following theorem. We note that the theorem below from [10] was proven only in the uncapacitated setting, however, their proof extends seamlessly.

Theorem IV.15 (see Theorem 5.2 in [10])**.** *Given a (dynamic) (β, ϕ, s)-expander hierarchy $\mathcal{H} = \{(G_0, \mathcal{X}_0), (G_1, \mathcal{X}_k), \ldots, (G_k, \mathcal{X}_k)\}$, and let $\mathcal{X}_{-1}$*

[5]Note that technically, Theorem IV.7 requires capacities to be at least 1 while some of the self-loops might be smaller. However, since correctness is not affected by scaling all capacities and all capacities in $\widetilde{G}_i$ are polynomially lower bounded in m, we can simply scale up all capacities by a large polynomial factor to increase them to be at least of size 1.

denote the partition of the vertex set of G into singleton sets. Let $T_{\mathcal{H}}$ denote the tree that has a node for each set X in any of the partitions $\mathcal{X}_i$ and if $i < k$ then the node in $T_{\mathcal{H}}$ associated with X is a child of the node $Y \in \mathcal{X}_{i+1}$ where $X \subseteq Y$ where the capacity of the edge (X, Y) in $T_{\mathcal{H}}$ is $\mathrm{vol}_{G_i}(X)$.

Then, $T_{\mathcal{H}}$ is a tree cut sparsifier of G with quality $O((s\beta)^{O(k)}/\phi)$.

From the definitions, maintenance of tree $\mathcal{T}_{\mathcal{H}}$ is straightforward. We note that to reduce the number of updates to the tree cut sparsifier T that we output, we let T be a version of $\mathcal{T}_{\mathcal{H}}$ where all edge capacities are rounded up to the nearest power of two, and enforce that all edge capacities in T are monotonically decreasing by using the smallest capacity value of an edge in $\mathcal{T}_{\mathcal{H}}$ that has been observed so far. Proving that T is still a correct tree cut sparsifier (only worse in quality by a constant factor) is trivial since by the decremental nature of G any fixed cut has monotonically decreasing capacity.

Proof of Theorem IV.11. We prove by induction on i that

1) it is valid to invoke Theorem IV.7 on the graph $\widetilde{G}_i$, i.e. $\widetilde{G}_i$ is undergoing only edge deletions, vertex splits and self-loop insertions, and
2) the total capacity on all edges in $E^{cut}_{\mathcal{X}_i}$ is at most $(2c_1\phi)^{i+1}U^{total}_G$, and
3) the number of updates to $\widetilde{G}_i$ is $\widetilde{O}(m)$.

Property 1: Since for every i, $\widetilde{G}_i$ is obtained from G_i by self-loop insertions/ deletions, we can conclude that Property 1 holds, if it holds for each graph G_i. For $i = 0$, it is vacuously true since $G_0 = G$ which is a decremental graph by assumption. For $i > 0$, we use that $\mathcal{X}_{i-1}$ is a refining partition which implies that G_i which is obtained from contracting partition sets of $\mathcal{X}_{i-1}$ in G and removing self-loops, only undergoes the deletions that G undergoes if the corresponding edge in G_i, and vertex splits for when $\mathcal{X}_{i-1}$ refines, possibly preceded by insertions of the removed self-loop at the vertex that is split in the current update.

Property 2: By Definition IV.12, we have that the total volume of all self-loops added to $\widetilde{G}_i$ that are not in G_i already is at most $\frac{1}{4c_1\phi} \cdot 2\boldsymbol{u}(E^{cut})$ (since each edge $e \in E^{cut}$ adds a self-loop of capacity $\frac{1}{4c_1\phi}\boldsymbol{u}(e)$ to both u and v). Thus, $U^{\text{total}}_{\widetilde{G}_i} \leq U^{\text{total}}_{G_i} + \frac{1}{2c_1\phi} \cdot \boldsymbol{u}(E^{cut})$ (see Item 2 in Definition IV.12).

On the other hand, since Theorem IV.7 maintains $\mathcal{X}_i$ to be an expander decomposition of $\widetilde{G}_i$ with parameter ϕ, we have that the capacity of all cut edges $E^{cut}_{\mathcal{X}_i}$ is bounded by $c_1\phi \cdot U^{\text{total}}_{\widetilde{G}_i}$.

Combining these inequalities, we obtain

$$U^{\text{total}}_{\widetilde{G}_i} \leq U^{\text{total}}_{G_i} + \frac{1}{2c_1\phi} \cdot c_1\phi \cdot U^{\text{total}}_{\widetilde{G}_i} = U^{\text{total}}_{G_i} + \frac{1}{2} \cdot U^{\text{total}}_{\widetilde{G}_i}.$$

Subtracting $\frac{1}{2} \cdot U^{\text{total}}_{\widetilde{G}_i}$ from both sides on the inequality thus yields $U^{\text{total}}_{\widetilde{G}_i} \leq 2 \cdot U^{\text{total}}_{G_i}$.

Finally, for $i = 0$, we have that $G_0 = G$ has total capacity U^{total}_G by definition, and thus the capacity of all cut edges $E^{cut}_{\mathcal{X}_0}$ is at most $c_1\phi \cdot 2U^{total}_G$. For $i > 0$, we have that G_i can only obtain edges in $E^{cut}_{\mathcal{X}_{i-1}}$ as can be seen from Definition IV.13. Thus, we have $U^{total}_{G_i} \leq \boldsymbol{u}(E^{cut}_{\mathcal{X}_{i-1}}) \leq (2c_1\phi)^i U^{total}_G$ where we used the induction hypothesis for the last inequality. This yields by Theorem IV.7 and our bound $U^{\text{total}}_{\widetilde{G}_i} \leq 2 \cdot U^{\text{total}}_{G_i}$ that $\boldsymbol{u}(E^{cut}_{\mathcal{X}_i}) \leq 2c_1\phi \cdot U^{\text{total}}_{G_i} \leq (2c_1\phi)^{i+1} \cdot U^{\text{total}}_G$, as desired.

Property 3: For $G_0 = G$, we have at most $\widetilde{O}(m)$ updates. For $i > 0$, we have that G_i undergoes at most $\widetilde{O}(m)$ updates since $\mathcal{X}_{i-1}$ is refining and thus, for G^{final} being the final graph G, causes at most $|V(G^{final})| - 1$ vertex splits and $|V(G^{final})| - 1$ self-loop insertions (to compensate for earlier removals of self-loops that go between two vertices in the graph G_i after the vertex splits) and additionally, undergoes the sequence of updates that G is undergoing if the corresponding edges are present in G_i. Thus, G_i undergoes $\widetilde{O}(m)$ updates.

Finally, we have that $\widetilde{G}_i$ undergoes 2 self-loop insertions whenever an edge is added to the set $E^{cut}_{\mathcal{X}_i}$. But since $E^{cut}_{\mathcal{X}_i}$ is monotonically increasing (see Theorem IV.7), this can cause at most twice as many updates as there are edges in G_i. Thus, $\widetilde{G}_i$ undergoes at most $\widetilde{O}(m)$ updates.

Putting it All Together: From Property 2, we can conclude that the number of levels of the hierarchy is $O(\log_{1/(2c_1\phi)}(U^{\text{total}})) = O(\sqrt{\log m}\log\log m)$ by choice of ϕ and the fact that capacities are polynomially-bounded.

Correctness of our algorithm thus follows immediately from Theorem IV.7.

Combining the bound on the number of levels of the expander hierarchy with the runtime bounds obtained by Theorem IV.7 and the bound on the number of updates to each graph $\widetilde{G}_i$ by Property 3, we obtain that the expander hierarchy can be maintained in time $\widetilde{O}(m/\phi^3)$. From Theorem IV.15, it can be observed that maintenance of tree $\mathcal{T}_{\mathcal{H}}$ and also of our modified tree T is straightforward and can be done in time $\widetilde{O}(m/\phi^3)$ as only $\widetilde{O}(1)$ operations suffice to update both trees after any update to the dynamic expander hierarchy. This yields the total runtime of our algorithm.

Finally, to bound the hop diameter of T follows immediately from the fact that $T_{\mathcal{H}}$ and thus T is a tree with k levels where $k = O(\log m)$. □

C. Fully Dynamic Tree Cut Sparsifiers

Finally, we present an algorithm to maintain a tree cut sparsifier as described in Theorem IV.1 by giving a reduction to the decremental setting.

Theorem IV.1. *Given an m-edge graph $G = (V, E, \boldsymbol{u})$ where $\boldsymbol{u} \in [1, U = m^{O(1)}]^E$. Let G be undergoing up to $\widetilde{O}(m)$ edge deletions/edge insertions and vertex splits. Then, there is a randomized algorithm that maintains a tree $T = (V', E', c')$ undergoing insertions and deletions of edges and isolated vertices, such that T is a tree cut sparsifier of quality $\gamma_q = 2^{O(\log^{3/4}(m)\log\log(m))}$ with total update time $m \cdot 2^{O(\log^{3/4}(m)\log\log(m))}$. The algorithm succeeds w.h.p.*

Definition IV.2. *Given a tree cut sparsifier T of quality q, a directed layer graph $H = (V_0 \cup V_1 \cup \cdots \cup V_k, E_H)$ has k layers where V_0 has a vertex for each edge $e \in E$, and all edges $e_H \in E_H$ have their tail in V_{i+1} and head in V_i*

for some $0 \leq i < k$, such that every vertex $v \in V(H)$ has in-degree $d = O(\log^{c'} m)$ for some constant $c' > 0$.

For every edge $e_T \in T$, let E_{e_T} be the set of edges in G that cross the cut induced by $T \setminus \{e_T\}$, i.e. let A, B be the connected components of $T \setminus \{e_T\}$, then $E_{e_T} = E_G(A \cap V, B \cap V)$. Let E'_{e_T} be the set of edges in G whose corresponding vertices in V_0 are reached by the vertex v_{e_T} that represents the edge e_T in the graph H. Then, we have at any time that $E_{e_T} \subseteq E'_{e_T}$ and $\boldsymbol{u}_G(E'_{e_T}) \leq q \cdot \boldsymbol{u}_G(E_{e_T})$.

Lemma IV.3. *The algorithm in Theorem IV.1 can be extended to explicitly maintain a directed layer graph $H = (V_0 \cup V_1 \cup \cdots \cup V_k, E_H)$ where $k = O(\log^{1/4}(m) \log\log(m))$.*

The additional total runtime for maintaining the graph H is again $m \cdot 2^{O(\log^{3/4}(m) \log\log(m))}$. The total number of updates to H consisting of insertions/deletions of edges and isolated vertices is bounded by $m \cdot 2^{O(\log^{3/4}(m) \log\log m)}$.

a) Core Graphs.: Before we describe our reduction, let us introduce the concept of core graphs which have been crucial in the design of recent dynamic graph algorithms.

Definition IV.16 (Core graph)**.** *Given a graph $G = (V, E, \boldsymbol{u})$, a rooted forest F (i.e. each component of F has a dedicated root vertex) with $V(F) \supseteq V$. We define the* core graph *$\mathcal{C}(G, F)$ to be the graph obtained from graph G by contracting the vertices of every connected component in F into a super-vertex that is then identified with the root vertex of the corresponding tree in F, i.e. the vertex set of $\mathcal{C}(G, F)$ is the set of roots of F. We let the capacities of edges in $\mathcal{C}(G, F)$ be equal to their capacities in G.*

In our algorithm, we use induced core graphs. For the definition, we also need to define the notion of a branch-free set.

Definition IV.17 (Branch-Free Set)**.** *Given a tree $T = (V, E, \boldsymbol{u})$, we say that $B \subseteq V$ is a branch-free set for T if we have that $\mathcal{P}_{T,B}$, the collection of all paths $T[a, b]$ for $a, b \in B$ that contain no internal vertex in B, consists of pairwise edge-disjoint paths.*

Definition IV.18 (Induced Core Graph)**.** *Given a graph $G = (V, E, \boldsymbol{u})$, a tree T with $V \subseteq V(T)$, and a set of roots $B \subseteq V$ such that B is a branch-free set for T. We let $F(T, B)$ denote the rooted forest obtained by removing from the tree T the lexicographically-first edge e_P of minimum capacity from each path $P \in \mathcal{P}_{T,B}$. Note, that this yields a forest $F(T, B)$ where each connected component contains exactly one node in B. We let the corresponding vertex in B be the root of its component to make $F(T, B)$ a rooted forest. We define the* induced core graph *$\mathcal{C}(G, T, B)$ to be the core graph $\mathcal{C}(G, F(T, B))$.*

Finally, we state the following algorithmic result that extends any set R to a branch-free set B that is not much larger. To unambiguously define the result, here, we require the notion of a monotonically increasing vertex set for a graph undergoing vertex splits.

Definition IV.19 (Monotonically Increasing Set in Graph Undergoing Vertex Splits)**.** *Given a graph $G = (V, E, \boldsymbol{u})$ undergoing a sequence of vertex splits. Whenever a vertex v is split into vertices v' and v'', we say that v' and v'' descend from v and we further extend this notion to be transitively closed, i.e. if v' is further split into v''' and v'''', then v''' and v'''' also descend from v, and so on. Then, we say that a set $X \subseteq V$ is monotonically increasing if for any two time steps $t' < t$, every v in X at time t' has a descendent in X at time t.*

The following standard result is then obtained via link-cut trees [72] (See e.g. [8]).

Theorem IV.20. *Given an m-vertex tree graph $T = (V, E, \boldsymbol{u})$ undergoing $\widetilde{O}(m)$ edge un-contractions, i.e. updates that split a vertex v into vertices v' and v'' and add an edge (v', v''), and a monotonically increasing set $R \subseteq V$. Then, there is a deterministic algorithm that maintains a monotonically increasing set B such that at any time, $R \subseteq B$, B is branch-free for the current tree T, and $|B| \leq 2|R|$. The algorithm outputs B explicitly after every update to T or R, and runs in total time $\widetilde{O}(m)$.*

b) A Hierarchy of Tree Cut Sparsifiers.: We are now ready to describe our reduction (see also Figure 1). Let $\widetilde{m} = \widetilde{O}(m)$ be a strict upper bound on the number of updates to G. Our algorithm maintains levels $0, 1, \ldots, L_{max} = \lceil \log^{1/4}(\widetilde{m}) \rceil$. We use a simple batching approach over the update sequence where we associate with each level $i \in [0, L_{max}]$, at current time t, an associated time $t_i = \lfloor t / \widetilde{m}^{(L_{max}-i)/L_{max}} \rfloor \cdot \widetilde{m}^{(L_{max}-i)/L_{max}}$ at which level i was last re-built.[6]

We further maintain with each level $i \in [0, L_{max}]$, a batch I_i consisting of all edges in the current graph G that were inserted after time t_i (note in particular that edges added and deleted after time t_i are not in I_i). We define $G_i = G \setminus I_i$ for all $0 \leq i \leq L_{max}$. We note in particular that $I_{L_{max}} = \emptyset$ since $t_{L_{max}} = t$ and therefore $G_{L_{max}} = G$.

For each level $i \in [0, L_{max}]$, our goal is to maintain a tree cut sparsifier T_i of the current graph G_i, thus in particular, $T_{L_{max}}$ is a tree cut sparsifier of the current graph G. For $i = 0$, we let T_0 be the tree cut sparsifier obtained by running the data structure from Theorem IV.11 on the graph $G_0 = G \setminus I_0$, that is, the initial graph where only decremental updates are applied. For $i > 0$, we let B_i be the monotonically increasing set obtained by running the algorithm in Theorem IV.20 on the tree T_{i-1} for vertices $V(I_{i-1} \setminus I_i)$ since time t_i. Let $\widehat{T}_i$ be the tree obtained from running the data structure in Theorem IV.11 on the graph $\widehat{G}_i = \mathcal{C}(T_{i-1} \cup (I_{i-1} \setminus I_i), T_{i-1}, B_i)$ (as defined in Definition IV.18) since time t_i. Then, we maintain $T_i = 2 \cdot (F(T_{i-1}, B_i) \cup \widehat{T}_i)$. Note here in particular that we are not adding the pre-images of edges in $\widehat{T}_i$ to T_i but instead the real edges in $\widehat{T}_i$ which are supported on B_i only.

As previously mentioned, we output the tree $T_{L_{max}}$ as our tree cut sparsifier T of G.

[6] We assume here that $\widetilde{m}^{(L_{max}-i)/L_{max}}$ is integer which is w.l.o.g.

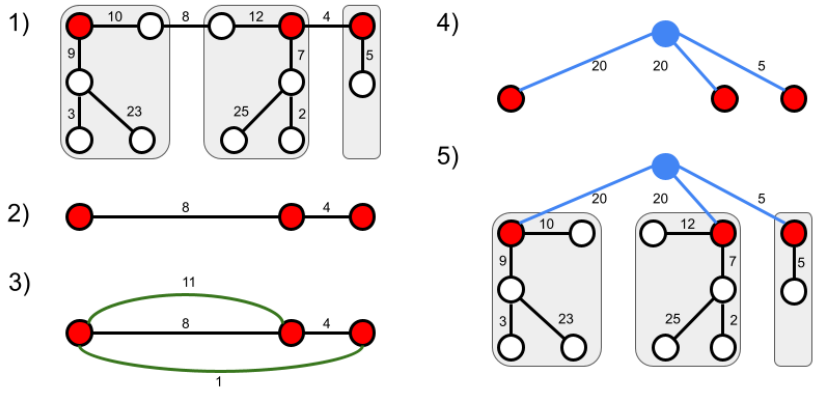

Fig. 1: 1) shows a tree cut sparsifier T_{i-1} (for a graph G_{i-1}). Red vertices are the vertices in B_i. The grey components show the connected components of $F(T_{i-1}, B_i)$, edges crossing such components are of minimum capacity on a path in $\mathcal{P}_{T_{i-1},B_i}$.
2) shows the induced core graph $\mathcal{C}(T_{i-1}, T_{i-1}, B_i)$.
3) shows the induced core graph $\widehat{G}_i = \mathcal{C}(T_{i-1} \cup (I_{i-1} \setminus I_i), T_{i-1}, B_i)$, i.e., the previous graph with all edges that are in G_i but not in G_{i-1} (in green).
4) shows a tree cut sparsifier $\widehat{T}_i$ of the graph $\widehat{G}_i$.
5) shows the final tree cut sparsifier T_i of G_i which is formed by the union of $F(T_{i-1}, B_i)$ and the tree cut sparsifier $\widehat{T}_i$ of the induced core graph $\widehat{G}_i$.

c) Analysis.: We first establish correctness of the algorithm.

Claim IV.21 (Correctness)**.** *The graph T is a tree cut sparsifier of G of quality $2^{O(\log^{3/4}(m)\log\log(m))}$ at all times.*

Proof. We prove by induction on i that T_i is a tree cut sparsifier of G_i of quality $q_i = (2\gamma_{quality})^{i+1}$. For $i = 0$, we have, by definition of $\widetilde{m}$, that all inserted edges since the start of the algorithm are in I_0. Thus, the data structure from Theorem IV.11 maintains T_0 correctly to be a tree cut sparsifier of G_0 of quality $\gamma_{quality}$.

For $i > 0$, we have by the induction hypothesis that T_{i-1} is a tree cut sparsifier of G_{i-1} of quality q_{i-1}. It is straightforward from the definition of G_i to see that $G_i = G \setminus I_i = (G \setminus I_{i-1}) \cup (I_{i-1} \setminus I_i) = G_{i-1} \cup (I_{i-1} \setminus I_i)$ since $I_{i-1} \supseteq I_i$.

Thus, it is straightforward to verify that by the induction hypothesis, we have that $T_{i-1} \cup (I_{i-1} \setminus I_i)$ is a cut sparsifier of $G_i = G_{i-1} \cup (I_{i-1} \setminus I_i)$ of quality q_{i-1}. Finally, consider the graph T_i as maintained by the hierarchy. To see that $T_i = 2 \cdot (F(T_{i-1}, B_i) \cup \widehat{T}_i)$ is a tree, we use the standard fact that the union of a tree in a graph contracted along a forest and the forest itself yields a tree spanning the original graph. It remains to verify the quality of T_i w.r.t. G_i.

$\text{min-cut}_{G_i}(A, B) \leq \text{min-cut}_{T_i}(A, B)$: Let us consider any disjoint sets $A, B \subseteq V(G_i) = V$. By sub-modularity of graph cuts in G_i, it suffices to focus on the special case that the AB-min-cut in T_i consists only of a single edge e. To show this by induction on the number of cut tree edges, we first extend the AB-min-cut to a realization $(A', V_{T_i} \setminus A')$ in T_i. Then, we assume that the claimed inequality holds for cuts involving at most k tree edges. Consider a cut involving $k+1$ tree edges. Remove one of the $k+1$ cut edges such that the remaining k edges are in the same tree component. Then, let A'_1 be the cut induced by the remaining k edges, and A'_2 be the cut induced by the removed edge such that $A' \subseteq A'_1$ and $A' \subseteq A'_2$. Then, we have

$$\begin{aligned}\text{min-cut}_{G_i}(A', V \setminus A') &= \text{min-cut}_{G_i}(A'_1 \cap A'_2, V \setminus (A'_1 \cap A'_2)) \\ &\leq \text{min-cut}_{G_i}(A'_1, V \setminus A'_1) \\ &\quad + \text{min-cut}_{G_i}(A'_2, V \setminus A'_2) \\ &\leq \text{min-cut}_{T_i}(A'_1 \cap A'_2, V \setminus (A'_1 \cap A'_2))\end{aligned}$$

where the first inequality is by sub-modularity of cuts, and the second follows from the induction hypothesis. We then distinguish by cases:

- if $e \in F(T_{i-1}, B_i)$: We now give a formal proof of this case and discuss an example of such a proof in Figure 2. Since $F(T_{i-1}, B_i)$ is a forest where each component contains exactly one vertex on B_i (see Definition IV.18), we have that $F(T_{i-1}, B_i) \setminus \{e\}$ contains a single connected component A' that contains no vertex in B_i. Further note that since T_i consists of $F(T_{i-1}, B_i)$ and edges supported only on B_i, we have that also $T_i \setminus \{e\}$ contains A' as one of its connected components. Thus, by assumption either

$A \subseteq A' \subseteq V(T_i) \setminus B$, or $B \subseteq A' \subseteq V(T_i) \setminus A$. Let us assume for the rest of the proof that $A' \supseteq A$ (the case where $A' \supseteq B$ is analogous).

Our key claim is that for any edge $e_1, e_2, \ldots, e_k$ in $E_{T_{i-1}}(A', V(T_{i-1}) \setminus A') \setminus \{e\}$ the path $P_j \in \mathcal{P}_{T_{i-1},B_i}$ that contains edge e_j also contains edge e. Note that this implies that $k \leq 1$, since all paths in $\mathcal{P}_{T_{i-1},B_i}$ are edge-disjoint. To see the claim, observe that for every path $P_j \in \mathcal{P}_{T_{i-1},B_i}$ there is only a single edge on P_j removed from T_{i-1} to obtain $F(T_{i-1}, B_i)$ and thus if two edges e_j and e_ℓ for some $\ell \neq j$ appear on P_j then one of them would still be in the cut $E_{F(T_{i-1},B_i)}(A', V(T_{i-1}) \setminus A')$ which contradicts that the latter set consists only of the edge e. But since all paths $P_1, P_2, \ldots, P_k$ must be distinct, each path P_j enters the component A' via edge e_j. But all paths in $\mathcal{P}_{T_{i-1},B_i}$ start and end in a vertex in B_i. Since $A' \cap B_i = \emptyset$, the path P_j must therefore use the edge e to reach a vertex in B_i as it is the only edge to leave A' that is not already on another path.

We observe that if $E_{T_{i-1}}(A', V(T_{i-1}) \setminus A') \setminus \{e\} = \emptyset$, then trivially

$$\begin{aligned}\boldsymbol{u}_{T_{i-1}}(e) &= \boldsymbol{u}_{T_{i-1}}(A', V(T_{i-1}) \setminus A') \\ &= \boldsymbol{u}_{F(T_{i-1},B_i)}(A', V(T_{i-1}) \setminus A').\end{aligned}$$

If it contains an additional edge e_1, then we have that the path P_1, defined as above, contains the edge e. But we have from Definition IV.18 that removing e_1 instead of e from T_{i-1} to obtain $F(T_{i-1}, B_i)$ implies $\boldsymbol{u}_{T_{i-1}}(e_1) \leq \boldsymbol{u}_{T_{i-1}}(e)$. And thus, we have in this case, $\boldsymbol{u}_{T_{i-1}}(A', V(T_{i-1}) \setminus A') = \boldsymbol{u}_{T_{i-1}}(e) + \boldsymbol{u}_{T_{i-1}}(e_1) \leq 2 \cdot \boldsymbol{u}_{T_{i-1}}(e) = 2 \cdot \boldsymbol{u}_{F(T_{i-1},B_i)}(A', V(T_{i-1}) \setminus A')$. We can thus finally use the induction hypothesis on T_{i-1} to obtain that $2\boldsymbol{u}_{F(T_{i-1},B_i)}(A', V(T_{i-1}) \setminus A') \geq \boldsymbol{u}_{G_{i-1}}(A', V \setminus A')$ from which we can conclude

$$\begin{aligned}&\boldsymbol{u}_{T_i}(A', V(T_{i-1}) \setminus A') \\ &= 2 \cdot \boldsymbol{u}_{F(T_{i-1},B_i) \cup \widehat{T}_i}(A', V(T_i) \setminus A') \\ &\geq 2 \cdot \boldsymbol{u}_{F(T_{i-1},B_i)}(A', V(T_i) \setminus A') \\ &\geq \boldsymbol{u}_{G_{i-1}}(A, V \setminus A) \\ &= \boldsymbol{u}_{G_i}(A, V \setminus A)\end{aligned}$$

where the last equality follows since no edge from $G_i \setminus G_{i-1}$ is incident to A (since $A \cap B_i \subseteq A' \cap B_i = \emptyset$).

- otherwise: in this case, we have $e \in \widehat{T}_i$. Let $A'' \supseteq A$ and $B'' \supseteq B$ be the connected components of $T_i \setminus \{e\}$. Let $A' = A'' \cap V(T_{i-1})$ and $B' = B'' \cap V(T_{i-1})$. We clearly have that $A \subseteq A' \subseteq A''$ and $B \subseteq B' \subseteq B''$ because $V \subseteq V(T_{i-1})$ by induction on T_{i-1}. Observe further that A, B partition V; A', B' partition $V(T_{i-1})$; and A'', B'' partition $V(T_i)$.

  Next, let $\widehat{A}, \widehat{B}$ be the connected components of $\widehat{T}_i \setminus \{e\}$ such that $\widehat{A} \subseteq A'', \widehat{B} \subseteq B''$. Then, we have by Theorem IV.11, that $\boldsymbol{u}_{\widehat{T}_i}(\widehat{A}, \widehat{B}) \geq \boldsymbol{u}_{\widehat{G}_i}(\widehat{A} \cap B_i, \widehat{B} \cap B_i)$ where we use that $B_i = V(\widehat{G}_i)$. By construction, we have that $\boldsymbol{u}_{\widehat{G}_i}(\widehat{A} \cap B_i, \widehat{B} \cap B_i)) = \boldsymbol{u}_{\mathcal{C}(T_{i-1} \cup (I_{i-1} \setminus I_i), T_{i-1}, B_i)}(\widehat{A} \cap B_i, \widehat{B} \cap B_i) = \boldsymbol{u}_{T_{i-1} \cup (I_{i-1} \setminus I_i)}(A', B') = \boldsymbol{u}_{T_{i-1}}(A', B') + \boldsymbol{u}_{I_{i-1} \setminus I_i}(A', B')$. By induction, we have that $\boldsymbol{u}_{T_{i-1}}(A', B') \geq \boldsymbol{u}_{G_{i-1}}(A, B)$ and since $I_{i-1} \setminus I_i \subseteq G_i$, we have that $\boldsymbol{u}_{I_{i-1} \setminus I_i}(A', B') = \boldsymbol{u}_{I_{i-1} \setminus I_i}(A, B)$.

  It remains to use that $T_i \supseteq 2 \cdot \widehat{T}_i$ and to combine inequalities which yields $\boldsymbol{u}_{T_i}(\widehat{A}, \widehat{B}) \geq 2\boldsymbol{u}_{\widehat{T}_i}(\widehat{A}, \widehat{B}) \geq 2\boldsymbol{u}_{G_{i-1}}(A, B) + 2\boldsymbol{u}_{I_{i-1} \setminus I_i}(A, B)$.

min-cut$_{T_i}(A, B) \leq q_i \cdot$ min-cut$_{G_i}(A, B)$: For this claim, note that it suffices to prove for all sets $A \subseteq V$ that min-cut$_{T_i}(A, B) \leq q_i \cdot \boldsymbol{u}_{G_i}(A, B)$ for $B = V \setminus A$.

Let us fix such a cut $(A, B = V \setminus A)$ in G_i. We have for $T_i = 2 \cdot (F(T_{i-1}, B_i) \cup \widehat{T}_i)$ that since $F(T_{i-1}, B_i) \subseteq T_{i-1}$ and $G_{i-1} \subseteq G_i$, we have that min-cut$_{F(T_{i-1},B_i)}(A, B) \leq q_{i-1} \cdot \boldsymbol{u}_{G_i}(A, B)$.

It thus only remains to obtain an upper bound on min-cut$_{\widehat{T}_i}(A, B)$. Since all edges in $\widehat{T}_i$ are either incident to vertices in B_i or to newly created vertices (not in V), we have that min-cut$_{\widehat{T}_i}(A, B) =$ min-cut$_{\widehat{T}_i}(A \cap B_i, B \cap B_i)$. We have from Theorem IV.11 that min-cut$_{\widehat{T}_i}(A \cap B_i, B \cap B_i) \leq \gamma_{quality} \cdot \boldsymbol{u}_{\widehat{G}_i}(A \cap B_i, B \cap B_i)$. But for every edge e in

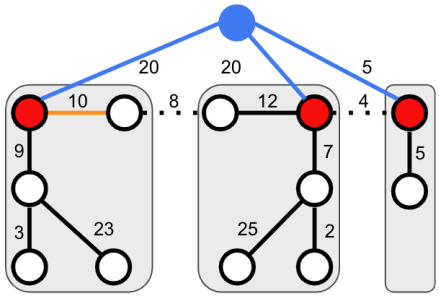

Fig. 2: Consider the example from Figure 1 where edges in T_{i-1} that are not in T_i are dashed, red vertices are the vertices of B_i and blue edges and vertices and the vertices of B_i form $\widehat{T}_i$.

Let us argue for the cut induced by the orange edge e in $F(T_{i-1}, B_i)$, where $T_i \setminus \{e\}$ is the AB-min-cut of T_i. We have that $F(T_{i-1}, B_i) \setminus \{e\}$ contains a connected component A' that has no vertex in B_i since it contains exactly one more connected component than vertices in B_i. This component A' is also a connected component of $T_i \setminus \{e\}$ since no edges of $\widehat{T}_i$ are incident to A' and $T_i = F(T_{i-1}, B_i) \cup \widehat{T}_i$.

We prove that either, we are in a case where T_{i-1} has no edge leaving A' other than e in which case we obtain a rather straightforward lower bound on the cut size, or, at most one such edge e_1 (in our case the dotted edge incident to the orange edge). But in this case, we have that e and e_1 are on a common path $P \in \mathcal{P}_{T_{i-1},B_i}$ and since from each such path only the edge of smallest capacity is removed, we have $\boldsymbol{u}_{T_{i-1}}(e_1) \geq \boldsymbol{u}_{T_{i-1}}(e)$. Thus, we can again bound the capacity of the cut $(A', V(T_i) \setminus A')$ by $2\boldsymbol{u}(e)$.

$\widehat{G}_i$, we either have that $e \in I_{i-1} \setminus I_i$ and thus the edge is also present in G_i with the same quality. Or, there is a path $P_e \in \mathcal{P}_{T_{i-1},B_i}$ between the two endpoints of e in T_{i-1} where each edge on the path has capacity at least $\boldsymbol{u}_{\widehat{G}_i}(e)$. Since all of these paths P_e are edge-disjoint, we have that for every edge $e \in E_{\widehat{G}_i}(A \cap B_i, B \cap B_i)$, we have at least one edge in $E_{T_{i-1}}(A \cap B_i, B \cap B_i)$ on P_e that has no less capacity than $\boldsymbol{u}_{\widehat{G}_i}(e)$. Thus, min-cut$_{\widehat{G}_i}(A \cap B_i, B \cap B_i) \leq \boldsymbol{u}_{G_i}(A,B) + \boldsymbol{u}_{T_{i-1}}(A,B)$. Combining these inequalities, we obtain that

$$\begin{aligned}\text{min-cut}_{\widehat{T}_i}(A,B) &\leq \gamma_{quality}(\boldsymbol{u}_{G_i}(A,B) + \boldsymbol{u}_{T_{i-1}}(A,B)) \\ &\leq \gamma_{quality}(\boldsymbol{u}_{G_i}(A,B) + q_{i-1} \cdot \boldsymbol{u}_{G_{i-1}}(A,B)) \\ &\leq \gamma_{quality} \cdot (q_{i-1}+1) \cdot \boldsymbol{u}_{G_i}(A,B)\end{aligned}$$

where we used in the second inequality the induction hypothesis on T_{i-1} and in the final inequality that $G_{i-1} \subseteq G_i$.

This yields that min-cut$_{T_i}(A,B) \leq (\gamma_{quality}+1) \cdot (q_{i-1}+1) \cdot \boldsymbol{u}_{G_i}(A,B)$ where we have that $(\gamma_{quality}+1) \cdot (q_{i-1}+1) < 2\gamma_{quality} \cdot q_{i-1} = q_i$, as required. □

It now only remains to bound the runtime.

Claim IV.22 (Runtime)**.** *The algorithm has amortized update time* $2^{O(\log^{3/4}(m)\log\log(m))}$.

Proof. The set I_{i-1} contains at most $t - t_{i-1}$ edges at any time which can be bounded by $\widetilde{m}_{i-1}$ where $\widetilde{m}_j := \widetilde{m}^{(L_{\max}-j)/L_{max}}$ for $j = 0, \dots L_{\max}$ by definition of t_{i-1}.

But this implies that for level i, the set $V(I_{i-1} \setminus I_i)$ is of size at most $2\widetilde{m}_{i-1}$, and thus the set B_i is of size at most $4\widetilde{m}_{i-1}$ (see Theorem IV.20). Since each graph $\widehat{G}_i$ consists only of a forest supported on the vertices in B_i, and the images of edges in $I_{i-1} \setminus I_i$ under the contractions (see Definition IV.18), we can bound the number of edges in $\widehat{G}_i$ at any time by $4\widetilde{m}_{i-1} - 1 + \widetilde{m}_{i-1} \leq 5\widetilde{m}_{i-1}$.

Thus, the runtime of the decremental tree cut sparsifier run on the graph $\widehat{G}_i$ has total update time $2^{O(\sqrt{\log \widetilde{m}_{i-1}})} \cdot \widetilde{m}_{i-1}$ in-between rebuilds by Theorem IV.11. Since $\widehat{G}_i$ gets re-built after $\widetilde{m}_i$ updates and there are $\widetilde{m}$ updates in total, the total time spend by all decremental tree cut sparsifier data structures for $\widehat{G}_i$ is at most $\widetilde{m}_{i-1} \cdot 2^{O(\sqrt{\log \widetilde{m}_{i-1}})} \cdot \frac{\widetilde{m}}{\widetilde{m}_i} = 2^{O(\sqrt{\log \widetilde{m}_{i-1}})} \cdot \widetilde{m}^{1+1/L_{max}} = 2^{O(\log^{3/4} \widetilde{m}) \cdot \widetilde{m}}$ since $L_{\max} = \left\lceil \log^{3/4} m \right\rceil$. The time to implement the remaining operations of our algorithm is asymptotically subsumed by this bound. □

d) Extending Theorem IV.1 to Maintain Cut Edges.: Finally, we will prove Lemma IV.3 and Lemma IV.5.

To this end, we maintain the layer graph L with levels $i \in [0,k]$ where we choose $k = L_{max}+1$. We define $\widehat{G}_0 = G$ and $\widehat{T}_0 = T_0$, and recall the definition of $\widehat{G}_i$ for $i > 0$ to be $\widehat{G}_i = \mathcal{C}(T_{i-1} \cup (I_{i-1} \setminus I_i), T_{i-1}, B_i)$. From the runtime analysis, we have that every graph $\widehat{G}_i$ undergoes at most $m \cdot 2^{\log^{3/4}(m)\log\log(m)}$ updates over the entire course of the algorithm.

Now, we maintain for the graph L the vertex set V_i at level i to be in one-to-one correspondence with the edges of $\widehat{G}_i$. We then add for every edge $e = (u,v) \in E(\widehat{G}_i)$ and edge $e' \in \widehat{T}_i[u,v]$, an edge from the vertex $v_{e'} \in V_{i+1}$ (in one-to-one correspondence with e') to the vertex $v_e \in V_i$ (in one-to-one correspondence with e) to the graph L.

For the analysis, let us first observe that the edges of $\widehat{G}_0$ are exactly the edges in G and thus V_0 is in one-to-one correspondence with the edges in G as required. We further use that the hop diameter of every tree $\widehat{T}_i$ is at most $O(\log m)$ by Theorem IV.11. This implies that the out-degree of every vertex in L is at most $O(\log m)$. Finally, we observe that each graph $\widehat{G}_i$ undergoes at most $m \cdot 2^{\log^{3/4}(m)\log\log(m)}$ updates which follows trivially from our runtime analysis and the fact that each such graph is maintained explicitly. It remains to observe that once an edge $e \in E(\widehat{G}_i)$ is embedded into an edge $e' \in \widehat{T}_i$, it remains embedded until the end of the algorithm or until e is deleted. And since each edge e' that e embeds into can be detected in constant time:

- if e is newly inserted, then it suffices to walk towards the root of $\widehat{T}_i$ (which we can root arbitrarily for this purpose such that vertices in $\widehat{G}_i$) from both endpoints of e to detect all edges that e currently embeds into. By walking in parallel and aborting once the two explorations meet, this operation can be implemented in constant time per detected edge, or
- if $\widehat{T}_i$ is undergoing an un-contraction (see Theorem IV.11), then for the newly created edge e', it suffices to copy the set of edges embedded into e'', where e'' is the edge that is incident to e' and closer to the root of $\widehat{T}_i$.

This yields both runtime and recourse bounds for the maintenance of graph L, as required.

e) Bounding the hop-diameter. : Since the decremental trees have depth $O(\log n)$, composing $L_{max}+1 = O(\log^{1/4} m)$ such trees as described above yields a tree of depth $\widetilde{O}(1)$. This proves Remark IV.6.

D. A Deterministic Algorithm to Maintain Fully Dynamic Tree Cut Sparsifiers

We finally describe how to derandomize our result. We first note that all algorithmic reductions presented in this paper are already deterministic. Thus, the only algorithm that uses randomization in our data structure above is the algorithm from Theorem IV.8. We note that a deterministic version of Theorem IV.8 was already given in [20] with only subpolynomially worse runtime and approximation guarantees. Here, however, we describe how to derandomize Theorem IV.8 more directly to obtain sligthly better subpolynomial factors. We note that both [20], [21] work even in the directed setting while we describe a derandomization for the undirected setting only.

The algorithm from Theorem IV.8 given in [21] in turn is also deterministic except for $\widetilde{O}(1/\phi)$ invocations of the static cut-matching game algorithm from [73] (the algorithm from [21] in fact uses a generalization of [73] to directed graphs, however, since we only work with undirected graphs,

using [73] in their algorithm is sufficent for our purposes). This algorithm obtains approximation guarantees of $\widetilde{O}(1)$ and runtime $\widetilde{O}(m/\phi)$ on a graph with m edges. Recently, this algorithm was derandomized in [74], however, the authors obtained a slightly weaker result: their approximation guarantee is $e^{O(\log^{1/3}(m)\log\log m)}$ while their runtime is $m \cdot e^{O(\log^{2/3}(m)\log\log m)}/\phi^2$ (this is implied in particular by Theorem 5.3 in [75], the second ArXiv version of [74]).

Using this algorithm internally in the framework from [21], we obtain a deterministic algorithm implementing Theorem IV.8 with $c_0 = e^{O(\log^{1/3}(m)\log\log m)}$, $c_1 = e^{O(\log^{1/3}(m)\log\log m)}$ and total update time $m \cdot e^{O(\log^{2/3}(m)\log\log m)}/\phi^3$. We thus obtain a deterministic version of Theorem IV.7 with the same values for c_0 and c_1 and total update time $m \cdot e^{O(\log^{2/3}(m)\log\log m)}/\phi^4$.

By carefully re-parameterizing the algorithm in Section IV-B to use $\phi = 1/2^{\log^{2/3}(m)}$, we obtain that the number of levels of the expander hierarchy can be bounded by $O(\log^{1/3}(m)\log\log(m))$. We thus recover a quality of the final tree cut sparsifier of $\gamma_{quality} = 2^{O(\log^{2/3}(m)\log\log(m))}$ and a runtime of $m \cdot e^{O(\log^{2/3}(m)\log\log m)}$. The bound on the hop diameter of the tree cut sparsifier T is again $O(\log m)$.

Finally, we use our deterministic algorithm to maintain a tree cut sparsifier of a decremental graph in lieu of the randomized algorithm, and re-parametrizing the algorithm in Section IV-C to only use $L_{max} = \lceil \log^{1/6}(\widetilde{m}) \rceil$ levels. This yields Theorem IV.4.

V. Dynamic Min-Ratio Cut

In this section, we build a data structure that allows us to toggle along approximate min-ratio cuts in a fully dynamic graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$. Unlike the previous section, every vertex v now has an extra associated value: a gradient $\boldsymbol{g}(v) \in \mathbb{R}$. When we compute a tree-cut sparsifier on such a graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$ it simply ignores these vertex gradients. We first define the min-ratio cut problem.

Definition V.1 (Min-Ratio Cut)**.** *For a graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$, we refer to $\min_{\boldsymbol{\Delta}\in\mathbb{R}^V} \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1}$ as the min-ratio cut problem.*

Then, we define the main data structure this section is concerned with. Together with the interior point method in Section VI, this data structure is what enables us to solve decremental threshold min-cost flow.

Definition V.2 (Min-Ratio Cut Data Structure)**.** *For a dynamic graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$ where $\boldsymbol{u} \in \mathbb{R}^E$ and $\boldsymbol{g} \in \mathbb{R}^V$, $\boldsymbol{g} \perp \mathbf{1}$, such that $\boldsymbol{u}(e) \in [1, U]$ and $\boldsymbol{g}(v) \in [-U, U]$ for $\log U = \widetilde{O}(1)$, an initial potential vector $\boldsymbol{y} \in R^V$, and a detection threshold parameter ϵ, a α-approximate min-ratio cut data structure $\mathcal{D}$ supports the following operations.*

- InsertEdge(e), DeleteEdge(e)*: Inserts/deletes edge e to/from G with capacity $\boldsymbol{u}(e)$.*
- UpdateGradient(u, v, δ)*: Updates $\boldsymbol{g}(u) = \boldsymbol{g}(u) + \delta$ and $\boldsymbol{g}(v) = \boldsymbol{g}(v) - \delta$.*
- InsertVertex(v)*: Inserts isolated vertex v to G.*
- Potential(v)*: Returns $\boldsymbol{y}(v)$.*

After every update the data structure $\mathcal{D}$ the data structure returns a tuple (g, u) where $g \in \mathbb{R}_{\leq 0}$ and $u \in \mathbb{R}_{\geq 0}$ such that for some implicit cut $\mathbf{1}_C$ for $C \subseteq V$ we have $\langle \boldsymbol{g}, \mathbf{1}_C\rangle = g$ and $\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_C\|_1 \leq u$, and

$$\frac{g}{u} \leq \frac{1}{\alpha} \min_{\boldsymbol{\Delta}\in\mathbb{R}^V} \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1}.$$

The data structure additionally allows updates of the following type based on the most recently returned tuple (g, u).

- ToggleCut(η)*: Given a parameter $\eta \leq 1/u$, the data structure implicitly updates $\boldsymbol{y}$ with $\boldsymbol{y}^{(new)}$ such that $\boldsymbol{B}\boldsymbol{y}^{(new)} = \boldsymbol{B}\boldsymbol{y} + \eta\boldsymbol{B}\mathbf{1}_C$.*
  Then, the data structure returns some edge set E' such that every edge $e = (u, v)$ for which $\boldsymbol{u}(e)(\boldsymbol{y}(u) - \boldsymbol{y}(v))$ has changed by at least ϵ since it was inserted/last returned in E'.

We then state two separate theorems showing that there is both a randomized and a deterministic algorithm implementing a min-ratio cut data structure.

Theorem V.3. *There is a randomized min-ratio cut data structure (Definition V.2) given a graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$ and ϵ for $\alpha = 2^{O(\log^{3/4} m \log\log m)}$ such that every update/query is processed in amortized time $2^{O(\log^{3/4} m \log\log m)} \log U$. Furthermore, the total number of edges returned by the algorithm after t calls to* ToggleCut *is at most $2^{O(\log^{1/4}\log\log m)} \cdot t/\epsilon$. The algorithm works against an adaptive adversary and succeeds with high probability.*

Theorem V.4. *There is a deterministic min-ratio cut data structure (Definition V.2) given a graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$ and ϵ for $\alpha = 2^{O(\log^{5/6} m \log\log m)}$ such that every update/query is processed in amortized time $2^{O(\log^{5/6} m \log\log m)} \log U$. Furthermore, the total number of edges returned by the algorithm after t calls to* ToggleCut() *is at most $2^{O(\log^{1/6}\log\log m)} \cdot t/\epsilon$.*

A. Toggling Min-Ratio Cuts on a Tree Cut Sparsifier

Before we prove Theorem V.3 and the deterministic version Theorem V.4, we show that a cut is a solution to the min-ratio cut problem, which explains the nomenclature.

Lemma V.5. $\min_{C\subset V} \frac{\langle \boldsymbol{g}, \mathbf{1}_C\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_C\|_1} = \min_{\boldsymbol{\Delta}\in\mathbb{R}^V} \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1}$

Proof. First observe that since the right hand side is a minimum over all $\boldsymbol{\Delta}$, thus the minimum objective value achieved must be negative. Further, considering $\boldsymbol{\Delta} = \mathbf{1}_C$ for all $C \subseteq V$ gives us that the right hand side is less than or equal to left hand side. To establish that left hand side is less than equal to right hand side, we consider a vector $\boldsymbol{\Delta}^\star$ minimizing the right hand side. Without loss of generality, we assume that the minimum entry of $\boldsymbol{\Delta}^\star$ is 0 and the maximum entry is 1 by shifting and scaling.

Let t be a random variable uniformly distributed on $[0, 1]$. Let C_t denote the set $\{v \in V | \boldsymbol{\Delta}^\star(v) > t\}$. Observe

that $\mathbb{E}_t[\mathbf{1}_{C_t}] = \boldsymbol{\Delta}^\star$. By linearity of expectations, we have $\mathbb{E}_t[\langle \boldsymbol{g}, \mathbf{1}_{C_t}\rangle] = \langle \boldsymbol{g}, \boldsymbol{\Delta}^\star\rangle$.

Moreover, $\mathbb{E}_t[\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C_t}\|_1] = \sum_e \boldsymbol{u}_e \mathbb{E}_t\left[\left|\chi_e^\top \mathbf{1}_{C_t}\right|\right]$. Observe that for each edge e, $\chi_e^\top \mathbf{1}_{C_t}$ has the same sign for all t. Thus, $\mathbb{E}_t\left[\left|\chi_e^\top \mathbf{1}_{C_t}\right|\right] = \left|\mathbb{E}_t\left[\chi_e^\top \mathbf{1}_{C_t}\right]\right| = \left|\chi_e^\top \boldsymbol{\Delta}^\star\right|$, and hence $\mathbb{E}_t[\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C_t}\|_1] = \|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}^\star\|_1$. Thus, we have,

$$\frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}^\star\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}^\star\|_1} = \frac{\mathbb{E}_t[\langle \boldsymbol{g}, \mathbf{1}_{C_t}\rangle]}{\mathbb{E}_t[\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C_t}\|_1]}.$$

Hence there exists a t where $\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C_t}\|_1 \neq 0$ and $\frac{\langle \boldsymbol{g}, \mathbf{1}_{C_t}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C_t}\|_1} \leq \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}^\star\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}^\star\|_1}$ by the well known fact that $\min_{i\in[n]} \boldsymbol{a}(i)/\boldsymbol{b}(i) \leq \sum_{i=1}^n \boldsymbol{a}(i)/\sum_{i=1}^n \boldsymbol{b}(i)$ for $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n_{>0}$. This concludes our proof. □

Next, we show that given a tree-cut sparsifier T of G, there exists a tree cut that corresponds to an approximate min-ratio cut.

Lemma V.6. *Given a tree cut sparsifier $T = (V(T), E(T), \boldsymbol{u}_T)$ of quality q of a graph $G = (V, E, \boldsymbol{u}, \boldsymbol{g})$ there exists an edge $e_T \in E(T)$ that induces a cut $(C, V(T) \setminus C)$ such that*

$$\frac{\langle \boldsymbol{g}, \mathbf{1}_{C\cap V}\rangle}{\boldsymbol{u}_T(e_T)} \leq \frac{1}{q} \min_{\boldsymbol{\Delta}\in\mathbb{R}^V} \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1}$$

Proof. By the well known fact that $\min_{i\in[n]} \boldsymbol{a}(i)/\boldsymbol{b}(i) \leq \sum_{i=1}^n \boldsymbol{a}(i)/\sum_{i=1}^n \boldsymbol{b}(i)$ for $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n_{>0}$ it suffices to consider cuts in a single connected component. We therefore without loss of generality assume that G and thus T are connected.

By Lemma V.5 there exists some cut C' such that $\frac{\langle \boldsymbol{g}, \mathbf{1}_{C'}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C'}\|_1} = \min_{\boldsymbol{\Delta}\in\mathbb{R}^V} \frac{\langle \boldsymbol{g}, \boldsymbol{\Delta}\rangle}{\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{\Delta}\|_1}$. Then, by Definition II.2 the cut $\mathrm{mincut}_T(C', V\setminus C') \leq q \cdot \|\boldsymbol{U}\boldsymbol{B}\mathbf{1}_{C'}\|_1$, and therefore this cut achieves the min-ratio up to a factor of $\frac{1}{q}$. We next show that the quality of this tree cut can be realized by an individual cut edge in T. To do so we arbitrarily root the tree at some vertex r, where we denote $\boldsymbol{y} = \mathbf{1}_{C'}$ and assume $\boldsymbol{y}(r) = 0$. Notice that such a vertex always exists.

For every edge $e = (u, v)$ where u is the parent of v, we then set $\boldsymbol{a}(e) = \boldsymbol{y}(v) - \boldsymbol{y}(u)$. Notice that $\boldsymbol{a}(e) \in \{-1, 0, 1\}$ and $|\boldsymbol{a}(e)| = 1$ if and only if the edge e is in the cut found by $\boldsymbol{y}$. Furthermore, we let $\boldsymbol{s}_e$ denote the indicator vector of all the vertices in the sub-tree rooted at v. We next show that $\widehat{\boldsymbol{y}} \stackrel{\text{def}}{=} \sum_{e\in T} \boldsymbol{a}(e)\boldsymbol{s}_e = \boldsymbol{y}$. Let u be an arbitrary vertex. Then

$$\begin{aligned}\widehat{\boldsymbol{y}}(u) &= \sum_{e\in T} \boldsymbol{a}(e)\boldsymbol{s}_e(u)\\ &= \sum_{e\in T[u,r]} \boldsymbol{a}(e)\boldsymbol{s}_e(u)\\ &= \sum_{e\in T[u,r]} \boldsymbol{a}(e) = \sum_{(v,w)\in T[u,r]} \boldsymbol{y}(v) - \boldsymbol{y}(w) = \boldsymbol{y}(u)\end{aligned}$$

where the first equality is by definition, the second follows from the fact that sub-trees not containing u do not affect its value in $\widehat{\boldsymbol{y}}$, the third follows since $\boldsymbol{s}_e(u) = 1$ for ever edge on the path $T[u, r]$, the forth follows by definition of $\boldsymbol{a}(e)$ and the final equality follows from $\boldsymbol{y}(r) = 0$ after cancellation.

Therefore we have $\langle \boldsymbol{g}, \boldsymbol{y}\rangle = \sum_{e\in T} \boldsymbol{a}(e)\langle \boldsymbol{g}, \boldsymbol{s}_e\rangle$ and $\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{y}\|_1 = \sum_{e\in T} |\boldsymbol{a}(e)|\,\|\boldsymbol{U}\boldsymbol{B}\boldsymbol{s}_e\|_1$, because $\boldsymbol{B}\boldsymbol{s}_e$ are indicators of single edges. The result then again follows from the well known fact that $\min_{i\in[n]} \boldsymbol{a}(i)/\boldsymbol{b}(i) \leq \sum_{i=1}^n \boldsymbol{a}(i)/\sum_{i=1}^n \boldsymbol{b}(i)$ for $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n_{>0}$. □

As a final ingredient, we need a data structure that detects when potential difference may have changed significantly.

Definition V.7 (Detection Algorithm)**.** *Given a fully dynamic tree cut sparsifier T and a corresponding fully dynamic directed layer graph H as in Theorem IV.1 and Lemma IV.3 respectively, and an edge significance function $s : E \mapsto \mathbb{R}_{>0}$, a γ-approximate detection algorithm supports the following operations.*

- AddDelta(e_T, δ)*: Given an edge $e_T \in E(T)$ and a value $\delta \in \mathbb{R}_{>0}$, it adds δ to the accumulated change of each edge e in E'_{e_T}, i.e. to each edge reachable from v_{e_T} in H. Then, it reports a set E' of edges such that*
  - *Every reported edge $e \in E'$ has accumulated a change of at least $s(e)/\gamma$.*
  - *Every edge e that has accumulated a change of at least $s(e)$ is in E'.*

  Then the accumulated change of the edges in E' is re-set to 0.
- Reset(e)*: Resets the accumulated change of edge e to 0.*

Furthermore, we let D be the total number of detected edges throughout the course of the algorithm, C be the number of updates to H and R be the total number of calls to Reset()*.*

We next state the main theorem for our detection algorithm.

Theorem V.8. *There exists a γ-approximate deterministic detection algorithm (Definition V.7) for $\gamma = d^k$ with total update time $\widetilde{O}(d^k(D + R + C))$. Recall that d is a bound on the in-degree of H, and k is a bound on the depth of H (See Definition IV.2).*

The proof of Theorem V.8 is deferred to the full version of the paper.

a) Proof of Theorem V.3 and Theorem V.4: We first prove Theorem V.3 using the slightly faster randomized three cut sparsifiers, and then proceed with the analogous proof of Theorem V.4 using deterministic tree cut sparsifiers.

Since the tree cut sparsifiers require the capacities to be polynomially bounded, our algorithm internally maintains data structures for $\log(U)$ different levels. We first describe the objects the data structure maintains at level $i = 0, \ldots, \log(U)-1$.

- Rounded Graph: We let $G_i = (V, E, \boldsymbol{u}_i)$ be the graph G with altered capacities

$$\boldsymbol{u}_i(e) = \begin{cases} \lceil \boldsymbol{u}(e)/n^i \rceil & \text{if } \lceil \boldsymbol{u}(e)/n^i \rceil \leq n^{10} \\ n^{20} & \text{otherwise} \end{cases}$$

- Tree Cut Sparsifer: We maintain a tree cut sparsifer T_i of G_i with quality $q = 2^{O(\log^{3/4} m \log\log m)}$ and the stated

update time in Theorem IV.1 (See Theorem IV.4 for the deterministic version).

- Min-Ratio Cut: We maintain the ratio achieved by every tree cut and keep them in a sorted list according to minimum ratio/quality, where we discard every cut that has capacity larger than n^{20}. Notice that this effectively contracts edges of capacity n^{20}, and does not affect cuts that do not contain such edges since other edges have capcity at most n^{10}, and therefore no cuts only involving such edges can reach capacity n^{20}.
  To explicitly maintain the ratio a tree cut achieves we directly have access to the capacities, and therefore only need to worry about maintaining the gradient sums. To do so, we use that the depth of T_i is bounded by $\widetilde{O}(1)$. We then maintain the value $\boldsymbol{g}(v)$ at every vertex v, and maintain the sum of these values on either side of each edge in the tree T_i (vertices that are not in G_i but in T_i contribute 0). These two values are the sum of the gradients of crossing edges with opposite sign. Whenever a gradient between two vertices gets updated, only two vertices are affected in their $\boldsymbol{g}(v)$ value and can be updated explicitly. Since the vertices change by the same amount with opposite sign, only tree cuts that place them in different components are relevant. Therefore, only the values stored on the edges on the tree path between the two endpoints need to be updated. Notice that there are only $\widetilde{O}(1)$ such edges by the depth bound on the tree cut sparsifiers (See Remark IV.6). The value of all the edges on the path between the endpoints of the edge for which the gradient was updated can be updated accordingly.
  Finally, whenever a tree edge (u, v) in the tree that contains vertices from G on either side is updated, we simulate it as moving the two endpoints separately from the old tree edge to the new tree edge. Then after moving a single endpoint, say v, only the edges on the path between the new endpoint and the old endpoint need to be updated with the sum of the values stored for the component containing u after removing edge (u, v). This value is readily available on edge e. The update is then analogous to the case where a gradient is updated.
  Finally, new additional vertices and edges to them can be inserted and always store 0 since they do not contain crossing edges.
- Detection Algorithm: Furthermore, every level i initializes a detection algorithm to detect whenever the quantity $\boldsymbol{u}(u, v)(\boldsymbol{y}(u)-\boldsymbol{y}(v))$ has changed by an additive ϵ. It will ignore cancellations in-between calls to ToggleCut() and track the difference $\boldsymbol{y}(u) - \boldsymbol{y}(v)$ while looking for changes of the size $\epsilon/\boldsymbol{u}(e)$ instead, which is equivalent. To this avail, it initializes a detection algorithm $\mathcal{D}_i$ (Definition V.7 and Theorem V.8) using the directed layer graph associated with T_i. Every level sets the significance of edge e to $s(e) = \epsilon/(\boldsymbol{u}(e) \log U)$, such that the change summed up over the levels is still bounded by $\epsilon/\boldsymbol{u}(e)$.

After each update the algorithm goes through the best min-ratio tree cut found at each level and scales the quality of the best cut at level i by dividing it by n^i. Then, it outputs the gradient sum g of this cut, and the cut estimate $u = n^i \cdot \boldsymbol{u}_{T_i}(e_{T_i})$ where e_{T_i} is the edge that induces the cut.

We next show that the best quality such tree cut across all levels is α competitive for $\alpha = 2^{\widetilde{O}(\log^{3/4} m \log\log m)}$. To do so, we fix a cut C optimal for the min-ratio cycle problem which exists by Lemma V.5. Consider the edge e' in $E(C, V \setminus C)$ with highest capacity. Let i be the smallest index such that $\lceil \boldsymbol{u}(e)/n^i \rceil \leq n^{10}$. If that i is 0, then the cut C is captured by the tree-cut sparsifier T_0 up to a factor of $2q$, and therefore one if its tree cuts is a $2q$ approximate min-ratio cut by Lemma V.6. For higher i, the cut is again approximately preserved, because capacities with $\boldsymbol{u}(e) \geq n^i$ are correct up to a factor 2, and capacities $\boldsymbol{u}(e) \leq n^i$ contribute less than $\boldsymbol{u}(e')$ altogether since there are at most n^2 such capacities. Therefore, the tree T_i again captures the cut up to a $2q$ factor (after re-scaling), and therefore one if its re-scaled tree cuts is $2q$ approximately min-ratio by Lemma V.6.

The potential vector $\boldsymbol{y}$ after updates can be tracked via a link-cut tree on T_i [72] and queries can be supported by querying the potential change on each tree and adding them up.

It remains to show that we can detect changes in potential difference of the right magnitude when they happen without returning too many edges. We can update the detection thresholds of the set E'_{e_T} from the directed layer graph H (Definition IV.2) via the routine AddDelta(e_T, η) of the detection algorithm. Then the detection algorithm clearly reports all edges that have changed by the required margin, because the set E'_{e_T} is a super-set of the edges the tree cut actually cuts and it does not factor in cancellations that happen across calls.

We finally bound the total number of returned edges. Since the potential change of an edge can only be detected whenever it has accumulated $\epsilon\boldsymbol{u}(e)/(\gamma \log(U))$ change in potential difference, and the total amount of change per update is $|E'|\cdot\eta \leq |E'|/\boldsymbol{u}_T(e_T) \leq |E'|/(\sum_{e\in E'_{e_T}} \boldsymbol{u}(e))$ we have that at most $(\gamma \cdot t \log U)/\epsilon$ edges get reported after processing t updates.

The runtime guarantee follows from Theorem V.8 and Theorem IV.1.

We finally remark that the completely analogous proof using deterministic tree cut sparsifiers (See Theorem IV.4) yields Theorem V.4.

VI. L1-IPM on the Dual

WLOG, we may consider the following uncapacitated transshipment problem:

$$\min_{\boldsymbol{B}^\top \boldsymbol{f} = \boldsymbol{d}, \boldsymbol{f} \geq 0} \langle \boldsymbol{c}, \boldsymbol{f} \rangle \tag{7}$$

Our main IPM result is summarized as follows:

Theorem VI.1 (Dual L1 IPM)**.** *Consider a decremental uncapacitated min-cost flow instance (7), a cost threshold F, and an approximation parameter $\kappa = m^{o(1)}$. There is a*

potential reduction framework for the dual problem that runs in $\widetilde{O}(m\kappa^2)$ iterations in total.

We start with a feasible dual $\boldsymbol{y}^{(0)}$ such that $\Phi(\boldsymbol{y}^{(0)}) = \widetilde{O}(m)$ and edge slacks $\widetilde{\boldsymbol{s}} \stackrel{\text{def}}{=} \boldsymbol{s}(\boldsymbol{y}^{(0)})$ and residual $\widetilde{r} \stackrel{\text{def}}{=} r(\boldsymbol{y}^{(0)})$. At each iteration, the following happens:

1) *We receive updates in $U \subseteq E$ to $\widetilde{\boldsymbol{s}}$ so that*

$$\widetilde{\boldsymbol{s}} \approx_{1+1/10\kappa} \boldsymbol{s}(\boldsymbol{y}^{(t)}) \tag{8}$$

   We also update $\widetilde{\boldsymbol{r}}$ so that $\widetilde{r} \approx_{1+1/10\kappa} r(\boldsymbol{y}^{(t)})$.

2) *Compute a κ-approximate min-ratio cut $\boldsymbol{\Delta} \in \mathbb{R}^V$, i.e., an κ-approximate solution to the problem:*

$$\min_{\boldsymbol{\Delta} \in \mathbb{R}^V} \frac{\langle \widetilde{\boldsymbol{g}}, \boldsymbol{\Delta} \rangle}{\left\| \widetilde{\boldsymbol{U}} \boldsymbol{B} \boldsymbol{\Delta} \right\|_1}$$

   where $\widetilde{\boldsymbol{g}} = \boldsymbol{g}(\widetilde{\boldsymbol{s}}, \widetilde{r})$ and $\widetilde{\boldsymbol{u}} = \boldsymbol{u}(\widetilde{\boldsymbol{s}})$. If the ratio is larger than $-\widetilde{O}(1/\kappa)$, we certify that the optimal value to (7) is less than F.

3) *Scale $\boldsymbol{\Delta}$ so that $\langle \widetilde{\boldsymbol{g}}, \boldsymbol{\Delta} \rangle = -1/(100\kappa^2)$ and update $\boldsymbol{y}^{(t+1)} \leftarrow \boldsymbol{y}^{(t)} + \boldsymbol{\Delta}$.*

After $\widetilde{O}(m\kappa^2)$ iterations, we have $\langle \boldsymbol{d}, \boldsymbol{y} \rangle \geq F - (mC)^{-10}$.

Over the course of the algorithm, the slacks $\boldsymbol{s}(\boldsymbol{y}^{(t)})$ stay quasi-polynomially bounded. That is, $\boldsymbol{s}(\boldsymbol{y}^{(t)})(e) \in [2^{-O(\log^2(mC))}, (mC)^{O(1)}]$ for any edge e at any iteration t.

Its proof is deferred to the full version.

REFERENCES

[1] M. Henzinger, S. Krinninger, and D. Nanongkai, "Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs," in *Symposium on Theory of Computing, STOC 2014*, pp. 674–683, ACM, 2014.

[2] M. Henzinger, S. Krinninger, and D. Nanongkai, "Improved algorithms for decremental single-source reachability on directed graphs," in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, vol. 9134 of *Lecture Notes in Computer Science*, pp. 725–736, Springer, 2015.

[3] M. Henzinger, S. Krinninger, and D. Nanongkai, "Decremental single-source shortest paths on undirected graphs in near-linear total update time," *J. ACM*, vol. 65, no. 6, pp. 36:1–36:40, 2018.

[4] A. Bernstein, M. Probst Gutenberg, and C. Wulff-Nilsen, "Near-optimal decremental SSSP in dense weighted digraphs," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pp. 1112–1122, IEEE, 2020.

[5] A. Bernstein, M. Probst, and C. Wulff-Nilsen, "Decremental strongly-connected components and single-source reachability in near-linear time," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 365–376, ACM, 2019.

[6] A. Bernstein, M. Probst Gutenberg, and T. Saranurak, "Deterministic decremental sssp and approximate min-cost flow in almost-linear time," in *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1000–1008, IEEE, 2022.

[7] L. Chen, R. Kyng, Y. P. Liu, S. Meierhans, and M. Probst Gutenberg, "Almost-linear time algorithms for incremental graphs: Cycle detection, sccs,s-t shortest path, and minimum-cost flow," *arXiv preprint arXiv:2311.18295*, 2023.

[8] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. Probst Gutenberg, and S. Sachdeva, "Maximum flow and minimum-cost flow in almost-linear time," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 612–623, IEEE, 2022.

[9] J. v. d. Brand, Y. P. Liu, and A. Sidford, "Dynamic maxflow via dynamic interior point methods," in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, (New York, NY, USA), p. 1215–1228, Association for Computing Machinery, 2023.

[10] G. Goranci, H. Räcke, T. Saranurak, and Z. Tan, "The expander hierarchy and its applications to dynamic graph algorithms," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2212–2228, SIAM, 2021.

[11] J. v. d. Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. Probst Gutenberg, S. Sachdeva, and A. Sidford, "Incremental approximate maximum flow on undirected graphs in subpolynomial update time," in *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2980–2998, SIAM, 2024.

[12] J. v. d. Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. Probst Gutenberg, S. Sachdeva, and A. Sidford, "A deterministic almost-linear time algorithm for minimum-cost flow," in *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pp. 503–514, IEEE, 2023.

[13] S. Forster and G. Goranci, "Dynamic low-stretch trees via dynamic low-diameter decompositions," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pp. 377–388, 2019.

[14] S. Chechik and T. Zhang, "Dynamic low-stretch spanning trees in subpolynomial time," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 463–475, SIAM, 2020.

[15] S. Forster, G. Goranci, and M. Henzinger, "Dynamic maintenance of low-stretch probabilistic tree embeddings with applications," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1226–1245, SIAM, 2021.

[16] A. Mądry, "Fast approximation algorithms for cut-based problems in undirected graphs," in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pp. 245–254, IEEE Computer Society, 2010.

[17] J. Sherman, "Nearly maximum flows in nearly linear time," in *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 263–269, IEEE Computer Society, 2013.

[18] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, "An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 217–226, SIAM, 2014.

[19] V. Rozhoň, C. Grunau, B. Haeupler, G. Zuzic, and J. Li, "Undirected (1+ eps)-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 478–487, 2022.

[20] Y. Hua, R. Kyng, M. Probst Gutenberg, and Z. Wu, "Maintaining expander decompositions via sparse cuts," in *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 48–69, SIAM, 2023.

[21] M. Probst Gutenberg and A. Sulser, "A simple and near-optimal algorithm for directed expander decompositions," 2024.

[22] A. Mądry, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pp. 121–130, ACM, 2010.

[23] R. Kyng, S. Meierhans, and M. Probst Gutenberg, "A Dynamic Shortest Paths Toolbox: Low-Congestion Vertex Sparsifiers and their Applications," Nov. 2023. Available at https://arxiv.org/abs/2311.06402.

[24] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *STOC*, pp. 21–30, ACM, 2015.

[25] S. Bhattacharya, P. Kiss, and T. Saranurak, "Dynamic algorithms for packing-covering lps via multiplicative weight updates," in *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pp. 1–47, SIAM, 2023.

[26] A. Jambulapati, Y. Jin, A. Sidford, and K. Tian, "Regularized box-simplex games and dynamic decremental bipartite matching," in *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, vol. 229 of *LIPIcs*, pp. 77:1–77:20, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[27] S. Chechik, T. D. Hansen, G. F. Italiano, J. Lacki, and N. Parotsidis, "Decremental single-source reachability and strongly connected components in $\widetilde{O}(m\sqrt{n})$ total update time," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pp. 315–324, IEEE Computer Society, 2016.

[28] G. F. Italiano, A. Karczmarz, J. Lacki, and P. Sankowski, "Decremental single-source reachability in planar digraphs," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pp. 1108–1121, ACM, 2017.

[29] A. Bernstein, M. P. Gutenberg, and T. 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)*, pp. 1123–1134, IEEE, 2020.

[30] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," *SIAM J. Comput.*, vol. 37, no. 2, pp. 630–652, 2007.

[31] J. Chuzhoy and S. Khanna, "A faster combinatorial algorithm for maximum bipartite matching," in *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2185–2235, SIAM, 2024.

[32] J. Chuzhoy and S. Khanna, "Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm," *STOC 2024*, 2024. Accepted to STOC 2024.

[33] M. Probst Gutenberg and C. Wulff-Nilsen, "Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pp. 2542–2561, SIAM, 2020.

[34] Y. Shiloach and S. Even, "An on-line edge-deletion problem," *Journal of the ACM (JACM)*, vol. 28, no. 1, pp. 1–4, 1981.

[35] L. Chen, G. Goranci, M. Henzinger, R. Peng, and T. Saranurak, "Fast dynamic cuts, distances and effective resistances via vertex sparsifiers," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1135–1146, IEEE, 2020.

[36] G. Goranci, M. Henzinger, and T. Saranurak, "Fast dynamic flows, cuts, distances via vertex sparsifers," 2019.

[37] M. Henzinger, B. Jin, R. Peng, and D. P. Williamson, "A combinatorial cut-toggling algorithm for solving laplacian linear systems," in *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, vol. 251 of *LIPIcs*, pp. 69:1–69:22, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[38] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pp. 911–920, ACM, 2013.

[39] G. Zuzic, "A simple boosting framework for transshipment," in *31st Annual European Symposium on Algorithms (ESA 2023)*, Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[40] J. Li, "Faster parallel algorithm for approximate short-

est path," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pp. 308–321, ACM, 2020.

[41] J. Blikstad, J. v. d. Brand, Y. Efron, S. Mukhopadhyay, and D. Nanongkai, "Nearly optimal communication and query complexity of bipartite matching," in *FOCS*, pp. 1174–1185, IEEE, 2022.

[42] P. F. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S. Teng, "Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs," in *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pp. 273–282, ACM, 2011.

[43] A. Mądry, "Navigating central path with electrical flows: From flows to matchings, and back," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pp. 253–262, IEEE Computer Society, 2013.

[44] Y. T. Lee and A. Sidford, "Path finding methods for linear programming: Solving linear programs in $\widetilde{O}(\sqrt{\mathrm{rank}})$ iterations and faster algorithms for maximum flow," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pp. 424–433, IEEE Computer Society, 2014.

[45] A. Mądry, "Computing maximum flow with augmenting electrical flows," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pp. 593–602, IEEE Computer Society, 2016.

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

[47] J. v. d. Brand, Y.-T. Lee, D. Nanongkai, R. Peng, T. Saranurak, A. Sidford, Z. Song, and D. Wang, "Bipartite matching in nearly-linear time on moderately dense graphs," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 919–930, IEEE, 2020.

[48] J. v. d. Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang, "Minimum cost flows, mdps, and ℓ1-regression in nearly linear time for dense instances," in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 859–869, 2021.

[49] Y. Gao, Y. P. Liu, and R. Peng, "Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao," in *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pp. 516–527, IEEE, 2021.

[50] J. v. d. Brand, Y. Gao, A. Jambulapati, Y. T. Lee, Y. P. Liu, R. Peng, and A. Sidford, "Faster maxflow via improved dynamic spectral vertex sparsifiers," in *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, 2022*, pp. 543–556, ACM, 2022.

[51] L. Chen, R. Kyng, M. P. Gutenberg, and S. Sachdeva, "A simple framework for finding balanced sparse cuts via apsp," in *Symposium on Simplicity in Algorithms (SOSA)*, pp. 42–55, SIAM, 2023.

[52] A. Bernstein and S. Chechik, "Deterministic decremental single source shortest paths: beyond the o (mn) bound," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 389–397, 2016.

[53] A. Bernstein and S. Chechik, "Deterministic partially dynamic single source shortest paths for sparse graphs," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 453–469, SIAM, 2017.

[54] J. Chuzhoy and S. Khanna, "A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pp. 389–400, 2019.

[55] M. Probst Gutenberg and C. Wulff-Nilsen, "Deterministic algorithms for decremental approximate shortest paths: Faster and simpler," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2522–2541, SIAM, 2020.

[56] J. Chuzhoy and T. Saranurak, "Deterministic algorithms for decremental shortest paths via layered core decomposition," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pp. 2478–2496, SIAM, 2021.

[57] A. Bernstein, "Maintaining shortest paths under deletions in weighted directed graphs," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pp. 725–734, 2013.

[58] S. Chechik, "Near-optimal approximate decremental all pairs shortest paths," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 170–181, IEEE, 2018.

[59] J. Chuzhoy, "Decremental all-pairs shortest paths in deterministic near-linear time," in *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pp. 626–639, ACM, 2021.

[60] S. Assadi, A. Bernstein, and A. Dudeja, "Decremental matching in general graphs," *arXiv preprint arXiv:2207.00927*, 2022.

[61] A. Dudeja, "Decremental matching in general weighted graphs," *arXiv preprint arXiv:2312.08996*, 2023.

[62] J. Chen, A. Sidford, and T.-W. Tu, "Entropy regularization and faster decremental matching in general graphs," *arXiv preprint arXiv:2312.09077*, 2023.

[63] M. Gupta, "Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time," in *FSTTCS*, vol. 29 of *LIPIcs*, pp. 227–239, Schloss Dagstuhl - Leibniz-Zentrum für Informatik,

2014.
[64] S. Bhattacharya, P. Kiss, and T. Saranurak, "Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time," in *Annual Symposium on Foundations of Computer Science*, IEEE, 2023.
[65] D. Adil and T. Saranurak, "Decremental $(1+\epsilon)$-approximate maximum eigenvector: Dynamic power method," *arXiv preprint arXiv:2402.17929*, 2024.
[66] M. Gupta and S. Khan, "Simple dynamic algorithms for maximal independent set and other problems," *arXiv preprint arXiv:1804.01823*, 2018.
[67] G. F. Italiano and P. Sankowski, "Improved minimum cuts and maximum flows in undirected planar graphs," *arXiv preprint arXiv:1011.2843*, 2010.
[68] A. Karczmarz, "Fully dynamic algorithms for minimum weight cycle and related problems," *arXiv preprint arXiv:2106.11744*, 2021.
[69] S. Dahlgaard, "On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter," in *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, vol. 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), pp. 48:1–48:14, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
[70] G. Goranci and M. Henzinger, "Incremental Approximate Maximum Flow in $m^{1/2+o(1)}$ update time," Nov. 2022.
[71] J. Li and T. Saranurak, "Deterministic weighted expander decomposition in almost-linear time," 2021.
[72] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. System Sci.*, vol. 26, no. 3, pp. 362–391, 1983.
[73] R. Khandekar, S. Rao, and U. Vazirani, "Graph partitioning using single commodity flows," *Journal of the ACM (JACM)*, vol. 56, no. 4, pp. 1–15, 2009.
[74] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1158–1167, IEEE, 2020.
[75] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," 2020.