Near-Linear Time Approximations for Cut Problems via Fair Cuts

Jason Li* Danupon Nanongkai† Debmalya Panigrahi‡ Thatchaphol Saranurak§

Abstract

We introduce the notion of *fair cuts* as an approach to leverage approximate (s,t)-mincut (equivalently (s,t)-maxflow) algorithms in undirected graphs to obtain near-linear time approximation algorithms for several cut problems. Informally, for any $\alpha \geq 1$, an α-fair (s,t)-cut is an (s,t)-cut such that there exists an (s,t)-flow that uses $1/\alpha$ fraction of the capacity of *every* edge in the cut. (So, any α-fair cut is also an α-approximate mincut, but not vice-versa.) We give an algorithm for $(1+\epsilon)$-fair (s,t)-cut in $\tilde{O}(m)$-time, thereby matching the best runtime for $(1+\epsilon)$-approximate (s,t)-mincut [Peng, SODA '16]. We then demonstrate the power of this approach by showing that this result almost immediately leads to several applications:

- the first nearly-linear time $(1+\epsilon)$-approximation algorithm that computes all-pairs maxflow values (by constructing an approximate Gomory-Hu tree). Prior to our work, such a result was not known even for the special case of Steiner mincut [Dinitz and Vainstein, STOC '94; Cole and Hariharan, STOC '03];
- the first almost-linear-work subpolynomial-depth parallel algorithms for computing $(1+\epsilon)$-approximations for all-pairs maxflow values (again via an approximate Gomory-Hu tree) in unweighted graphs;
- the first near-linear time expander decomposition algorithm that works even when the expansion parameter is polynomially small; this subsumes previous incomparable algorithms [Nanongkai and Saranurak, FOCS '17; Wulff-Nilsen, FOCS '17; Saranurak and Wang, SODA '19].

*Simons Institute for Theory of Computing, UC Berkeley. Email: jmli@alumni.cmu.edu

†Max Planck Institute for Informatics & University of Copenhagen & KTH. Email: danupon@gmail.com

‡Department of Computer Science, Duke University. Email: debmalya@cs.duke.edu

§University of Michigan, Ann Arbor. Email: thsa@umich.edu

Contents

1 Introduction

In the (s,t)-mincut problem, we are given an n-vertex m-edge graph $G = (V, E)$ with integer edge weights $w : E \to \mathbb{Z}_+$ bounded by U. The goal is to minimize the sum of the weight of edges whose removal make s unable to reach t. Unless stated otherwise, the input graphs are assumed to be undirected throughout the paper.

The (s,t)-mincut problem and its dual—(s,t)-maxflow—are among the most fundamental tools in graph algorithms and optimization. In particular, many reductions have been recently developed to show that if (s,t)-mincut (equivalently, (s,t)-maxflow) can be solved in almost or nearly linear time, then so are a number of fundamental graph problems. These problems include vertex connectivity [LNP+21] and Gomory-Hu tree [AKL+21] in unweighted graphs, deterministic global mincut and Steiner mincut [LP20], edge connectivity augmentation and edge splitting-off [CLP22], and hypergraph global mincut [CQ21a, MN21].

All these results require *exact* (s,t)-mincut algorithms. In other words, these reductions cannot exploit *approximate* (s,t)-mincut algorithms which can offer many advantages. For example, while the best randomized $(1+\epsilon)$-*approximate* (s,t)-mincut algorithm takes *nearly-linear*[1] time on weighted graphs [Pen16] (and *almost-linear*[2] time for deterministic algorithms [She13, KLOS14]), the fastest *exact* algorithms require $\tilde{O}\left(\min(m + n^{3/2}, m^{\frac{3}{2}-\frac{1}{328}}, m^{4/3+o(1)}U^{1/3})\right)$ time [GLP21, LS20, vdBLL+21, vdBLN+20][3] and are all inherently randomized.[4]

Moreover, in many popular models of computation such as parallel computing, distributed computing, etc., computing exact (s,t)-mincut is still far from efficient, and using approximation algorithms might be the only alternative. For example, it is known that the $(1+\epsilon)$-approximation algorithm (implied by [CS19, She13]) on undirected unweighted graphs requires almost-linear work and sub-polynomial depth in PRAM. In contrast, we are far from emulating this result for exact algorithms. In fact, the first small step toward solving exact (s,t)-mincut with almost-linear work and sub-polynomial depth would be doing so for the much simpler problem of (s,t)-reachability. And, the latter would involve breaking a major $\Omega(\sqrt{n})$ depth barrier.[5] Another example is in the distributed setting (the CONGEST model), where a nearly optimal algorithm for computing $(1+\epsilon)$-approximate (s,t)-mincut exists [GKK+15] while no nontrival algorithm is known for the exact version. These advantages of approximate (s,t)-mincuts motivate a natural question: *Can the existing reductions work with approximate (s,t)-mincut algorithms instead of the exact ones?*

To answer the above question, let us discuss first *why* many reductions work only with exact (s,t)-mincut. A crucial property of exact (s,t)-mincuts in undirected graphs that is used by these reductions (e.g., for Gomory-Hu tree, deterministic global mincut, Steiner cut, edge connectivity augmentation, and edge splitting-off) is the following *uncrossing property*:

(Uncrossing Property) For any vertices s and t, let $X \subset V$ be an (s,t)-mincut. Then,

[1] By nearly-linear time, we mean a running time of $\tilde{O}(m)$.

[2] By almost-linear time, we mean a running time of $m^{1+o(1)}$.

[3] Throughout, we use $\tilde{O}$ to hide poly $\log(n)$.

[4] In an independent result [CKL+22], an almost-linear time randomized algorithm has been shown for the (s,t)-mincut problem. Even when this independent result is taken into account, the best $(1+\epsilon)$-approximation algorithms are still superior to the best exact algorithm with respect to time complexities and randomness requirements.

[5] This is due to the reduction from directed maxflow to undirected maxflow (see e.g. [Mad11]) which works in the parallel setting. The reduction implies that if we can solve (s,t)-mincut exactly on undirected unweighted graphs in $O(W)$ work and $O(D)$ depth, then we can solve (s,t)-mincut exactly on *directed* unweighted graphs with $\tilde{O}(W)$ work and $\tilde{O}(D)$ depth. The latter captures the st-reachability problem as a special case.

for any $u, v \in X$, there exists $Y \subset X$ that is a (u, v)-mincut.

The uncrossing property is very useful from an algorithmic perspective since it gives a natural recursive tool – after finding an (s, t)-mincut, we can recurse on each side of the cut to find a (u, v)-mincut for every pair of vertices (u, v) on the same side of the cut. Indeed, the uncrossing property is more generally true for symmetric, submodular minimization problems and is at the heart of most of the beautiful structure displayed by undirected graph cuts and other symmetric, submodular functions. The uncrossing property, however, does *not* hold for $(1+\epsilon)$-approximate mincuts in general. This is the main bottleneck that prevents these reductions from being robust to approximation. As a result, for these problems, we fail to exploit the benefits of $(1+\epsilon)$-approximate (s, t)-mincut algorithms.

1.1 Our contributions

We subvert the above bottleneck by introducing a more robust notion of approximate mincuts called *fair cuts*. Informally, an α-fair (s, t)-cut is an (s, t)-cut such that there exists an (s, t)-flow f that uses $1/\alpha$ fraction of the capacity of every edge in the cut. (The reader should think of α as being close to 1.) Formally:

Definition 1.1 (Fair Cut)**.** Let $G = (V, E)$ be an undirected graph with edge capacities $c \in \mathbb{R}^E_{>0}$. Let s, t be two vertices in V. For any parameter $\alpha \geq 1$, we say that a cut (S, T) is a *α-fair (s, t)-cut* if there exists a feasible (s, t)-flow f such that $f(u, v) \geq \frac{1}{\alpha} \cdot c(u, v)$ for every $(u, v) \in E(S, T)$ where $u \in S$ and $v \in T$.

Observe that a 1-fair (s, t)-cut is an exact (s, t)-mincut. Moreover, an α-fair (s, t)-cut is also an α-approximate (s, t)-mincut. However, *not* all α-approximate (s, t)-mincuts are α-fair (s, t)-cuts.[6] In other words, a set of α-fair cuts is a proper subset of α-approximate cuts and a superset of exact (s, t)-mincuts.

We show that the notion of fair cuts allow us to combine the key features of both approximate cuts and exact cuts. First, fair cuts admit a property for approximate cuts that is analogous to uncrossing for exact mincuts, which we prove in Appendix B for completeness.

Lemma 1.2 (Approximate Uncrossing Property)**.** *For any vertices s and t, let (S, T) be an α-fair (s, t)-mincut. Then, for any $u, v \in S$, there exists $R \subset S$ such that $(R, V \setminus R)$ is an α-approximate (u, v)-mincut.*

Second, while computing a fair cut can be harder than an approximate mincut (since any fair cut is an approximate mincut but not vice-versa), we give a nearly-linear time algorithm for computing a $(1+\epsilon)$-fair (s, t)-mincut.

Theorem 1.3 (Fair Cut)**.** *Given a graph $G = (V, E)$, two vertices $s, t \in V$, and $\epsilon \in (0, 1]$, we can compute with high probability a $(1+\epsilon)$-fair (s, t)-cut in $\tilde{O}(m/\epsilon^3)$ time.*

[6] As a simple example, consider a path $v - s - t$ on three vertices. Clearly, the cut $\{s\}$ contains both edges and is therefore a 2-approximate (s, t)-mincut. However, there is no (s, t)-flow that can saturate both edges to fraction $\frac{1}{2}$. To motivate our choice of terminology (fair cuts), note that if an (s, t)-cut is a α-approximate (s, t)-mincut, it follows by flow-cut duality that any (s, t)-maxflow will *cumulatively* saturate the edges of the cut to a fraction $\geq \frac{1}{1+\alpha}$. But, as we saw in the previous example, this saturation need not be *fair* in the sense that some edges might not be saturated at all. In this context, a α-fair cut demands the additional property that *each* edge be saturated to a fraction $\geq \frac{1}{\alpha}$ (in the sense of "max-min" fairness).

We note that the only reason why our algorithm is randomized is because we use the *congestion approximator* by [RST14, Pen16]. This can be made deterministic based on an algorithm by [CGL+20], but the running time would be $m^{1+o(1)}/\epsilon^3$ instead. Moreover, we remark that although we will focus on $(1+\epsilon)$-fair (s,t)-cuts, the corresponding (s,t)-flow can be obtained from a fair cut in $\tilde{O}(m/\epsilon)$ time using a standard application of a $(1+\epsilon)$-approximate max-flow algorithm of Sherman [She17].

1.2 Applications

We demonstrate the power of fair cuts by using it to improve the time complexity of several problems.

Gomory-Hu Tree. The Gomory-Hu (GH) tree is a compact representation of a (u,v)-mincut (and therefore, (u,v)-maxflow values) between every pair of vertices (u,v) of a graph, and has a large number of applications. It captures fundamental questions such as global, $(s,t)-$, and Steiner mincuts as special cases. There has been much progress on exact and approximation algorithms for this problem recently (e.g., [LP21, AKT20a, AKT20b, AKT21c, AKL+21, AKT21a, AKT21b, LPS21, Zha21a, Zha21b]). The fastest among these is the $(1+\epsilon)$-*approximation* algorithm by Li and Panigrahi [LP21] whose time complexity is equal to poly-logarthmic calls to any *exact* (s,t)-mincut algorithm, i.e. $\tilde{O}\left(\min(m+n^{3/2}, m^{\frac{3}{2}-\frac{1}{328}}, m^{4/3+o(1)}U^{1/3})\right)$.

By replacing the exact max-flow calls by our $(1+\epsilon)$-fair cut algorithm in [LP21], we get a nearly-linear time algorithm for approximating the Gomory-Hu tree (which is equivalent to finding all-pairs maxflow values by known reductions, e.g., [AKT20a]):

Theorem 1.4 (Nearly-linear time Gomory-Hu tree)**.** *For any* $\epsilon > 0$*, there is a* $\tilde{O}(m \cdot \mathrm{poly}(1/\epsilon))$*-time randomized algorithm that constructs, with high probability, a* $(1+\epsilon)$*-approximate Gomory-Hu tree in weighted undirected graphs.*

Prior to our work, a nearly-linear time (approximation) algorithm was not known even for the special case of the *Steiner mincut problem*. In this problem [DV94, CH03, HKP07, BHKP07, LP20], we are interested in finding a cut of minimum value that disconnects a given set of terminal vertices. For this problem, Li and Panigrahi [LP20] gave an exact algorithm using poly-logarithmic exact max-flow calls. Before our work, no improvement in the running time was known if we allow $(1+\epsilon)$-approximation instead of the exact Steiner mincut. Since the Steiner mincut problem is a minimal generalization of global and (s,t)-mincuts, our paper is the first to obtain nearly-linear time (approximation) algorithms for cut problems that go beyond these two problems.

Parallelization. Since the use of exact max-flow is the only bottlenect to parallelize the approximate GH tree algorithm of [LP21], the following parallel algorithm also follows.

Theorem 1.5 (Parallel GH-tree)**.** *For any* $\epsilon > 0$*, there is a* $\tilde{O}(m^{1+o(1)}/\mathrm{poly}(\epsilon))$*-work* $(m^{o(1)}/\mathrm{poly}(\epsilon))$*-depth randomized algorithm that constructs, with high probability, a* $(1+\epsilon)$*-approximate Gomory-Hu tree in unweighted undirected graphs.*

We are not aware of prior work on parallel GH algorithms (except some experiments, e.g. [MCJ20, CRJ17]). This is likely because previous GH trees algorithms, even the approximate ones [LP21], inherently require solving max-flow exactly, which is well beyond current techniques in the parallel setting.

Expander Decomposition. In the last decade, numerous fast graph algorithms are based on fast algorithms for computing an expander decomposition. For some examples of such applications, see e.g. [ST04, KLOS14, She13, NSW17, CS19, BBG$^+$20].

We say that a (weighted) graph $G = (V, E)$ is a ϕ-expander if for every cut $(S, V \setminus S)$, we have that the cut size $\delta(S) \geq \min\{\mathrm{vol}(S), \mathrm{vol}(V \setminus S)\}$ where the volume of S is $\mathrm{vol}(S) = \sum_{v \in S} \deg(v)$. A (ϵ, ϕ)-expander decomposition of G is a partition $\{V_1, \ldots, V_k\}$ of vertices such that each $G[V_i]$ is a ϕ-expander and $\sum_i \delta(V_i) \leq \epsilon \cdot \mathrm{vol}(V)$, i.e., the total weight of edges crossing the partition is at most ϵ-fraction.

There are two incomparable fastest algorithms for computing expander decompositions. First, [NS17, Wul17] gave $m^{1+o(1)}$-time algorithms that computes a $(\phi n^{o(1)}, \phi)$-expander decomposition for any $\phi > 0$. These subpolynomial factors are sometimes undesirable. Second, [SW19] gave a $\tilde{O}(m/\phi)$-time algorithm that computes a $(\tilde{O}(\phi), \phi)$-expander decomposition for any $\phi > 0$. This algorithm is slower than the first one when $\phi < 1/n^{0.1}$. Using fair cuts, we obtain an algorithm that subsumes both these sets of results and is optimal up to poly-logarithmic factors.

Theorem 1.6 (Near-linear expander decomposition)**.** *For any $\phi > 0$, there is a randomized $\tilde{O}(m)$-time algorithm that with high probability computes a $(\tilde{O}(\phi), \phi)$-expander in weighted undirected graphs.*

Open problems. We believe that our notion of fair cuts opens up many interesting directions for future research. We mention some examples. (i) A natural goal is to extend our efficient $(1+\epsilon)$-fair (s,t)-cut to other computational models, such as the distributed (CONGEST) setting, where exact (s,t)-mincut algorithms are much slower/inefficient compared to approximate (s,t)-mincut algorithms. This will lead to efficient algorithms for approximating, e.g., Gomory-Hu tree and Steiner mincut in these models as well. (ii) The notion of *fair vertex cuts* can be defined in a similar fashion to fair (edge) cuts defined in this paper. It would be interesting to design an efficient algorithm for finding a fair vertex cut and use it to obtain nearly-linear time algorithms for approximating the vertex connectivity and hypergraph global mincut. These results can also be extended to other computational models. (iii) We also hope that the notion of fair cuts can be extended to more general contexts such as the minimization of symmetric, submodular functions. In turn, this will significantly improve our understanding of the approximation-efficiency tradeoff in minimization problems defined for these function classes.

Independent result. Our result is obtained independently from the recently announced almost-linear time bound for min-cost flow by Chen, Kyng, Liu, Peng, Gutenberg, and Sachdeva [CKL$^+$22]. Plugging this result into existing reductions in [AKL$^+$21, LNP$^+$21, CQ21b, MN20] help solve problems such as GH tree and vertex connectivity in unweighted graphs, approximate GH tree in weighted graphs, and hypergraph global mincut in $m^{1+o(1)}$ time. Even assuming this result, our algorithms are faster in both randomized and deterministic settings; for the latter, our running time is $m^{1+o(1)}$ whereas the best exact (s,t)-mincut algorithm takes $\tilde{O}(m \min(\sqrt{m}, n^{2/3}))$ time [GR98]. Finally, our algorithms can be adapted to other models such as parallel computation whereas this is well beyond existing techniques for exact (s,t)-mincut.

2 Overview of Techniques

2.1 Computing Fair Cuts (Proof Idea of Theorem 1.3)

Our key subroutine for computing fair cuts is called AlmostFair. Here, we describe at a high-level what the AlmostFair subroutine does, how to use it for computing fair cuts, and finally how to obtain the AlmostFair subroutine itself.

Say we are given an (s,t)-cut (S,T) which may be far from being fair. The AlmostFair subroutine works on one side of the (s,t)-cut, say T, and returns a partition (P_t, T') of T such that $t \in T'$. We think of P_t as the part that we "prune" out of T. Our first guarantee is that the remaining part T' is "almost fair" in the following sense: each boundary edge in $E(S,T')$ (i.e., those edges that are not in $E(P_t, T')$) can simultaneously send flow of value at least $(1-\beta)$-fraction of its capacity to t, for a small parameter β that we can choose. This guarantee alone would have been weak if the pruned set P_t is so big that there are few edges left in $E(S,T')$. However, the second guarantee of AlmostFair says that, if P_t is big, then $(V \setminus T', T')$ is actually a much smaller (s,t)-cut than the original cut (S,T) in terms of cut value. More formally, we have $\delta_G(T') \leq \delta_G(T) - \beta \cdot \delta_G(S, P_t)$ meaning that the decrease in the cut size is at least β times the total capacity of $E(S, P_t)$.

With these two guarantees of AlmostFair, given any (s,t)-cut (S,T), we can iteratively improve this cut to make it fair as follows. We call AlmostFair on both S and T and obtain (P_s, S') and (P_t, T'). Let's consider two extremes. If both pruned sets P_s and P_t are tiny, then there is an (s,t)-flow that almost fully saturates *every* edge in $E(S', T')$. This certifies (S,T) is very close to being fair as P_s and P_t are tiny. However, if either P_s or P_t is very big, say P_t, then $(S \cup P_t, T')$ is an (s,t)-cut of much smaller value than the original cut (S,T). Therefore, this is progress too and we can recursively work on this new cut $(S \cup P_t, T')$. To make the intuition on these two extremes work, we iteratively call AlmostFair using a parameter β that increases slightly in every round. The full algorithm is presented in Section 5.

Now, let us sketch the AlmostFair subroutine itself. This subroutine is based on Sherman's algorithm for computing a $(1+\epsilon)$-approximate max-flow [She13] (for any $\epsilon > 0$), which in turn uses the multiplicative weight update (MWU) framework.[7] Given the t-side T of an (s,t)-cut, if we call Sherman's algorithm where the demand is specified so that each boundary edge should send flow at its full capacity to sink t, then the algorithm would either return a flow satisfying this demand with congestion $(1+\epsilon)$ or return a "violating" cut certifying that the demand is not feasible. In the former case, this would satisfy the guarantee of AlmostFair where $P_t = \emptyset$ after scaling down the flow by a $(1+\epsilon)$ factor. Unfortunately, in the latter case, the algorithm does not guarantee the existence of the flow that we want. The reason behind this problem is that whenever the algorithm detects a violating cut, the algorithm is just terminated. In a more general context, this holds for most (if not all) MWU-based algorithms for solving linear programs; in each round of the MWU algorithm, whenever "the oracle" certifies that the linear program is infeasible, then we just terminate the whole algorithm.

Interestingly, we fix this issue by "insisting on continuing" the MWU algorithm. Once we detect a violating cut, we include it into the pruned set, cancel the demand inside this pruned set, and continue updating weights in the MWU algorithm. After the last round, the flow constructed

[7] Sherman's original presentation in [She13] does not explicitly use the MWU framework. Although this alternative interpretation was already known to experts, our MWU-based presentation of his algorithm is arguably simpler and more intuitive.

via MWU indeed sends flow from each remaining boundary edge that is not pruned out, which is exactly our goal. The detailed algorithm is presented in Section 4.

2.2 From Fair Cuts to Approximate Isolating Cuts

We believe that the notion of fair cuts can be useful in many contexts since it offers a more robust alternative to approximate mincuts. In this paper, we first use it to obtain an approximate isolating cuts algorithm. We define the isolating cuts problem first.

Definition 2.1. Given a weighted, undirected graph $G = (V, E)$ and a subset of terminals $S = \{s_1, s_2, \ldots, s_k\}$, the goal of the isolating cuts problem is to find a set of disjoint sets $S_1, S_2, \ldots, S_k$ such that for each i, the cut $(S_i, V \setminus S_i)$ is a mincut that separates $s_i \in S_i$ from the remaining terminals $S \setminus \{s_i\} \subseteq V \setminus S_i$. If S_i is a $(1+\epsilon)$-approximate mincut separating s_i from the remaining terminals, then the corresponding problem is called the $(1+\epsilon)$-approximate isolating cuts problem.

Using fair cuts, we obtain a near-linear time algorithm for approximate isolating cuts.

Theorem 2.2. *There is an algorithm for finding $(1+\epsilon)$-approximate isolating cuts that takes $\tilde{O}(m \cdot \text{poly}(1/\epsilon))$ time.*

Li and Panigrahi [LP20] gave an algorithm for finding *exact* isolating cuts using $O(\log n)$ (s,t)-max-flow/mincut computations that crucially relies on the uncrossing property of (s,t)-mincuts. This property ensures that if we take a minimum isolating cut X containing a terminal vertex s and a crossing mincut Y, then their intersection $X \cap Y$ or difference $X \setminus Y$ (depending on which set the terminal vertex s is in) is also a minimum isolating cut. This allows partitioning of the graph by removing edges corresponding to a set of mincuts, such that each terminal is in one of the components of this partition. For each terminal, the corresponding minimum isolating cut is now obtained by simply contracting the rest of the components and running a max-flow algorithm on this contracted graph. The advantage of this contraction is that the total size of all the graphs on which we are running max-flows is only a constant times the size of the overall graph.

Unfortunately, approximate mincuts don't satisfy this uncrossing property, which renders this method unusable if we replace exact mincut subroutines with faster $(1+\epsilon)$-approximate mincuts. But, if we instead used fair cuts, then we can show the following: if X is a $(1+\epsilon)$-approximate minimum isolating cut containing terminal s and Y is a $(1+\alpha)$-fair cut, then either $X \cap Y$ or $X \setminus Y$ (whichever set contains s) is a $(1+\epsilon)(1+\alpha)$-approximate minimum isolating cut. This allows us to use the framework in [LP20]. Since the number of fair cuts we remove in forming the components is only $O(\log n)$, the multiplicative growth in the approximation factor can be offset by scaling the parameter in fair cuts by the same logarithmic factor. The advantage in using fair cuts over exact mincuts is that the running time of the former is near-linear by Theorem 1.3, which helps establish Theorem 2.2. The details of this algorithm are presented in Section 6.

2.3 From Approximate Isolating Cuts to Approximate GH-trees

Finally, we use approximate isolating cuts to obtain an approximate GH tree algorithm. [LP21] gives a recursive algorithm for computing an approximate GH tree but using exact isolating cuts. We observe that the latter can be replaced by approximate isolating cuts provided the approximation is *one-sided* in the following sense: the "large" recursive subproblem needs to preserve mincut values exactly. But, in general, if we use the approximate isolating cuts subroutine as a black box, this would not be the case. To alleviate this concern, we augment the approximate isolating cuts procedure using an additional fairness condition for the isolating cuts returned by the algorithm. This fairness condition ensures that although we do not have one-sided approximation, the approximation factor in the "large" subproblem can be controlled using a much finer parameter than the overall approximation factor of the algorithm, which then allows us to run the recursion correctly. The details of the GH tree algorithm establishing Theorem 1.4 are presented in Section 7.

2.4 From Fair Cuts to Near-linear time Expander Decomposition

Via fair cuts, we will speed up the algorithm by [SW19] for computing a $(\tilde{O}(\phi), \phi)$-expander decomposition in time $\tilde{O}(m/\phi)$ to $\tilde{O}(m)$. There are two main steps in the algorithm by [SW19]: the cut-matching step and the trimming step. The cut-matching step can be solved in $\tilde{O}(m)$ time simply by applying the near-linear-time approximate maxflow algorithm by [Pen16].[8] The harder step to speed up is the trimming step. However, we observe that the cut problem needed to be solved in this step is actually a one-sided version of the fair cut problem, which is an easier problem. By calling our fair cut algorithm, we immediately obtain a $\tilde{O}(m)$-time algorithm for the trimming step. See details in Section 8.

3 Preliminaries

Given a undirected capacitated/weighted graph $G = (V, E)$ with edge capacities/weights is $c \in \mathbb{R}^E_{\geq 0}$ and an edge set $E' \subseteq E$, we let $c(E') = \sum_{e \in E'} c(e)$ be the total capacity of E'. For simplicity, we assume that the ratio between the largest and lowest edge capacities or weights are poly(n). For any disjoint sets $S, T \subseteq V$, we let $\delta_G(S) = c(E(S, V \setminus S))$ denote the cut size of S and $\delta_G(S, T) = c(E(S, T))$ denote the total capacity of edges from S to T. For any distinct vertices s and t, let $\lambda_G(s, t)$ be the minimum-weight s-t cut. We sometimes omit G when it is clear from the context.

Flow. A *flow* $f : V \times V \to \mathbb{R}$ satisfies $f(u, v) = -f(v, u)$ and $f(u, v) = 0$ for $\{u, v\} \notin E$. The notation $f(u, v) > 0$ means that mass is routed in the direction from u to v, and vice versa. The *congestion* of f is $\max_{\{u,v\} \in E} \frac{|f(u,v)|}{c(e)}$. If the congestion is at most 1, we say that f *respects the capacity* or f is *feasible*. For each vertex $u \in V$, the *net flow out of vertex* u, denoted by $f(u) = \sum_{v \in V} f(u, v)$, is the total mass going out of u minus the total mass coming into u. More generally, for any vertex set $S \subseteq V$, we can define the *net flow out of* S as $f(S) = \sum_{u \in S} f(u) = \sum_{u \in S, v \in V} f(u, v)$. The *net flow out from* S *to* T is denoted by $f(S, T) = \sum_{u \in S, v \in T} f(u, v)$. Observe that we always have $f(V) = 0$.

A *demand function* $\Delta : V \to \mathbb{R}$ is a function where $\sum_{v \in V} \Delta(v) = 0$. We say that flow f *satisfies* demand Δ if $f(v) = \Delta(v)$ for all $v \in V$. For any $S \subseteq V$, let $\Delta(S) = \sum_{v \in S} \Delta(v)$ be the *total demand* on S. Observe $\Delta(V) = f(V) = 0$. By the max-flow min-cut theorem, we have the following:

Fact 3.1. *For any $\epsilon \geq 0$, $|\Delta(S)| \leq \epsilon \cdot \delta(S)$ for all $S \subseteq V$ iff there is a flow with congestion ϵ that satisfies Δ.*

[8] For reader who are familiar with [SW19], their algorithm applies the push-relabel flow algorithm that takes $\tilde{O}(m/\phi)$ time, instead of using an $\tilde{O}(m)$-time approximate max flow algorithm, because the push-relabel algorithm has fewer log factors in the running time.

For a flow f and a demand function Δ, define the *excess* Δ^f as $\Delta^f(v) = \Delta(v) - f(v)$ for every $v \in V$. We think of excess as a remaining demand function. We say that f *ϵ-satisfies* Δ if $|\Delta^f(S)| \leq \epsilon \cdot \delta(S)$ for all $S \subseteq V$. That is, by Fact 3.1, there exists a flow f_{aug} with congestion ϵ where $f + f_{aug}$ satisfies Δ. Note that f 0-satisfies Δ iff f satisfies Δ.

For any two vertices $s, t \in V$, an *(s,t)-cut* (S,T) is a cut such that $s \in S$ and $t \in T$. An *(s,t)-flow* f obeys $f(v) = 0$ for all $v \neq s, t$. Similarly, an *(s,t)-demand function* Δ obeys $\Delta(v) = 0$ for all $v \neq s, t$. That is, an (s,t)-demand function is satisfied only by an (s,t)-flow.

Congestion Approximators. When we want to argue that flow f ϵ-satisfies a demand function Δ, it can be inconvenient to ensure that $|\Delta^f(S)| \leq \epsilon \cdot \delta(S)$ for all $S \subseteq V$ because there are exponentially many sets. Surprisingly, there is a collection $\mathcal{S}$ of linearly many sets of vertices, where if $|\Delta(S)| \leq \epsilon \cdot \delta(S)$ for each $S \in \mathcal{S}$, then this is also true for all $S \subseteq V$ with some polylog(n) blow-up factor. Moreover, $\mathcal{S}$ can be computed in near-linear time.

Theorem 3.2 (Congestion approximator [RST14, Pen16])**.** *There is a randomized algorithm that, given a graph $G = (V, E)$ with n vertices and m edges, constructs in $\tilde{O}(m)$ time with high probability a laminar family $\mathcal{S}$ of subsets of V such that*

1. *$\mathcal{S}$ contains at most $2n$ sets,*
2. *each vertex appears in $O(\log n)$ sets of $\mathcal{S}$, and*
3. *for any demand function Δ on V, if $|\Delta(S)| \leq \delta(S)$ for all $S \in \mathcal{S}$, then $|\Delta(R)| \leq \gamma_{\mathcal{S}}\delta(R)$ for all $R \subseteq V$ for a* quality *factor $\gamma_{\mathcal{S}} = O(\log^4 n)$.*

Graphs with Boundary Vertices. Given a set $U \subseteq V$, let $G\{U\}$ denote the following "*induced subgraph with boundary vertices*": start with induced subgraph $G[U]$, and for each edge $e = (u, v) \in E(U, V \setminus U)$ with endpoint u in U, create a new vertex x_e and add the edge (x_e, u) to $G\{U\}$ of the same capacity as e. Let $N_G\{U\}$ be the vertex set of $G\{U\}$ and define $N_G\langle U\rangle = N_G\{U\} \setminus U$. We call vertices in $N_G\langle U\rangle$ *boundary vertices*. We simply write $N\{U\}$ and $N\langle U\rangle$ instead of $N_G\{U\}$ and $N_G\langle U\rangle$ when the context is clear. Observe that the degree $\deg_{G\{U\}}(x_e)$ of each boundary vertex $x_e \in N\langle U\rangle$ in $G\{U\}$ is simply the capacity $c(e)$ of edge e. We will use this notation very often in the paper.

Boundary Demand Functions. In our context, the sink node $t \in U$ is usually given. The *full U-boundary demand function* $\Delta_U : V(G\{U\}) \to \mathbb{R}$ is defined such that

$$\Delta_U(v) = \begin{cases} \deg_{G\{U\}}(v) & \text{if } v \in N\langle U\rangle \\ 0 & \text{if } v \in U \setminus t \\ -\Delta(N\langle U\rangle) & \text{if } v = t. \end{cases}$$

That is, any flow satisfying Δ_U sends flow from each boundary vertex of $G\{U\}$ at full capacity to t. We also write $\Delta_{U,t}$ when it is not clear from the context what t is. More generally, given any demand function $\Delta' : V(G\{U\}) \to \mathbb{R}$, we say that Δ' is a *U-boundary demand function* if $\Delta'(v) = 0$ for all $v \in U \setminus t$, $\Delta'(t) = -\Delta(N\langle U\rangle)$. That is, Δ' is completely determined once we specify the demand values on boundary vertices $N\langle U\rangle$.

One-Sided Fair Cut. Finally, the following "one-sided" version of a fair cut (Definition 1.1) will be useful.

Definition 3.3 (One-sided Fair Cut)**.** Let $G = (V, E)$ be an undirected graph with edge capacities $c \in \mathbb{R}^E_{>0}$. Let s be a vertex in V. For any parameter $\alpha \geq 1$, we say that a cut (S, T) is an *s-sided α-fair cut* if there exists a feasible flow f such that

1. $f(v) = 0$ for all $v \in S \setminus \{s\}$
2. $f(u, v) \geq \frac{1}{\alpha} \cdot c(u, v)$ for every $(u, v) \in E(S, T)$ where $u \in S$ and $v \in T$.

In other words, the flow f sends flow from s to the boundary $E(S, T)$ in a way that almost saturates every edge in $E(S, T)$, but we do not care about the behavior of f beyond $E(S, T)$.

Clearly, an α-fair (s, t)-cut is an s-sided α-fair cut since we can take the same flow f that witnesses the α-fair (s, t)-cut. However, we will only require the one-sided version in our isolating cuts application in Section 6.

4 Almost Fair Cuts via Multiplicative Weight Updates

The key subroutine used for proving Theorem 1.3 is the algorithm below.

Theorem 4.1 (Almost Fair Cuts)**.** *There is an algorithm* $\text{AlmostFair}(G, U, t, \epsilon, \beta)$ *that, given a graph* $G = (V, E)$ *with a sink node* $t \in V$*, a set* $U \subseteq V$ *where* $t \in U$*, and parameters* $\beta \geq 0$ *and* $\epsilon > 0$*, returns a partition* (P, U') *of* U *where* $t \in U'$ *with the following properties:*

1. $\delta_G(U') \leq \delta_G(U) - \beta\delta_G(P, V \setminus U)$ *(equivalently,* $\delta_G(P, U') \leq (1 - \beta)\delta_G(P, V \setminus U)$*), and*
2. *There exists a flow* f'_{sat} *in* $G\{U'\}$ *with congestion* $(1 + \epsilon)$ *satisfying a* U'*-boundary demand function* Δ' *such that*

$$\Delta'(v) = (1 - \beta) \deg_{G\{U'\}}(v) \qquad \text{for all old boundary vertices } v \in N\langle U'\rangle \cap N\langle U\rangle$$
$$|\Delta'(v)| \leq (1 + \epsilon) \deg_{G\{U'\}}(v) \qquad \text{for all new boundary vertices } v \in N\langle U'\rangle \setminus N\langle U\rangle$$

The algorithm takes $\tilde{O}(|E(G\{U\})|/\epsilon^2)$ *time and is correct with high probability.*[9]

The rest of this section is for proving Theorem 4.1. For convenience, we write $H = G\{U\}$ and let n and m denote the number of vertices and edges in H throughout this section. Let B be the incidence matrix of H. Observe that, for any flow f on H, we have $(Bf)_v = f(v)$ is the net flow out of v. We can view Bf as a vector in $\mathbb{R}^{V(H)}$. Define

$$\Delta = (1 - \beta)\Delta_U$$

as the full U-boundary demand function on $G\{U\}$ after scaled down by $(1 - \beta)$ factor. For any $U' \subseteq U$, the *restriction* $\Delta|_{U'}$ of Δ is a U-boundary demand function obtained from Δ by zeroing out the entries on $N\langle U\rangle \setminus N\langle U'\rangle$, i.e., the boundary vertices of U which are not boundaries of U', and then setting the entry on t so that $\sum_{v \in V(H)} \Delta|_{U'}(v) = 0$. Similarly, we view Δ and also $\Delta|_{U'}$ as vectors in $\mathbb{R}^{V(H)}$.

[9]Note that the guarantee that $|\Delta'(v)| \leq (1 + \epsilon) \deg_{G\{U'\}}(v)$ for all new boundary vertices $v \in N\langle U'\rangle \setminus N\langle U\rangle$ in fact follows from the guarantee that f'_{sat} has congestion $(1 + \epsilon)$. We state both guarantees explicitly for convenience.

4.1 Algorithm

Initialization. We start by computing a congestion approximator $\mathcal{S}$ of H with quality $\gamma_{\mathcal{S}} = O(\log^4 n)$ using Theorem 3.2. For a technical reason, it is more convenient if no set in $\mathcal{S}$ contains sink t. From now, we will assume this, which is justified by the following observation:

Proposition 4.2. *Given the family $\mathcal{S}$ from Theorem 3.2 and a vertex t, there is a linear time algorithm that returns another family $\mathcal{S}'$ with the same guarantee as in Theorem 3.2 but with additional guarantee that each set $S \in \mathcal{S}'$ does not contains t.*

Proof. Replace each set $S \in \mathcal{S}$ where $t \in S$ with its complement $V(H) \setminus S$. Observe that $\mathcal{S}$ is now a larminar family on $V(H) \setminus t$ where $|\mathcal{S}|$ does not change, and the number of sets containing each vertex may increase only by $O(\log n)$. Hence, the first and second properties of Theorem 3.2 still hold. The third property still holds as well because $|\Delta(S)| = |\Delta(V(H) \setminus S)|$ for all S. □

Our algorithm is based on the Multiplicative Weight Update framework and so it works in rounds. For round i, we maintain *weights* $w^i_{S,\circ} \geq 0$ for each $S \in \mathcal{S}$ and $\circ \in \{+,-\}$ and define the potential $\phi^i \in \mathbb{R}^{V(H)}$ where

$$\phi^i_v = \sum_{S \ni v} \frac{1}{\delta_H(S)} (w^i_{S,+} - w^i_{S,-})$$

for each vertex v. As no set $S \in \mathcal{S}$ contains t, we will always have $\phi^i_t = 0$ for all i. Initially, we set $w^1_{S,\circ} = 1$ for all $S \in \mathcal{S}$, $\circ \in \{+,-\}$.

The algorithm also maintains a decremental subset V^i where $t \in V^i \subseteq V^{i-1}$ for all i. We initialize V^0 as follows. First, set $V^0 = V(H)$. While there exists $S \in \mathcal{S}$ where $\Delta|_{V^0}(S) > \delta_H(S)$, which certifies that there is no feasible flow on H satisfying $\Delta|_{V^0}$ by Fact 3.1, we update $V^0 \leftarrow V^0 \setminus S$ (in particular, the function $\Delta|_{V^0}$ changes). Let D^0 contain all the vertices we removed from V^0. Now, we are ready to state the main algorithm.

Main Algorithm. For round $i = 1, 2, \ldots, T$ where $T = \Theta(\log(n)/\alpha^2)$ and $\alpha = \epsilon/\gamma_{\mathcal{S}}$, we do the following:

1. Define f^i on H such that for each edge (u,v), $f^i(u,v)$ flows from high potential to low potential at maximum capacity. That is, for every edge (u,v) in H,

$$f^i(u,v) = \begin{cases} c(u,v) & \text{if } \phi^i_u > \phi^i_v \\ 0 & \text{if } \phi^i_u = \phi^i_v \\ -c(u,v) & \text{if } \phi^i_u < \phi^i_v. \end{cases}$$

2. Using Lemma 4.9, compute a *deletion set* $D^i \subseteq V(H) \setminus t$ and set $V^i \leftarrow V^{i-1} \setminus D^i$, where D^i satisfies the following:

$$\text{if } D^i \neq \emptyset, \text{ then } \Delta|_{V^{i-1}}(D^i) > \delta_H(D^i), \text{ and}$$
$$\langle \phi^i, \Delta|_{V^i} \rangle = \langle \phi^i, \Delta|_{V^{i-1} \setminus D^i} \rangle \leq \langle \phi^i, Bf^i \rangle.$$

3. For each $S \in \mathcal{S}$, let

$$r^i_S = \frac{(\Delta|_{V^i})^{f^i}(S)}{\delta_H(S)} = \frac{\Delta|_{V^i}(S) - f^i(S)}{\delta_H(S)}$$

be the *relative total excess* at S compared to the cut size in round i.

4. Update the weights

$$w_{S,+}^{i+1} = w_{S,+}^{i} \cdot e^{\alpha r_S^i} \quad \text{and} \quad w_{S,-}^{i+1} = w_{S,-}^{i} \cdot e^{-\alpha r_S^i}.$$

After T rounds, we compute the *pruned set* $P = \cup_{i=0}^{T} D^i$ and let $U' = U \setminus P$. Finally, we return the partition (P, U').

4.2 Correctness

We prove that the partition (P, U') outputted by our algorithm satisfies the requirement in Theorem 4.1. The first important thing to understand our algorithm is to formally see how it is captured by the Multiplicative Weight Update (MWU) algorithm, which we recall below:

Theorem 4.3 (Multiplicative Weights Update [AHK12])**.** *Let J be a set of indices, and let $\alpha \leq 1$ and $\omega > 0$ be parameters. Consider the following algorithm:*

1. *Set $w_j^{(1)} \leftarrow 1$ for all $j \in J$*
2. *For $i = 1, 2, \ldots, T$ where $T = O(\omega^2 \log(|J|)/\alpha^2)$:*
   (a) *The algorithm is given a "gain" vector $g^i \in \mathbb{R}^J$ satisfying $\|g^i\|_\infty \leq \omega$ and $\langle g^i, w^i \rangle \leq 0$*
   (b) *For each $j \in J$, set $w_j^i \leftarrow w_j^{i-1} \exp(\alpha g_j^i) = \exp(\alpha \sum_{i' \in [i]} g_j^{i'})$*

At the end of the algorithm, we have $\frac{1}{T} \sum_{i \in [T]} g_j^i \leq \alpha$ for all $j \in J$.[10]

To apply Theorem 4.3 into our setting, we define $J = \mathcal{S} \times \{+, -\}$. That is, we work with indices $(S, +)$ and $(S, -)$ for $S \in \mathcal{S}$. We use the same weights w^i and error parameter α as the algorithm, and we set $\omega = 2$. For each iteration i and $S \in \mathcal{S}$, we define

$$g_{S,\pm}^i = \pm r_S^i = \pm \frac{\Delta|_{V^i}(S) - f^i(S)}{\delta_H(S)}.$$

Observe that the weights $w_{S,\pm}^i$ are updated in Step 4 exactly as $w_{S,\pm}^i \leftarrow w_{S,\pm}^{i-1} \exp(\alpha g_{S,\pm}^i)$. With this setting, we show that our gain vector g^i indeed satisfies the condition in Step 2a of Theorem 4.3.

Lemma 4.4. *For each i, we have $\|g^i\|_\infty \leq 2$ and $\langle g^i, w^i \rangle \leq 0$.*

Proof. To show $\|g^i\|_\infty \leq 2$, we have

$$|g_{S,\pm}^i| = \left| \frac{\Delta|_{V^i}(S) - f^i(S)}{\delta_H(S)} \right| \leq \left| \frac{\Delta|_{V^i}(S)}{\delta_H(S)} \right| + \left| \frac{f^i(S)}{\delta_H(S)} \right| \leq 1 + 1,$$

To see why the last inequality holds, we have (1) $\Delta|_{V^0}(S) \leq \delta_H(S)$ for all $S \in \mathcal{S}$ by the initialization of V^0, (2) $\Delta|_{V^i}(S) \geq 0$ for all i because $t \notin S$, and (3) $\Delta|_{V^i}(S)$ may only decrease as V^i is a decremental set. Also, we have $|f^i(S)| \leq \delta_H(S)$ because each f^i respects the capacity.

[10] More generally, for any value val, if we have $\langle g^i, w^i \rangle \leq \mathsf{val}$ for all i, the MWU algorithm guarantees that $\frac{1}{T} \sum_{i \in [T]} g_j^i \leq \mathsf{val} + \alpha$, for all j. Here, we use a special case when $\mathsf{val} = 0$.

To show $\langle g^i, w^i \rangle \le 0$, first observe that $\langle g^i, w^i \rangle = \langle \phi^i, \Delta|_{V^i} \rangle - \langle \phi^i, Bf^i \rangle$ exactly.

$$\begin{aligned}
\langle g^i, w^i \rangle &= \sum_{S \in \mathcal{S}} (g^i_{S,+} w^i_{S,+} + g^i_{S,-} w^i_{S,-}) \\
&= \sum_{S \in \mathcal{S}} (w^i_{S,+} - w^i_{S,-}) r^i_S \\
&= \sum_{S \in \mathcal{S}} \frac{w^i_{S,+} - w^i_{S,-}}{\delta_H(S)} \left(\Delta|_{V^i}(S) - f^i(S) \right) \\
&= \sum_{S \in \mathcal{S}} \frac{w^i_{S,+} - w^i_{S,-}}{\delta_H(S)} \sum_{v \in S} \left(\Delta|_{V^i}(v) - (Bf^i)_v \right) \\
&= \sum_{v \in V(H)} \left(\Delta|_{V^i}(v) - (Bf^i)_v \right) \sum_{S \ni v} \frac{w^i_{S,+} - w^i_{S,-}}{\delta_H(S)} \\
&= \sum_{v \in V(H)} \left(\Delta|_{V^i}(v) - (Bf^i)_v \right) \phi^i_v \\
&= \langle \phi^i, \Delta|_{V^i} \rangle - \langle \phi^i, Bf^i \rangle.
\end{aligned}$$

Since the deletion set D^i from Step 2 is designed to guarantee that $\langle \phi^i, \Delta|_{V^i} \rangle \le \langle \phi^i, Bf^i \rangle$, we have that $\langle g^i, w^i \rangle \le 0$. □

From the above, we have verified that our algorithm is indeed captured by the MWU algorithm. Now, we derive the implication of this fact. Only for analysis, we define the average flow $\bar{f} = \frac{1}{T} \sum_{i=1}^{T} f^i \in \mathbb{R}^{E(H)}$ on H and the average U-boundary demand function $\overline{\Delta} = \frac{1}{T} \sum_{i=1}^{T} \Delta|_{V^i} \in \mathbb{R}^{V(H)}$ on H.

Lemma 4.5. *We have $\bar{f}$ ϵ-satisfies $\overline{\Delta}$ in H.*

Proof. Define $\bar{r} = \frac{1}{T} \sum_{i=1}^{T} r^i \in \mathbb{R}^{\mathcal{S}}$. First, we prove that $|\bar{r}_S| \le \alpha$ for all $S \in \mathcal{S}$. This is because

$$\pm \bar{r}_S = \frac{1}{T} \sum_{i \in [T]} \pm r^i_S = \frac{1}{T} \sum_{i \in [T]} g^i_{S,\pm} \le \alpha$$

where the last inequality is precisely the guarantee of the MWU algorithm from Theorem 4.3. Next observe that the excess is

$$\overline{\Delta}^{\bar{f}}(S) = \overline{\Delta}(S) - \bar{f}(S) = \bar{r}_S \delta_H(S).$$

Therefore, we have that $|\overline{\Delta}^{\bar{f}}(S)| \le \alpha \delta_H(S)$ for all $S \in \mathcal{S}$. Since $\mathcal{S}$ is a congestion approximator, it follows by Theorem 3.2 that

$$|\overline{\Delta}^{\bar{f}}(S)| \le \gamma_{\mathcal{S}} \alpha \delta_H(S) = \epsilon \delta_H(S)$$

for all $S \subseteq V(H)$. This precisely means that $\bar{f}$ ϵ-satisfies $\overline{\Delta}$. □

Now, we are ready to prove Item 2 of Theorem 4.1. By Lemma 4.5, there exists a flow $\bar{f}_{aug}$ in H with congestion ϵ such that $\bar{f}_{sat} := \bar{f} + \bar{f}_{aug}$ satisfies $\overline{\Delta}$. We define f'_{sat} as the restriction of $\bar{f}_{sat}$ into $G\{U'\}$. That is, for each new boundary vertex $x_e \in N\langle U' \rangle \setminus N\langle U \rangle$ where u is its unique neighbor, we set $f'_{sat}(x_e, u) = \bar{f}_{sat}(e)$. For every other edge $e \in E(G\{U'\})$, we set $f'_{sat}(e) = \bar{f}_{sat}(e)$. Let Δ' be a U'-boundary demand function where, for each U'-boundary vertex $v \in N\langle U' \rangle$, we set $\Delta'(v) = f'_{sat}(v)$ as the net flow out of v via f'_{sat}.

Lemma 4.6. *We have*

1. f'_{sat} *is a flow in* $G\{U'\}$ *with congestion at most* $(1+\epsilon)$ *that satisfies* Δ'.
2. Δ' *is a* U'*-boundary demand function where*

$$\Delta'(v) = (1-\beta)\deg_{G\{U'\}}(v) \qquad \text{for all old boundary vertices } v \in N\langle U'\rangle \cap N\langle U\rangle$$
$$|\Delta'(v)| \leq (1+\epsilon)\deg_{G\{U'\}}(v) \qquad \text{for all new boundary vertices } v \in N\langle U'\rangle \setminus N\langle U\rangle$$

Proof. (1) As f'_{sat} is a restriction of $\bar{f}_{sat}$ into $G\{U'\}$, then the congestion of f'_{sat} is at most that of $\bar{f}_{sat}$ which is $(1+\epsilon)$. To see why f'_{sat} satisfies Δ', we have that $\Delta'(v) = f'_{sat}(v)$ for all U'-boundary vertex $v \in N\langle U'\rangle$ by construction. For non-boundary vertex $v \in U' \backslash t$, we have $f'_{sat}(v) = \bar{f}(v) = 0 = \Delta'(v)$. So $f'_{sat}(v) = \Delta'(v)$ for all $v \neq t$. This implies that $f'_{sat}(t) = \Delta'(t)$ too and so f'_{sat} satisfies Δ'.

(2) For each new boundary vertex $v \in N\langle U'\rangle \setminus N\langle U\rangle$, we have $\Delta'(v) = f'_{sat}(v)$ and so $|\Delta'(v)| \leq (1+\epsilon)\deg_{G\{U'\}}(v)$ because f'_{sat} has congestion $(1+\epsilon)$ in $G\{U'\}$. For each old boundary vertex $v \in N\langle U'\rangle \cap N\langle U\rangle$, we have $\Delta'(v) = f'_{sat}(v) = \bar{f}_{sat}(v)$. As $\bar{f}_{sat}$ satisfies $\overline{\Delta}$, we have $\bar{f}_{sat}(v) = \overline{\Delta}(v)$. But $\overline{\Delta}(v) = (1-\beta)\deg_{G\{U\}}(v)$ as, for every i, $\Delta|_{V^i}(v) = (1-\beta)\deg_{G\{U\}}(v)$ for every $v \notin P$. Therefore, $\Delta'(v) = (1-\beta)\deg_{G\{U'\}}(v)$. □

This proves Item 2 of Theorem 4.1. It remains to prove Item 1 of Theorem 4.1.

Lemma 4.7. $\delta_{G\{U\}}(P) \leq \Delta(P)$.

Proof. First observe that $\delta_H(D^0) \leq \Delta(D^0)$ because every time we remove a set S from V^0, we have $\delta_H(S) < \Delta|_{V^0}(S)$ and we can charge $\delta_H(S)$ to the decrease of $\Delta|_{V^0}(S)$. Next, the sets D^i for $i \geq 1$ satisfy $\delta_H(D^i) \leq \Delta|_{V^{i-1}}(D^i)$, so

$$\delta_{G\{U\}}(P) = \delta_H(P) \leq \sum_{i\geq 0} \delta_H(D^i) \leq \Delta(D^0) + \sum_{i\geq 1} \Delta|_{V^{i-1}}(D^i) = \Delta(P).$$

□

Corollary 4.8. $\delta_G(U') \leq \delta_G(U) - \beta \cdot \delta_G(P, V \setminus U)$.

Proof. We have $\delta_{G\{U\}}(P) = \delta_G(P, U')$ and $\Delta(P) = (1-\beta)\delta_G(V \setminus U, P)$. By adding $\delta_G(V \setminus U, U')$ into both sides of the inequality of Lemma 4.7, we have

$$\delta_G(V \setminus U, U') + \delta_G(P, U') \leq \delta_G(V \setminus U, U') + \delta_G(V \setminus U, P) - \beta\delta_G(V \setminus U, P)$$

which concludes the proof because $\delta_G(U') = \delta_G(V \setminus U, U') + \delta_G(P, U')$ and $\delta_G(V \setminus U, U') + \delta_G(V \setminus U, P) = \delta_G(V \setminus U) = \delta_G(U)$. □

This proves the correctness of Theorem 4.1.

4.3 Running Time

Here, we explain some implementation details and analyze the total running time. Computing the congestion approximator $\mathcal{S}$ takes $\tilde{O}(m)$ by Theorem 3.2. The step which ensures that no set in $\mathcal{S}$ contains t is at most $O(n \log n)$ time because t was contained in at most $O(\log n)$ sets S and the complement of S has size at most n.

Next, we explain how to implement the initialization of V^0 efficiently. Observe that, for any $S \in \mathcal{S}$, if $\Delta|_{V^0}(S) > \delta_H(S)$, then we set $V^0 \leftarrow V^0 \setminus S$ and then we have $\Delta|_{V^0}(S) = 0$. Otherwise, if $\Delta|_{V^0}(S) \leq \delta_H(S)$, then it remains so forever because $\Delta|_{V^0}(S)$ is monotonically decreasing when V^0 is a decremental set. In any case, for each $S \in \mathcal{S}$, we only need to compare $\Delta|_{V^0}(S)$ with $\delta_H(S)$ once, which takes time at most $O(|S| + |E_H(S, V(H))|)$. So the total time is $O(m \log n)$ because $\mathcal{S}$ can be partitioned into $O(\log n)$ layers of disjoint subsets by the second property of Theorem 3.2.

In round i of the main algorithm, computing f^i takes $O(m)$ time. Using the fact that $\mathcal{S}$ is a laminar family and $\mathcal{S}$ contains $O(n)$ sets, we can compute r^i_S for all $S \in \mathcal{S}$ in $O(n)$ time, and so we can compute the weights $w^{i+1}_{S,\circ}$ for all $S \in \mathcal{S}$, $\circ \in \{+, -\}$ in $O(n)$. The most technical step is Step 2 whose implementation details is shown at the end of the section.

Lemma 4.9. *The "deletion set" $D^i \subseteq V(H) \setminus t$ from Step 2 can be computed in $O(m + n \log n)$ time.*

In total, the running time is $\tilde{O}(m) + T \cdot O(m + n \log n)$ time. Recall that $m = |E(H)| = O(|E(G\{U\})|)$ where T is the number of rounds. So we conclude the running time analysis:

Lemma 4.10. *The total running time of the algorithm for Theorem 4.1 is at most $\tilde{O}(|E(G\{U\})|/\epsilon^2)$.*

4.4 Proof of Lemma 4.9

In this section, we show how to construction $D^i \subseteq V(H) \setminus t$ where

$$\text{if } D^i \neq \emptyset, \text{ then } \Delta|_{V^{i-1}}(D^i) > \delta_H(D^i) \tag{1}$$

$$\langle \phi^i, \Delta|_{V^i} \rangle = \langle \phi^i, \Delta|_{V^{i-1} \setminus D^i} \rangle \leq \langle \phi^i, Bf^i \rangle. \tag{2}$$

If $\langle \phi^i, \Delta|_{V^{i-1}} \rangle \leq \langle \phi^i, Bf^i \rangle$, then we simply set $D^i = \emptyset$, which trivially fulfills both conditions. For the remainder of the proof, we assume that $\langle \phi^i, \Delta|_{V^{i-1}} \rangle > \langle \phi^i, Bf^i \rangle$.

For real number x, define $V_{>x} = \{v \in V(H) : \phi^i_v > x\}$. Fix some large number $M > \max_{v \in N\{U\}} |\phi^i_v|$. We first prove the chain of relations

$$\int_{x=-M}^{M} \Delta|_{V^{i-1}}(V_{>x}) dx = \langle \phi^i, \Delta|_{V^{i-1}} \rangle > \langle \phi^i, Bf^i \rangle = \int_{x=-M}^{M} \delta_H(V_{>x}) dx. \tag{3}$$

We start with

$$\begin{aligned}
\int_{x=-M}^{M} \Delta|_{V^{i-1}}(V_{>x}) dx &= \int_{x=-M}^{M} \left(\sum_{v \in V(H)} \Delta|_{V^{i-1}}(v) \cdot \mathbf{1}\{\phi^i_v > x\} \right) dx \\
&= \sum_{v \in V(H)} \Delta|_{V^{i-1}}(v) \int_{x=-M}^{M} \mathbf{1}\{\phi^i_v > x\} dx \\
&= \sum_{v \in V(H)} \Delta|_{V^{i-1}}(v) (\phi^i_v - (-M)).
\end{aligned}$$

Since $\sum_{v\in V(H)} \Delta|_{V^{i-1}}(v) = 0$ by construction, this is equal to

$$\sum_{v\in V(H)} \Delta|_{V^{i-1}}(v)\, \phi^i_v = \langle \phi^i, \Delta|_{V^{i-1}}\rangle.$$

By definition of the flow f^i,

$$\begin{aligned}
\langle \phi^i, Bf^i\rangle &= \sum_{(u,v)\in E(H)} c_H(u,v)|\phi^i_u - \phi^i_v| \\
&= \sum_{(u,v)\in E(H)} c_H(u,v) \int_{x=-M}^{M} \mathbf{1}\{(u,v)\in \partial_H(V_{>x})\}dx \\
&= \int_{x=-M}^{M} \sum_{(u,v)\in E(H)} c_H(u,v)\mathbf{1}\{(u,v)\in \partial_H(V_{>x})\}dx \\
&= \int_{x=-M}^{M} \delta_H(V_{>x})dx.
\end{aligned}$$

Together with the assumption $\langle \phi^i, \Delta|_{V^{i-1}}\rangle > \langle \phi^i, Bf^i\rangle$, we obtain (3).

Let x^* be the largest value such that

$$\int_{x=-M}^{x^*} \Delta|_{V^{i-1}}(V_{>x})dx = \int_{x=-M}^{x^*} \delta_H(V_{>x})dx,$$

which must exist since $x^* = -M$ works. Next, we claim that we must have

$$\Delta|_{V^{i-1}}(V_{>x^*}) > \delta_H(V_{>x^*}). \tag{4}$$

Otherwise, for small enough $\epsilon > 0$ we would have $\int_{x=-M}^{x^*+\epsilon} \Delta|_{V^{i-1}}(V_{>x})dx \le \int_{x=-M}^{x^*+\epsilon} \delta_H(V_{>x})dx$, and since $\int_{x=-M}^{M} \Delta|_{V^{i-1}}(V_{>x})dx > \int_{x=-M}^{M} \delta_H(V_{>x})dx$, there is another choice of x^* between $x^*+\epsilon$ and M that achieves equality, a contradiction.

We now claim that $t \notin V_{>x^*}$. Otherwise, since $\Delta|_{V^{i-1}}(V(H)) = 0$ and $\Delta|_{V^{i-1}}(t)$ is the only negative entry, we would have $\Delta|_{V^{i-1}}(V_{>x^*}) \le 0$ which would violate (4). Since $t \notin V_{>x^*}$ and $\phi^i_t = 0$, we conclude that $x^* \ge 0$.

Let $\overline{\phi}^i = \min\{\phi^i, x^*\}$ as ϕ^i truncated to a maximum of x^*. Then, similar to (3), we obtain

$$\langle \overline{\phi}^i, \Delta|_{V^{i-1}}\rangle = \int_{x=-M}^{x^*} \Delta|_{V^{i-1}}(V_{>x})dx = \int_{x=-M}^{x^*} \delta_H(V_{>x})dx = \langle \overline{\phi}^i, Bf^i\rangle. \tag{5}$$

Define our deletion set as $D^i \triangleq V_{>x^*}$, so $t \notin D^i$ and Equation (1) follows from (4). We now prove the chain of relations

$$\langle \phi^i, \Delta|_{V^{i-1}\setminus D^i}\rangle = \langle \overline{\phi}^i, \Delta|_{V^{i-1}\setminus D^i}\rangle \le \langle \overline{\phi}^i, \Delta|_{V^{i-1}}\rangle = \langle \overline{\phi}^i, Bf^i\rangle \le \langle \phi^i, Bf^i\rangle,$$

which would fulfill Equation (2). For the first relation, if $\phi^i_v \ne \overline{\phi}^i_v$ then $v \in D^i$, which means that $\Delta|_{V^{i-1}\setminus D^i}(v) = 0$. For the second relation, we use $\overline{\phi}^i_t = \phi^i_t = 0$ to obtain

$$\begin{aligned}
\langle \overline{\phi}^i, \Delta|_{V^{i-1}\setminus D^i}\rangle = \sum_{v\in V(H)\setminus t} \overline{\phi}^i(v)\Delta|_{V^{i-1}\setminus D^i}(v) &= \sum_{v\in V(H)\setminus t} \overline{\phi}^i(v)\Delta|_{V^{i-1}}(v) - x^*\Delta|_{V^{i-1}}(D^i) \\
&= \langle \overline{\phi}^i, \Delta|_{V^{i-1}}\rangle - x^*\Delta|_{V^{i-1}}(D^i)
\end{aligned}$$

which is at most $\langle \overline{\phi}^i, \Delta|_{V^{i-1}} \rangle$ since $x^* \geq 0$. The third relation follows from (5). For the last relation, we have

$$\langle \overline{\phi}^i, Bf^i \rangle = \sum_{(u,v)\in E(H)} c_H(u,v)|\overline{\phi}_u^i - \overline{\phi}_v^i| \leq \sum_{(u,v)\in E(H)} c_H(u,v)|\phi_u^i - \phi_v^i| = \langle \phi^i, Bf^i \rangle.$$

This concludes Equation (2).

Finally, we claim the running time $O(m + n \log n)$. The only nontrivial step in the algorithm is computing x^*. We first sort the values ϕ_v^i in $O(n \log n)$ time. Then, by sweeping through the sorted list, we can compute $\Delta|_{V^{i-1}}(V_{>x}) - \delta_H(V_{>x})$ for all $x \in \{\phi_v^i : v \in V(H)\}$ in $O(m)$ time. The function $\Delta|_{V^{i-1}}(V_{>x}) - \delta_H(V_{>x})$ is linear between consecutive values of ϕ_v^i, so we can locate the largest value x^* for which the function is 0.

5 From Almost Fair Cuts to Fair Cuts

In this section, we prove Theorem 1.3 using the AlmostFair subroutine.

5.1 Algorithm

Let (G, s, t, α) be the input and we want to compute a $(1 + \alpha)$-fair (s,t)-cut in G. Let $c_{\min} = \min_e c(e)$ and let $C = c(E)/c_{\min}$ be the ratio between total capacity and the minimum capacity. Recall that we assume $C = \text{poly}(n)$. We also assume $\alpha \geq \frac{1}{\text{poly}(n)}$, otherwise we could solve the problem using exact max flow algorithms.

Our algorithm runs in iterations where in iteration j we compute $(S^j, T^j, k^j, \overline{\mathsf{def}}^j)$ where (S^j, T^j) is an (s,t)-cut where $s \in S^j$ and $t \in T^j$, $k^j \in \mathbb{Z}_{\geq 0}$, and $\overline{\mathsf{def}}^j \in \mathbb{R}_{\geq 0}$ represents an upper bound of the *deficit* which will be explained in the analysis. Define $\beta = \Theta(\alpha/\log n)$ and $\epsilon = \beta/16$. Initially, (S^0, T^0) is an arbitrary (s,t)-cut, $\overline{\mathsf{def}}^0 = \delta_G(S^0, T^0)$, and $k^0 = 0$.

While $\overline{\mathsf{def}}^j > \beta c_{\min}$, do the following starting from $j = 0, 1, 2, \ldots$

1. Compute

$$(P_s^j, S^j \setminus P_s^j) = \textsc{AlmostFair}(G, S^j, s, \epsilon, (k^j + 1)\beta), \text{ and}$$
$$(P_t^j, S^j \setminus P_t^j) = \textsc{AlmostFair}(G, T^j, t, \epsilon, (k^j + 1)\beta)$$

   by calling Theorem 4.1.

2. If $\max\{\delta_G(P_s^j, T^j), \delta_G(P_t^j, S^j)\} \leq \overline{\mathsf{def}}^j/40$, then we update

$$k^{j+1} = k^j + 1, \text{ and}$$
$$\overline{\mathsf{def}}^{j+1} = \overline{\mathsf{def}}^j/2.$$

   Then, we set $T^{j+1} = T^j \setminus P_t^j$ and $S^{j+1} = V \setminus T^{j+1}$.[11]

[11] We could also symmetrically set $S^{j+1} = S^j \setminus P_s^j$ and $T^{j+1} = V \setminus S^{j+1}$. This choice is arbitrary.

3. Else, if $\max\{\delta_G(P_s^j, T^j), \delta_G(P_t^j, S^j)\} > \overline{\mathsf{def}}^j/40$, then we update

$$k^{j+1} = k^j, \text{ and}$$
$$\overline{\mathsf{def}}^{j+1} = (1 - \beta/80)\overline{\mathsf{def}}^j$$

   If $\delta_G(P_s^j, T^j) > \overline{\mathsf{def}}^j/40$, then, we set $S^{j+1} = S^j \setminus P_s^j$ (and $T^{j+1} = V \setminus S^{j+1}$). Otherwise, we set $T^{j+1} = T^j \setminus P_t^j$ (and $S^{j+1} = V \setminus T^{j+1}$).

After the while loop, we return (S^j, T^j) as a $(1+\alpha)$-fair (s,t)-cut. As $\overline{\mathsf{def}}^0 \leq c(E)$ we have that $\overline{\mathsf{def}}^j \leq (1-\beta/80)^j c(E)$ for all j. So there are at most $O(\log(C/\beta)/\beta)$ iterations before $\overline{\mathsf{def}}^j < \beta \cdot c_{\min}$. Therefore, the algorithm takes $O(\log(C/\beta)/\beta) \times \tilde{O}(m/\epsilon^2) = \tilde{O}(m/\alpha^3)$ total time by Theorem 4.1. It remains to show the correctness of the algorithm.

5.2 Analysis

For convenience, whenever we refer to an edge $(a,b) \in E(A,B)$, we mean $a \in A$ and $b \in B$. Only for the analysis, we construct a feasible flow f^j in G on each iteration j, and ensure that f^j satisfies the following two properties:

1. Define the *deficit* of flow f^j as $\mathsf{def}^j(f^j) = \sum_{(u,v)\in E(S^j,T^j)} \max\{0, (1-k^j\beta)c(u,v) - f^j(u,v)\}$. We maintain an invariant that $\mathsf{def}^j(f^j) \leq \overline{\mathsf{def}}^j$.

2. For all $R \subseteq V \setminus \{s,t\}$, we require that $|f^j(R)| \leq \epsilon\delta_G(R)$. Equivalently, f^j ϵ-satisfies an (s,t)-demand function in G.

In words, each cut edge $(u,v) \in E(S^j, T^j)$ contributes to the deficit of flow f^j when the flow in f^j *from* u *to* v is less than $(1-k^j\beta)$-fraction of its capacity. With our definition of deficit in Property 1, we have that the cut is fair whenever the deficit is very small:

Proposition 5.1. *If* $\overline{\mathsf{def}}^j < \beta c_{\min}$*, then* (S^j, T^j) *is a* $(1+\alpha)$*-fair* (s,t)*-cut.*

Proof. First we claim that $k^j = O(\log n)$. This is because everytime k^j increments, $\overline{\mathsf{def}}$ is halved. So at the end of the algorithm, we have $\frac{\beta c_{\min}}{2} < \overline{\mathsf{def}}^j < c(E)/2^{k^j}$, which implies $k^j = O(\log(C/\beta)) = O(\log n)$. Now, by the assumption and Property 1, for all $(u,v) \in E(S^j, T^j)$, we have $(1 - k^j\beta)c(u,v) - f^j(u,v) < \beta \cdot c_{\min}$ and so

$$f^j(u,v) > (1-(k^j+1)\beta)c(u,v) \geq \frac{1}{(1+\alpha/2)}c(u,v)$$

where the last inequality is because $k^j = O(\log n)$ and we can set the constant in $\beta = \Theta(\alpha/\log n)$ to be small enough. Since f^j ϵ-satisfies an (s,t)-demand function, by the observation below Fact 3.1, there exists f_{aug} with congestion ϵ such that $f^* = f^j + f_{aug}$ is an (s,t)-flow. Now, we have that for all $(u,v) \in E(S^j, T^j)$,

$$f^*(u,v) \geq f^j(u,v) - \epsilon c(u,v) \geq \frac{1}{(1+\alpha)}c(u,v)$$

because $\epsilon = \beta/16 = \Theta(\alpha/\log n)$ and the constant in it is small enough. Therefore, f^* certifies that (S^j, T^j) is a $(1+\alpha)$-fair (s,t)-cut. □

Initially, we set f^0 as the zero flow, which satisfies both properties since $\overline{\mathsf{def}}^0 = \delta_G(S^0, T^0)$. Property 2 will help us show the following inductive step, which would conclude the correctness of Theorem 1.3.

Lemma 5.2. *Suppose there exists a feasible flow f^j satisfying Properties 1 and 2 for j. Then, we can construct a feasible flow f^{j+1} satisfying Properties 1 and 2 for $j+1$.*

We analyze the two cases based on $\max\{\delta_G(P_s^j, T^j), \delta_G(P_t^j, S^j)\}$ in the subsections below.

Case 1: $\max\{\delta_G(P_s^j, T^j), \delta_G(P_t^j, S^j)\} \leq \overline{\mathsf{def}}^j/40$

Let $S'^j = S^j \setminus P_s^j$. By the guarantees of AlmostFair$(G, S^j, s, \epsilon, (k^j+1)\beta)$, let Δ_s be the S'^j-boundary demand function satisfied by a flow f_s in $G\{S'^j\}$ with congestion $(1+\epsilon)$. As $k^{j+1} = k^j + 1$ in this case, by Theorem 4.1, we have $f_s(v) = \Delta_s(v) = (1 - k^{j+1}\beta)\deg_{G\{S^j\}}(v)$ for all old boundary vertices $v \in N\langle S^j\rangle \cap N\langle S'^j\rangle$. Let T'^j, Δ_t, f_t be defined symmetrically. From f_s and f_t, we will construct a new flow f^{j+1} in three steps.

Step 1: Concatenate. Get $\hat{f}$. Consider the "concatenation" of f_s and f_t, denoted by f_{st}, where we reverse the direction of f_s so that the flow is sent out of s. The concatenated flow f_{st} is on the graph $G\{S'^j\} \cup G\{T'^j\}$ where the two graphs share $N\langle S'^j\rangle \cap N\langle T'^j\rangle$ as common boundary vertices. Now, we want to define a flow $\hat{f}$ on G that corresponds to f_{st} in a natural way. See Figure 1.

1. For each edge $e \in E(G[S'^j]) \cup E(G[T'^j])$ in the "interior" of S'^j or T'^j, we set $\hat{f}(e) = f_{st}(e)$.
2. For each common boundary vertex $x_e \in N\langle S'^j\rangle \cap N\langle T'^j\rangle$ where $e = (u, v) \in E(S'^j, T'^j)$, we have $f_{st}(u, x_e) = f_{st}(x_e, v) = (1 - k^{j+1}\beta)c(e)$ and so we set $\hat{f}(e) = (1 - k^{j+1}\beta)c(e)$.
3. For each new boundary vertex $x_e \in (N\langle S'^j\rangle \setminus N\langle S^j\rangle) \cup (N\langle T'^j\rangle \setminus N\langle T^j\rangle)$ where $e = (u, v) \in E(S'^j, P_s^j) \cup E(T'^j, P_t^j)$, we set $\hat{f}(e) = f_{st}(u, x_e)$.
4. For each old boundary vertex $x_e \in N\langle S^j\rangle \cap N\langle T^j\rangle$ incident to the pruned set P_s^j or P_t^j on one side, i.e., $e = (u, v) \in E(S'^j, P_t^j) \cup E(T'^j, P_s^j)$, we set $\hat{f}(e) = f_{st}(u, x_e)$.
5. For each old boundary vertex $x_e \in N\langle S^j\rangle \cap N\langle T^j\rangle$ incident to the pruned set P_s^j or P_t^j on both sides, i.e., $e = (u, v) \in E(P_s^j, P_t^j)$, we set $\hat{f}(e) = 0$.
6. For each edge in the "interior" of P_s^j or P_t^j, we set $\hat{f}(e) = 0$.

By construction, $\hat{f}$ satisfies some demand function $\hat{\Delta}$ where $\hat{\Delta}(v) = 0$ for $v \notin \{s, t\} \cup V(P_s^j) \cup V(P_t^j)$.

Step 2: Remove Flow Paths Through New Boundaries. Get $\hat{f}'$. Take a path decomposition of $\hat{f}$ in G, and then remove all paths starting or ending at vertices in $V(P_s^j) \cup V(P_t^j)$; let the resulting flow be $\hat{f}'$, which satisfies some demand function that is only nonzero at s, t. That is, $\hat{f}'$ is an (s, t)-flow. Note that $\hat{f}'$ still has congestion at most $(1+\epsilon)$.

Step 3: Truncate to a Feasible Flow. Get f^{j+1}. Finally, for any edges congested by more than 1 in $\hat{f}'$, lower the flow along that edge to congestion exactly 1. We define f^{j+1} as the resulting flow.

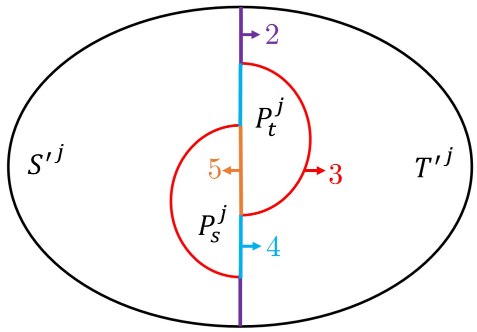

Figure 1: A diagram indicating the cases for defining $\hat{f}$ from f_{st}.

Proving Properties of f^{j+1}. Since f^{j+1} is obtained from the (s,t)-flow $\hat{f}'$ by removing a flow of congestion at most ϵ, Property 2 is satisfied. Now, we prove Property 1. We write the deficit of f^{j+1} as follows

$$\begin{aligned}
&\mathsf{def}^{j+1}(f^{j+1}) \\
&= \sum_{e\in E(S^{j+1},T^{j+1})} \max\{0,(1-k^{j+1}\beta)c(e)-f^{j+1}(e)\} \\
&\leq \sum_{e\in E(S^{j+1},T^{j+1})} \Big(\max\{0,(1-k^{j+1}\beta)c(e)-\hat{f}(e)\}+|\hat{f}(e)-\hat{f}'(e)|+|\hat{f}'(e)-f^{j+1}(e)|\Big) \\
&= \sum_{e\in E(S^{j+1},T^{j+1})} \max\{0,(1-k^{j+1}\beta)c(e)-\hat{f}(e)\}+ \\
&\sum_{e\in E(S^{j+1},T^{j+1})} |\hat{f}(e)-\hat{f}'(e)|+\sum_{e\in E(S^{j+1},T^{j+1})} |\hat{f}'(e)-f^{j+1}(e)|
\end{aligned}$$

Now, we bound each of the three terms above. We use the fact $T^{j+1}=T'^j$ and $S^{j+1}=S'^j\cup P_s^j\cup P_t^j$.

For the first term, we consider the concatenated flow $\hat{f}$. We have $\hat{f}(e)=(1-k^{j+1}\beta)c(e)$ for each old boundary edge $e\in E(S'^j,T'^j)$. So, the first term is bounded by

$$\begin{aligned}
\sum_{e\in E(S'^j\cup P_s^j\cup P_t^j,T'^j)} \max\{0,(1-k^{j+1}\beta)c(e)-\hat{f}(e)\} &\leq \sum_{e\in E(P_s^j\cup P_t^j,T'^j)} (1-k^{j+1}\beta)c(e)-\hat{f}(e) \\
&\leq ((1-k^{j+1}\beta)+(1+\epsilon))\cdot\delta(P_s^j\cup P_t^j,T'^j) \\
&\leq (2+\epsilon)\cdot\delta(P_s^j\cup P_t^j,T'^j)
\end{aligned}$$

where the second inequality is because $\hat{f}$ has $(1+\epsilon)$ congestion.

For the second term, consider the flow $\hat{f}'$ obtained by the flow-path removal. We rewrite the second term as

$$\sum_{e\in E(P_s^j\cup P_t^j,T'^j)} |\hat{f}(e)-\hat{f}'(e)|+\sum_{e\in E(S'^j,T'^j)} |\hat{f}(e)-\hat{f}'(e)|.$$

Trivially, we have

$$\sum_{e\in E(P_s^j\cup P_t^j,T'^j)} |\hat{f}(e)-\hat{f}'(e)|\leq(1+\epsilon)\delta(P_s^j\cup P_t^j,T'^j)$$

because the flow has congestion $(1+\epsilon)$. Now, we claim that

$$\sum_{e \in E(S'^j, T'^j)} |\hat{f}(e) - \hat{f}'(e)| \leq \sum_{e \in E(P_s^j \cup P_t^j, S'^j \cup T'^j)} |\hat{f}(e) - \hat{f}'(e)| \leq (1+\epsilon)\delta(P_s^j \cup P_t^j, S'^j \cup T'^j).$$

To see this, consider each flow-path P removed from $\hat{f}$ to obtain $\hat{f}'$. Observe that P cannot cross directly from T'^j to S'^j because, for *every* edge $e \in E(S'^j, T'^j)$, the flow is directed from S'^j to T'^j as $\hat{f}(e) = (1 - k^{j+1}\beta)c(e)$. Thus, between any two consecutive times that P crosses from S'^j to T'^j, P must have crossed from T'^j to $P_s^j \cup P_t^j$. Also, note that the first edge of P is from $E(P_s^j \cup P_t^j, S'^j \cup T'^j)$. Therefore, we can charge the flow changes in edges of $E(S'^j, T'^j)$ to the changes in edges of $E(P_s^j \cup P_t^j, S'^j \cup T'^j)$. So $\sum_{e \in E(S'^j, T'^j)} |\hat{f}(e) - \hat{f}'(e)| \leq \sum_{e \in E(P_s^j \cup P_t^j, S'^j \cup T'^j)} |\hat{f}(e) - \hat{f}'(e)|$ as claimed.

Finally, for the third term, we consider the truncated flow f^{j+1} with congestion at most 1 on all edges. Again, we have $\hat{f}'(e) - f^{j+1}(e) = 0$ for all $e \in E(S'^j, T'^j)$ because $0 \leq \hat{f}'(e) \leq (1 - k^{j+1}\beta)c(e)$. In particular, the congestion on e was already less than 1. Also, we have $|\hat{f}'(e) - f^{j+1}(e)| \leq \epsilon c(e)$ for any edges e as $\hat{f}'$ has congestion $1+\epsilon$. Hence, we have

$$\sum_{e \in E(S^{j+1}, T^{j+1})} |\hat{f}'(e) - f^{j+1}(e)| \leq \sum_{e \in E(P_s^j \cup P_t^j, T'^j)} \epsilon c(e) = \epsilon \delta(P_s^j \cup P_t^j, T'^j).$$

From the above bounds, we obtain

$$\mathsf{def}^{j+1}(f^{j+1}) \leq ((2+\epsilon) + (1+\epsilon) + (1+\epsilon) + \epsilon)\delta(P_s^j \cup P_t^j, S'^j \cup T'^j).$$

Now, write $\delta(P_s^j \cup P_t^j, S'^j \cup T'^j) = \delta(P_s^j, S'^j) + \delta(P_t^j, S'^j) + \delta(P_s^j, T'^j) + \delta(P_t^j, T'^j)$. Note that $\delta(P_t^j, T'^j) \leq \delta(P_t^j, S^j)$ and $\delta(P_s^j, S'^j) \leq \delta(P_s^j, T^j)$ by the guarantee of AlmostFair. Trivially, we also have $\delta(P_t^j, S'^j) \leq \delta(P_t^j, S^j)$ and $\delta(P_s^j, T'^j) \leq \delta(P_s^j, T^j)$. But we have $\delta(P_t^j, S^j), \delta(P_s^j, T^j) \leq \overline{\mathsf{def}}^j/40$ by the assumption of this case. So we have, as $\epsilon \leq 1/4$,

$$\mathsf{def}^{j+1}(f^{j+1}) \leq (4+4\epsilon) \cdot 4 \cdot \frac{\overline{\mathsf{def}}^j}{40} \leq \overline{\mathsf{def}}^j/2 = \overline{\mathsf{def}}^{j+1}$$

fulfilling Property 1.

Case 2: $\max\{\delta_G(P_s^j, T^j), \delta_G(P_t^j, S^j)\} > \overline{\mathsf{def}}^j/40$

In this case, we set f^{j+1} as the same old flow f^j. So Property 2 of f^{j+1} trivially continues to hold. For Property 1, assume without loss of generality the case $\delta_G(P_t^j, S^j) > \mathsf{def}^j/40$, so $T^{j+1} = T^j \setminus P_t^j$. (The case $\delta_G(P_s^j, T^j) > \mathsf{def}^j/40$ is symmetric, so we omit it.) As $f^{j+1} = f^j$ and $k^{j+1} = k^j$, we have

$$\begin{aligned}
&\mathsf{def}^{j+1}(f^{j+1}) \\
&= \sum_{e \in E(S^{j+1}, T^{j+1})} \max\{0, (1 - k^j\beta)c(e) - f^j(e)\} \\
&= \mathsf{def}^j(f^j) - \sum_{e \in E(S^j, P_t^j)} \max\{0, (1 - k^j\beta)c(e) - f^j(e)\} + \sum_{e \in E(P_t^j, T^{j+1})} \max\{0, (1 - k^j\beta)c(e) - f^j(e)\}.
\end{aligned}$$

For the second term (without the minus sign), we can lower bound it as

$$\geq \sum_{e \in E(S^j, P_t^j)} (1 - k^j\beta)c(e) - f^j(e) = (1 - k^j\beta)\delta(S^j, P_t^j) - f^j(S^j, P_t^j).$$

For the third term, we can upper bound it as

$$\leq \sum_{e \in E(P_t^j, T^{j+1})} c(e) - f^j(e) = \delta(P_t^j, T^{j+1}) - f^j(P_t^j, T^{j+1}).$$

where the first inequality is because $0 \leq c(e) - f^j(e)$ as f^j is feasible. Putting these together, we have

$$\mathsf{def}^{j+1}(f^{j+1}) \leq \mathsf{def}^j(f^j) - \left((1 - k^j\beta)\delta(S^j, P_t^j) - \delta(P_t^j, T^{j+1})\right) + \left(f^j(S^j, P_t^j) - f^j(P_t^j, T^{j+1})\right).$$

That is, the increase in deficit can be upper bounded as follows. It will decrease proportional to $(1 - k^j\beta)\delta(S^j, P_t^j) - \delta(P_t^j, T^{j+1})$ which is related cut size. It may increase proportional to $f(S^j, P_t^j) - f(P_t^j, T^{j+1})$ which is related to flow.

For the decrease caused by cut size, AlmostFair$(G, T^j, t, \epsilon, (k^j{+}1)\beta)$ guarantees that $\delta(P_t^j, T^{j+1}) \leq (1{-}(k^j{+}1)\beta)\delta(S^j, P_t^j)$. So the deficit must decrease by at least $\left((1 - k^j\beta) - (1 - (k^j + 1)\beta)\right)\delta(S^j, P_t^j) \geq \beta\delta(S^j, P_t^j)$. For the increase caused by flow, we have that $f^j(S^j, P_t^j) - f^j(P_t^j, T^{j+1}) = f^j(S^j, P_t^j) + f^j(T^{j+1}, P_t^j) = -f^j(P_t^j)$ is exactly the net flow of f^j into P_t^j. As $|f^j(P_t^j)| \leq \epsilon\delta_G(P_t^j)$ by Property 2 on P_t^j, we now have

$$\mathsf{def}^{j+1}(f^{j+1}) \leq \mathsf{def}^j(f^j) - \beta\delta(S^j, P_t^j) + \epsilon\delta_G(P_t^j).$$

Observe that $\delta_G(P_t^j) = \delta_G(S^j, P_t^j) + \delta_G(P_t^j, T^{j+1})$ but $\delta(P_t^j, T^{j+1}) \leq \delta(S^j, P_t^j)$ by AlmostFair again. So $\epsilon\delta_G(P_t^j) \leq 2\epsilon\delta_G(S^j, P_t^j) \leq \frac{\beta}{2}\delta_G(S^j, P_t^j)$ because $\epsilon \leq \beta/4$. Therefore,

$$\mathsf{def}^{j+1}(f^{j+1}) \leq \mathsf{def}^j(f^j) - \frac{\beta}{2}\delta(S^j, P_t^j) \leq (1 - \frac{\beta}{80})\mathsf{def}^j(f^j) = \overline{\mathsf{def}}^{j+1}$$

because $\delta_G(S^j, P_t^j) > \mathsf{def}^j/40$ by our initial assumption.

6 Approximate Isolating Cuts and Steiner Cut

The focus of this section is to compute approximate isolating cuts and show its application in the Steiner mincut problem.

6.1 Approximate Minimum Isolating Cuts

The approximate minimum isolating cuts problem is defined below.

Definition 6.1. Given an undirected graph $G = (V, E)$ with non-negative edge weights and a set of terminals $T \subseteq V$, a cut $\emptyset \subset S \subset V$ is said to be an *isolating cut* for a terminal $t \in T$ if $T \cap S = \{t\}$. A *minimum* isolating cut for t is a minimum value cut among all the isolating cuts for t. Similarly, a $(1 + \epsilon)$-approximate minimum isolating cut for t is an isolating cut for t whose value is at most $(1 + \epsilon)$ times that of a minimum isolating cut for t.

Below is our main theorem. We state our result in general before plugging in the current best runtime from Theorem 1.3.

Theorem 6.2. *We can compute $(1+\epsilon)$ approximate minimum isolating cuts in $\tilde{O}(m)$ time.*

More precisely, fix any $\epsilon < 1$. Given an undirected graph $G = (V, E)$ on m edges and n vertices with non-negative edge weights and a set of terminals $T \subseteq V$, there is an algorithm that outputs a $(1+\epsilon)$-approximate minimum isolating cut S_t for every terminal $t \in T$ in $O(m)$ time plus a set of $(1+\gamma)$-fair (s,t)-cut calls on undirected graphs that collectively contain $O(m \log |T|)$ edges and $O(n \log |T|)$ vertices, where $\gamma = \frac{\epsilon}{4\lceil \lg |T| \rceil}$. Moreover, the sets S_t are disjoint, and for each $t \in T$, the cut $(S_t, V \setminus S_t)$ is a t-sided $(1+\gamma)$-fair cut. Using Theorem 1.3 to compute $(1+\gamma)$-fair (s,t)-cuts, our algorithm for $(1+\epsilon)$-approximate minimum isolating cuts runs in $\tilde{O}(m/\epsilon^3)$ time.

Algorithm 1 $(1+\epsilon)$-approximate Minimum Isolating Cuts Algorithm on terminal set T

Arbitrarily order the terminals in $T = \{t_1, t_2, \dots, t_{|T|}\}$
Phase 1:
for $i = 1$ to $\lceil \lg |T| \rceil$ **do**
  $X_i \leftarrow \{v_j \in T : i^{\text{th}} \text{ bit in } j \text{ is } 1\}$
  $Y_i \leftarrow \{v_j \in T : i^{\text{th}} \text{ bit in } j \text{ is } 0\}$
  Use Theorem 1.3 to find a $(1+\gamma)$-fair (X_i, Y_i)-cut S_i
end for
Phase 2:
for every terminal $t \in T$ **do**
  Let S_t be the connected component containing t in $G \setminus \cup_i \delta S_i$, i.e., the graph where we delete all the edges in cuts δS_i for all i.
  G_t is obtained from G by contracting all vertices in $V \setminus S_t$ into a single vertex $\bar{s}_t$. {To implement this step efficiently, we construct a new graph that is identical to G_t instead of contracting G.}
  Find a $(1+\beta)$-approximate minimum $(t, \bar{s}_t)$-cut in graph G_t; call this cut C_t
end for
Return the cuts $\{C_t : t \in T\}$

To establish Theorem 6.2, we describe Algorithm 1 for finding $(1+\epsilon)$-approximate isolating cuts. First, we establish correctness of the algorithm by showing that the cut C_t returned by Algorithm 1 for a terminal $t \in T$ is indeed a $(1+\epsilon)$-approximate minimum isolating cut for T. The following claim establishes an approximate version of the standard uncrossing property of minimum cuts, and is crucial for the correctness of our algorithm.

Lemma 6.3. *Let A be a $(1+\alpha)$-approximate minimum isolating cut for some terminal t and let B be a $(1+\gamma)$-fair (X, Y)-cut where $X \cup Y = T$, $t \in X$, and $X \subseteq B$. Then, $A \cap B$ is a $(1+\alpha)(1+\gamma)$-approximate minimum isolating cut for t.*

Proof. First, note that since A is an isolating cut for t and $t \in X, X \subseteq B$, it follows that $A \setminus B$ does not contain any terminal and $A \cap B$ contains a single terminal t. Now, consider the two cuts A and $A \cap B$. Using the notation $\uplus$ for disjoint union, we can write

$$E(A, V \setminus A) = E(A \cap B, V \setminus (A \cup B)) \uplus E(A \cap B, B \setminus A) \uplus E(A \setminus B, V \setminus A)$$
$$E(A \cap B, V \setminus (A \cap B)) = E(A \cap B, V \setminus (A \cup B)) \uplus E(A \cap B, B \setminus A) \uplus E(A \cap B, A \setminus B).$$

Since the first two sets are identical, we only need to compare the third sets $E(A \setminus B, V \setminus A)$ and $E(A \cap B, A \setminus B)$. Since B is a $(1+\gamma)$-fair (X, Y)-cut, there is a feasible flow from X to Y that, for each edge in $E(B, V \setminus B)$, sends at least $\frac{1}{1+\gamma}$ times capacity in the direction from B to $V \setminus B$. Now, consider the flow on the subset of edges $E(A \cap B, A \setminus B) \subseteq E(B, V \setminus B)$. Since the flow must end at Y and since $Y \cap (A \setminus B) = \emptyset$, it follows that this flow must exit the set $A \setminus B$ on the edges in $E(A \setminus B, V \setminus (A \cup B))$. Thus,

$$\delta(A \cap B, A \setminus B) \leq (1+\gamma) \cdot \delta(A \setminus B, V \setminus (A \cup B)) \leq (1+\gamma) \cdot \delta(A \setminus B, V \setminus A).$$

It follows that $\delta(A \cap B) \leq (1+\gamma) \cdot \delta(A)$, which proves the lemma. □

Lemma 6.4. *For $\gamma = \frac{\epsilon}{4\lceil \lg |T| \rceil}$ and $\beta = \frac{\epsilon}{4}$, the cut C_t returned by Algorithm 1 is a $(1+\epsilon)$-approximate minimum isolating cut for every $t \in T$.*

Proof. Lemma 6.3 implies that in Algorithm 1, the minimum isolating cut of t in graph G_t, i.e., the minimum $t - \bar{s}_t$ cut, is a $(1+\gamma)^{\lceil \lg |T| \rceil}$-approximate minimum isolating cut of t in the input graph G. Since C_t is a $(1+\beta)$-approximate minimum $t - \bar{s}_t$ cut, it follows that C_t is a $(1+\gamma)^{\lceil \lg |T| \rceil} \cdot (1+\beta)$-approximate minimum isolating cut of t in the input graph G. Using the values of γ and β, we have

$$\left(1 + \frac{\epsilon}{4\lceil \lg |T| \rceil}\right)^{\lceil \lg |T| \rceil} \cdot \left(1 + \frac{\epsilon}{4}\right) \leq e^{\epsilon/4} \cdot e^{\epsilon/4} = e^{\epsilon/2} \leq 1 + \epsilon \text{ since } \epsilon < 1.$$

□

For the $(1+\beta)$-approximate mincut in Step 12, we can use Theorem 1.3 to compute a $(1+\gamma)$-fair cut, which is also a $(1+\beta)$-approximate mincut since $\gamma \leq \beta$. This also guarantees that the cut C_t is a t-sided $(1+\gamma)$-fair cut. Finally, it is clear from the algorithm that all cuts C_t are disjoint.

The runtime analysis is identical to that in [LP20], so we omit it for brevity.

6.2 $(1+\epsilon)$-approximate Minimum Steiner Cut

As an immediate application of our isolating cut result, we can solve the Steiner cut problem below efficiently.

Definition 6.5. Given an undirected graph $G = (V, E)$ with non-negative edge weights and a set of terminals $T \subseteq V$, a minimum Steiner cut is a cut of minimum value among all cuts $\emptyset \subset S \subset V$ that satisfy $\emptyset \subset S \cap T \subset T$.

Using Theorem 6.2, we give the following algorithm for finding a $(1+\epsilon)$-approximate minimum Steiner cut.

Theorem 6.6. *Given an undirected graph $G = (V, E)$ on m edges and n vertices and with non-negative edge weights and a set of terminals $T \subseteq V$, Algorithm 2 computes a $(1+\epsilon)$-minimum Steiner cut for T wuth probability at least $1 - 1/n$ in $\tilde{O}(m)$ time.*

Proof. Fix a minimum Steiner cut for the terminal set T and let S denote the side of this cut such that $|T \cap S| \leq |T \setminus S|$. Let $i \in [\lceil \lg |T| \rceil]$ such that $2^{i-1} \leq |S \cap T| < 2^i$. Then, T_{ij} contains exactly one vertex in $T \cap S$ with probability

$$|T \cap S| \cdot \frac{1}{2^i} \cdot \left(1 - \frac{1}{2^i}\right)^{|T \cap S| - 1} \geq 2^{i-1} \cdot \frac{1}{2^i} \cdot \left(1 - \frac{1}{2^i}\right)^{2^i} \geq \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{8}.$$

Algorithm 2 $(1+\epsilon)$-approximate minimum Steiner cut Algorithm on terminal set T

```
for i = 1 to ⌈lg |T|⌉ do
  for j = 1 to ⌈log_{8/7} n⌉ do
    T_ij is drawn i.i.d. from T where every vertex t ∈ T appears in T_ij with probability 1/2^i
    Use Theorem 6.2 to find isolating cuts S_ij = {S_t : t ∈ T_ij} for the terminal set T_ij
  end for
end for
Return arg min{δ(S) : S ∈ S_ij, i ∈ [⌈lg |T|⌉], j ∈ [⌈log_{8/7} n⌉]}
```

This implies that the probability that there is no index $j \in [\lceil \log_{8/7} n \rceil]$ such that T_{ij} contains exactly one terminal in $T \cap S$ is at most $1/n$, thereby establishing the correctness of the algorithm.

The running time bound follows from Theorem 6.2. □

7 Approximate Gomory-Hu Tree Algorithm

The main result in this section is the near-linear time algorithm for computing an approximate Gomory-Hu tree. In fact, our algorithm can solve a more general problem called approximate Gomory-Hu Steiner tree defined below. (The definition is copied verbatim from [LP21].)

Definition 7.1 (Approximate Gomory-Hu Steiner tree)**.** Given a graph $G = (V, E)$ and a set of terminals $U \subseteq V$, the $(1+\epsilon)$-approximate Gomory-Hu Steiner tree is a weighted tree T on the vertices U, together with a function $f : V \to U$, such that

- For all $s, t \in U$, consider the minimum-weight edge (u, v) on the unique $s - t$ path in T. Let U' be the vertices of the connected component of $T - (u, v)$ containing s. Then, the set $f^{-1}(U') \subseteq V$ is a $(1+\epsilon)$-approximate (s, t)-mincut, and its value is $w_T(u, v)$.

Our main result is stated below. Recall that we assume that the ratio between the largest and lowest edge weights are $\text{poly}(n)$.

Theorem 7.2. *Let G be a weighted, undirected graph, and let U be a subset of vertices. There is a randomized algorithm that w.h.p., outputs a $(1+\epsilon)$-approximate Gomory-Hu Steiner tree in $\tilde{O}(m \cdot \text{poly}(1/\epsilon))$ time.*

The algorithm and analysis are similar to those in [LP21], except we replace (exact) minimum isolating cuts with an approximate version, which requires overcoming a few more technical issues. For completeness, we redo all the proofs. We also restate Theorem 6.2 below in the form we precisely need.

Theorem 7.3. *Fix any $\epsilon < 1$. Given an undirected graph $G = (V, E)$ on m edges and n vertices with non-negative edge weights and a set of terminals $T \subseteq V$, there is an algorithm that outputs a $(1+\epsilon)$-approximate minimum isolating cut $S_t \subseteq V$ for every terminal $t \in T$ in $\tilde{O}(m/\epsilon^{O(1)})$ time. Moreover, the sets S_t are disjoint, and for each $t \in T$, the set S_t is a t-sided $(1+\gamma)$-fair $(\{t\}, T \setminus \{t\})$-cut.*

7.1 Cut Threshold Step Algorithm

We begin with the following "Cut Threshold Step" subroutine from [LP21], described in Algorithm 3 below. Loosely speaking, the algorithm inputs a source vertex s and a threshold W, and aims to find a large fraction of vertices whose mincut from s is approximately at most W.

Algorithm 3 $(1+\gamma)$-approximate "Cut Threshold Step" on inputs (G, U, W, s)

1: Initialize $D \leftarrow \emptyset$
2: **for** independent iteration $i \in \{0, 1, 2, \ldots, \lfloor \lg |U| \rfloor\}$ **do**
3: $\quad R^i \leftarrow$ sample of U where each vertex in $U \setminus \{s\}$ is sampled independently with probability $1/2^i$, and s is sampled with probability 1
4: $\quad$ Compute $(1 + \frac{\gamma}{2\lceil \lg |U| \rceil})$-approximate minimum isolating cuts $\{S^i_v : v \in R^i\}$ on inputs G and R^i with the additional guarantees of Theorem 7.3 (for large enough constant $c > 0$)
5: $\quad$ Let $\mathcal{F}^i$ be the family of sets S^i_v satisfying $\delta S^i_v \le (1+\gamma)W$, and let $D^i \leftarrow \bigcup_{S^i_v \in \mathcal{F}^i} S^i_v \cap U$
6: $\quad$ Let $\widetilde{R}^i \subseteq R^i$ be the set of all $v \in R^i$ satisfying $\delta S^i_v \le (1+\gamma)W$
7: **end for**
8: Let $i_{\max}$ be the index i maximizing $|D^i|$
9: Return $D \leftarrow D^{i_{\max}}$, $R \leftarrow \widetilde{R}^{i_{\max}}$, and $\mathcal{F} \leftarrow \mathcal{F}^{i_{\max}}$

Lemma 7.4. *For any i, each set S^i_v added to D^i satisfies $\lambda(s, v) \le (1+\gamma)W$.*

Proof. For each $v \in D^i$, the corresponding set S^i_v on line 5 contains v and not s, so $\lambda(s, v) \le \delta S^i_v \le (1+\gamma)W$. ☐

Lemma 7.5. *Let D^* be all vertices $v \in U \setminus s$ for which there exists an (s, v)-cut in G of weight at most W whose side containing v has at most $|U|/2$ vertices in U. Then, $\mathbb{E}[|D|] = \Omega(|D^*|/\log |U|)$.*

Proof. We will show that

$$\mathbb{E}\left[\sum_{i=0}^{\lfloor \lg |U| \rfloor} |D^i|\right] \ge \Omega(|D^*|), \tag{6}$$

which is sufficient, since the largest D^i will have at least $1/(\lfloor \lg |U| \rfloor + 1)$ fraction of the total size. Fix a vertex $v \in D^*$. For each $0 \le j \le \lceil \lg |U| \rceil$, define $C^j_v \subseteq V$ as the (s, v)-cut of weight at most $(1 + \frac{\gamma}{2\lceil \lg |U| \rceil})^j W$ that minimizes $|C^j_v \cap U|$, which must exist since $v \in D^*$. By construction, $|C^j_v \cap U|$ is decreasing in j.

We focus on a value j^* ($0 \le j^* < \lceil \lg |U| \rceil$) satisfying $|C^{j^*+1}_v \cap U| \ge |C^{j^*}_v \cap U|/2$, which is guaranteed to exist. Consider sampling iteration $i = \lfloor \lg |C^{j^*}_v \cap U| \rfloor$, where each vertex in $U \setminus \{s\}$ is sampled with probability $1/2^i$. With probability $\Omega(1/|C^{j^*}_v \cap U|)$, we have $C^{j^*}_v \cap R^i = \{v\}$, i.e., we sampled v and nothing else in $C^{j^*}_v \cap U$. If this occurs, then $C^{j^*}_v$ is a valid isolating cut separating v from $R^i \setminus \{v\}$. Since S^i_v is a $(1 + \frac{\gamma}{2\lceil \lg |U| \rceil})$-approximate minimum isolating cut, we have

$$\delta S^i_v \le \left(1 + \frac{\gamma}{2\lceil \lg |U| \rceil}\right) \delta C^{j^*}_v \le \left(1 + \frac{\gamma}{2\lceil \lg |U| \rceil}\right)^{j^*+1} W \le \left(1 + \frac{\gamma}{2\lceil \lg |U| \rceil}\right)^{\lceil \lg |U| \rceil} W \le e^{\gamma/2} W \le (1+\gamma)W,$$

so $S_v^i \cap U$ is added to D^i on line 5. By definition of $C_v^{j^*+1}$, we have $|S_v^i \cap U| \geq |C_v^{j^*+1} \cap U|$, which is at least $|C_v^{j^*} \cap U|/2$ by our choice of j^*. In other words, if $C_v^{j^*} \cap R^i = \{v\}$, which occurs with probability $\Omega(1/|C_v^{j^*} \cap U|)$, then v is "responsible" for adding at least $|C_v^{j^*} \cap U|/2$ vertices to D^i.

Thus, each vertex $v \in D^*$ is responsible for adding $\Omega(1)$ vertices in expectation to some D^i, which increases $\mathbb{E}\left[\sum_{i=0}^{\lfloor \lg |U| \rfloor} |D^i|\right]$ by $\Omega(1)$ in expectation. Finally, (6) follows by linearity of expectation over all $v \in D^*$. □

For our approximate Gomory-Hu tree algorithm, we actually need a bound on $\mathbb{E}[|D \cap D^*|]$, not $\mathbb{E}[|D|]$, since we want to remove D from U and claim that the size of the new D^* drops by a large enough factor. Unfortunately, it is possible that D is largely disjoint from D^*, so a bound on $\mathbb{E}[|D|]$ does not directly translate to a bound on $\mathbb{E}[|D \cap D^*|]$. Therefore, we wrap Algorithm 3 into another routine that achieves a good bound on $\mathbb{E}[|D \cap D^*|]$. We actually prove the stronger guarantee that D^* can be any *subset* of all vertices $v \in U \setminus s$ for which $\lambda(s, v) \leq W$, which is needed in our Gomory-Hu tree algorithm.

Algorithm 4 $(1+\gamma)$-approximate Gomory-Hu Steiner tree "step" on inputs (G, U_0, W_0, s)

Initialize $U \leftarrow U_0$
for $O(\log^3 n)$ sequential iterations **do**
  for independent iteration $j \in \{0, 1, 2, \ldots, \lceil \lg |U| \rceil - 1\}$ **do**
    Call Algorithm 3 on parameter $\frac{\gamma}{2\lceil \lg |U| \rceil}$ and inputs $(G, U, (1 + \frac{\gamma}{2\lceil \lg |U| \rceil})^j W_0, s)$ and let $(D_j, R_j, \mathcal{F}_j)$ be the output
  end for
  Update $U \leftarrow U \setminus \bigcup_j D_j$ for the values D_j computed on this sequential iteration
end for
Return an output $(D, R, \mathcal{F})$ selected uniformly at random out of the $O(\log^3 n \log |U|)$ calls to Algorithm 3.

Lemma 7.6. *Each set $S \in \mathcal{F}$ in the output $(D, R, \mathcal{F})$ of Algorithm 4 satisfies $\delta S \leq (1+\gamma) W_0$.*

Proof. By Lemma 7.4 applied to any $j \in \{0, 1, 2, \ldots, \lceil \lg |U| \rceil - 1\}$, each set $S \in \mathcal{F}_j$ satisfies

$$\delta S \leq \left(1 + \frac{\gamma}{2\lceil \lg |U| \rceil}\right) \cdot \left(1 + \frac{\gamma}{2\lceil \lg |U| \rceil}\right)^j W_0 \leq \left(1 + \frac{\gamma}{2\lceil \lg |U| \rceil}\right)^{\lceil \lg |U| \rceil} W_0 \leq e^{\gamma/2} W_0 \leq (1+\gamma) W_0.$$

So the same holds for the randomly chosen output $(D, R, \mathcal{F})$. □

Lemma 7.7. *Let D^* be an arbitrary set of vertices $v \in U \setminus s$ satisfying $\lambda(s, v) \leq W_0$. The output $(D, R, \mathcal{F})$ satisfies $\mathbb{E}[D \cap D^*] \geq \Omega(|D^*|/\log^4 n)$.*

Proof. We claim that after $O(\log^3 n)$ iterations of the main for loop, the set $D^* \cap U$ becomes empty. This would mean that D^* is contained in the union of all $O(\log^4 n)$ sets D_j computed over all iterations, so a random set D_j must contain a $\Omega(1/\log^4 n)$ fraction of D^* in expectation. For the rest of the proof, we prove this claim.

For each $0 \leq j \leq \lceil \lg |U| \rceil$, let D_j^* be all vertices $v \in U \setminus s$ for which $\lambda(s, v) \leq (1 + \frac{\gamma}{2\lceil \lg |U| \rceil})^j W_0$. By construction, $D^* \subseteq D_0^* \subseteq D_1^* \subseteq \cdots \subseteq D_{\lceil \lg |U| \rceil}^*$. We track the sets $D_j^* \cap U$ throughout the algorithm. Consider the set U at the beginning of one of the $O(\log^3 |U|)$ sequential iterations. We focus on a value j^* ($0 \le j^* < \lceil \lg |U| \rceil$) satisfying $|D^*_{j^*} \cap U| \ge |D^*_{j^*+1}|/2$. Consider iteration j^* of the inner for loop. By Lemma 7.4, we have $\lambda(s,v) \le (1 + \frac{\gamma}{2\lceil \lg |U| \rceil}) \cdot (1 + \frac{\gamma}{2\lceil \lg |U| \rceil})^{j^*} W_0 = (1 + \frac{\gamma}{2\lceil \lg |U| \rceil})^{j^*+1} W_0$, so in particular, $D_{j^*} \subseteq D^*_{j^*+1}$. By Lemma 7.5, we have $\mathbb{E}[|D_{j^*}|] \ge \Omega(|D^*_{j^*}|/\log |U|) \ge \Omega(|D^*_{j^*+1}|/\log |U|)$. Therefore, once we delete $\bigcup_j D_j$ at the end of this sequential iteration, the size of $D^*_{j^*+1}$ drops by factor $(1 - \Omega(1/\log |U|))$ in expectation.

In other words, on each sequential iteration, there exists j^* ($1 \le j^* \le \lceil \lg |U| \rceil$) for which the size of $D^*_j \cap U$ drops by factor $(1 - \Omega(1/\log |U|))$ in expectation. Since the other sets $D^*_{j'} \cap U$ can never increase in size, the product $\prod_{j=1}^{\lceil \lg |U| \rceil} |D^*_j \cap U|$ decreases by factor $(1 - \Omega(1/\log |U|))$ in expectation. Since the product is at most $|U|^{\lceil \lg |U| \rceil} \le 2^{O(\log^2 n)}$ initially, it follows that after $O(\log^3 n)$ sequential iterations, the product becomes zero w.h.p. Therefore, at the end of the algorithm, there exists j ($1 \le j^* \le \lceil \lg |U| \rceil$) with $D^*_j \cap U = \emptyset$. Since $D^* \subseteq D^*_j$, we also get $D^* \cap U = \emptyset$, which proves the claim. $\square$

7.2 The Algorithm for Approximating Gomory-Hu Steiner Tree

We present our approximate Gomory-Hu tree algorithm in Algorithm 5. It uses Algorithm 4 as a subroutine. See Figure 7.2 for a visual guide to the algorithm. Once again, the algorithm and analysis closely follow those in [LP21].

We require the lemma below for both running time and approximation guarantee analysis.

Lemma 7.8. *Each set $S \in \mathcal{F}$ satisfies $\delta_G S \le (1+\gamma)(1+10\epsilon)\lambda$ and $|S \cap U| \le 2|U|/3$.*

Proof. By Lemma 7.6 on the call to Algorithm 4 (line 6), each set $S \in \mathcal{F}$ satisfies $\delta_{G'} S \le (1 + \gamma) \cdot (1 + 10\epsilon)\lambda$, so $\delta_G S \le \delta_{G'} S \le (1+\gamma)(1+10\epsilon)\lambda$. We now prove the second statement. By construction, the cut $\partial_{G'} S$ has $|S \cap U|$ edges of weight $18\epsilon\lambda/|U|$ that were added to G'. Since $\partial_G S$ is a valid Steiner cut in G and the Steiner mincut is at least $(1-\epsilon)\lambda$, the cut $\partial_{G'} S$ has at least $(1-\epsilon)\lambda$ weight of edges from G. So $\delta_{G'} S \ge (1-\epsilon)\lambda + |S \cap U| \cdot 18\epsilon\lambda/|U|$. Suppose for contradiction that $|S \cap U| > 2|U|/3$; then, this becomes $\delta_{G'} S > (1-\epsilon)\lambda + 12\epsilon\lambda = (1+11\epsilon)\lambda$, which contradicts the earlier statement $\delta_{G'} S \le (1+\gamma)(1+10\epsilon)\lambda$. $\square$

7.3 Running Time Bound

Let $P(G, U, W)$ be the set of unordered pairs of distinct vertices whose mincut is at most W:

$$P(G,U,W) = \left\{ \{u,v\} \in \binom{U}{2} : \lambda_G(u,v) \le W \right\}.$$

In particular, we will consider its size $|P(G,U,W)|$, and show the following expected reduction:

Lemma 7.9. *For any W that is at most $(1+\epsilon)$ times the Steiner mincut of G, we have*

$$\mathbb{E}[|P(G_{\text{large}}, U_{\text{large}}, W)|] \le \left(1 - \Omega\left(\frac{1}{\log^4 n}\right)\right) |P(G,U,W)|,$$

where the expectation is taken over the random selection of s and the randomness in Algorithm 4.

Before we prove Lemma 7.9, we show how it implies progress on the recursive call for G_{large}.

Algorithm 5 $(1+\epsilon)$-approximate Gomory-Hu Steiner tree on inputs (G_0, U). Assume $\epsilon < 1/100$.

If $|U| = 1$, then return the trivial Gomory-Hu Steiner tree (T, f) where T is the empty tree on the single vertex $u \in U$, and $f(v) = u$ for all vertices v. Otherwise, if $|U| > 1$, then do the steps below.
$\gamma \leftarrow \epsilon^2 / \log^6 n$
$\lambda \leftarrow (1+\epsilon)$-approximate global Steiner mincut of G with terminals U, so that the Steiner mincut is in the range $[(1-\epsilon)\lambda, \lambda]$
$s \leftarrow$ uniformly random vertex in U
Construct graph G' by starting with G and adding an edge (s, u) of weight $18\epsilon\lambda/|U|$ for each $u \in U$
Call Algorithm 4 on parameter γ and inputs $(G', U, (1+10\epsilon)\lambda, s)$, and let $(D, R, \mathcal{F})$ be the output. Write $\mathcal{F} = \{S_v : v \in R\}$ where $v \in S_v$ for all $v \in R$.
Phase 1: Construct recursive graphs and apply recursion
for each $v \in R$ **do**
  Let G_v be the graph G with vertices $V \setminus S_v$ contracted to a single vertex x_v
  Let $U_v \leftarrow S_v \cap U$
  Recursively call (G_v, U_v) to obtain output (T_v, f_v)
end for
Let G_{large} be the graph G with (disjoint) vertex sets S_v contracted to single vertices y_v for all $v \in R$
Let $U_{\text{large}} \leftarrow U \setminus \bigcup_{v \in R}(S_v \cap U)$
Recursively call $(G_{\text{large}}, U_{\text{large}})$ to obtain $(T_{\text{large}}, f_{\text{large}})$
Phase 2: Merge the recursive Gomory-Hu Steiner trees
Construct T by starting with the disjoint union $T_{\text{large}} \cup \bigcup_{v \in R} T_v$ and, for each $v \in R$, adding an edge between $f_v(x_v) \in U_v$ and $f_{\text{large}}(y_v) \in U_{\text{large}}$ of weight $w(\partial_G S_v)$
Construct $f : V \to U$ by $f(v') = f_{\text{large}}(v')$ if $v' \in U_{\text{large}}$ and $f(v') = f_v(v')$ if $v' \in U_v$ for some $v \in R$
Return (T, f)

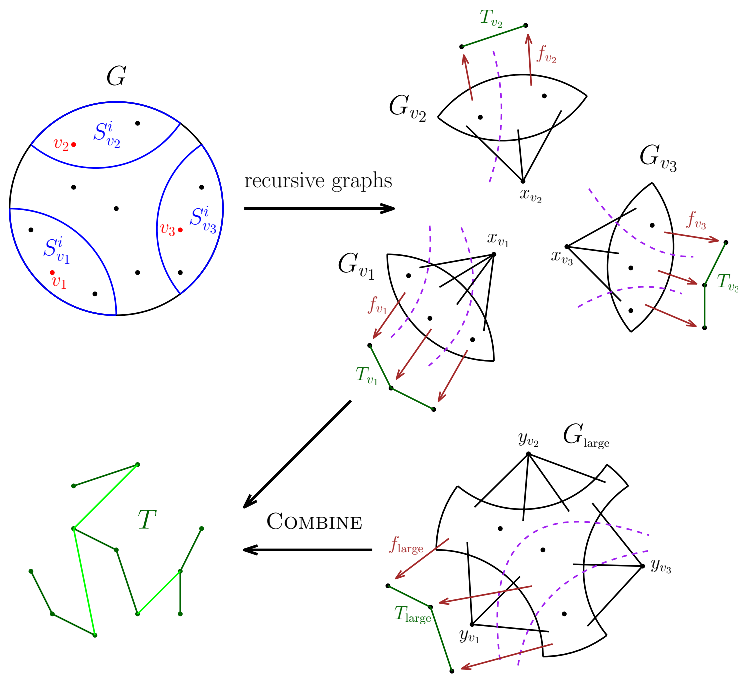

Figure 2: Recursive construction of G_{large} and G_v for $v \in R$. Here, $R = \{v_1, v_2, v_3\}$, denoted by red vertices on the top left. The dotted blue curves on the right mark the boundaries of the regions $f_{v_i}^{-1}(u) : u \in U_{v_i}$ and $f_{v_{\text{large}}}^{-1}(u) : u \in U_{\text{large}}$. The light green edges on the bottom left are the edges $(f_{v_i}(x_{v_i}), f_{\text{large}}(y_{v_i}))$ added on line 17.

Corollary 7.10. *Let λ_0 be the global Steiner mincut of G. W.h.p., after $\Omega(\log^5 n)$ recursive calls along G_{large} (replacing $G \leftarrow G_{\text{large}}$ each time), the global Steiner mincut of G is at least $(1+\epsilon)\lambda_0$ (where λ_0 is still the global Steiner mincut of the initial graph).*

Proof. Let $W = (1+\epsilon)\lambda_0$. Initially, we trivially have $|P(G,U,W)| \le \binom{|U|}{2}$. The global Steiner mincut can only increase in the recursive calls, since G_{large} is always a contraction of G, so W is always at most $(1+\epsilon)$ times the current Steiner mincut of G. By Lemma 7.9, the value $|P(G,U,W)|$ drops by factor $1-\Omega(\frac{1}{\log^4 n})$ in expectation on each recursive call, so after $\Omega(\log^5 n)$ calls, we have

$$\mathbb{E}[|P(G,U,W)|] \le \binom{|U|}{2} \cdot \left(1-\Omega\left(\frac{1}{\log^4 n}\right)\right)^{\Omega(\log^5 n)} \le \frac{1}{\text{poly}(n)}.$$

In other words, w.h.p., we have $|P(G,U,W)| = 0$ at the end, or equivalently, the Steiner mincut of G is at least $(1+\epsilon)\lambda_0$. □

Combining both recursive measures of progress together, we obtain the following bound on the recursion depth:

Lemma 7.11. *W.h.p., each path down the recursion tree of Algorithm 5 has $O(\log n)$ calls on a graph G_v, and between two consecutive such calls, there are $O(\epsilon^{-1}\log^6 n)$ calls on the graph G_{large}.*

Proof. For any $\Theta(\log^5 n)$ successive recursive calls down the recursion tree, either one call was on a graph G_v, or all $\Theta(\log^5 n)$ of them were on the graph G_{large}. In the former case, $|U|$ drops by a constant factor by Lemma 7.8, so it can happen $O(\log n)$ times total. In the latter case, by Corollary 7.10, the global Steiner mincut increases by factor $(1+\epsilon)$. Let $w_{\min}$ and $w_{\max}$ be the minimum and maximum weights in G, so that $\Delta = w_{\max}/w_{\min}$, which we assume to be $\text{poly}(n)$. Note that for any recursive instance (G', U') and any $s,t \in U'$, we have $w_{\min} \le \lambda_{G'}(s,t) \le w(\partial(\{s\})) \le n w_{\max}$, so the global Steiner mincut of (G', U') is always in the range $[w_{\min}, n w_{\max}]$. It follows that the global Steiner mincut can increase by factor $(1+\epsilon)$ at most $O(\epsilon^{-1}\log(n w_{\max}/w_{\min})) = O(\epsilon^{-1}\log n)$ times. Therefore, there are at most $O(\epsilon^{-1}\log^6 n)$ consecutive calls on G_{large} before a call on some G_v must occur. □

Lemma 7.12. *For an unweighted/weighted graph $G=(V,E)$, and terminals $U \subseteq V$, Algorithm 5 takes time $\tilde{O}(m\epsilon^{-1})$ plus calls to Theorem 7.3 with parameter $\gamma = \epsilon^2/\log^6 n$ on unweighted/weighted instances with a total of $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges.*

Proof. For a given recursion level, consider the instances $\{(G_i, U_i, W_i)\}$ across that level. By construction, the terminals U_i partition U. Moreover, the total number of vertices over all G_i is at most $n + 2(|U|-1) = O(n)$ since each branch creates 2 new vertices and there are at most $|U|-1$ branches.

To bound the total number of edges, we consider the unweighted and weighted cases separately, starting with the unweighted case. The total number of new edges created is at most the sum of weights of the edges in the final $(1+\epsilon)$-approximate Gomory-Hu Steiner tree. For an unweighted graph, this is $O(m)$ by the following well-known argument. Root the Gomory-Hu Steiner tree T at any vertex $r \in U$; for any $v \in U \setminus r$ with parent u, the cut $\partial\{v\}$ in G is a (u,v)-cut of value $\deg(v)$, so $w_T(u,v) \le (1+\epsilon)\lambda_G(u,v) \le (1+\epsilon)\deg(v)$. Overall, the sum of the edge weights in T is at most $(1+\epsilon)\sum_{v\in U}\deg(v) \le (1+\epsilon)\cdot 2m$.

For the weighted case, define a *parent* vertex in an instance as a vertex resulting from either (1) contracting $V \setminus S_v$ in some previous recursive G_v call, or (2) contracting a component containing a parent vertex in some previous recursive call. There are at most $O(\log n)$ parent vertices: at most $O(\log n)$ can be created by (1) since each G_v call decreases $|U|$ by a constant factor (Lemma 7.8), and (2) cannot increase the number of parent vertices. Therefore, the total number of edges adjacent to parent vertices is at most $O(\log n)$ times the number of vertices. Since there are $O(n)$ vertices in a given recursion level, the total number of edges adjacent to parent vertices is $O(n \log n)$ in this level. Next, we bound the number of edges not adjacent to a parent vertex by m. To do so, we first show that on each instance, the total number of these edges over all recursive calls produced by this instance is at most the total number of such edges in this instance. Let $P \subseteq V$ be the parent vertices; then, each G_v call has exactly $|E(G[S_v \setminus P])|$ edges not adjacent to parent vertices (in the recursive instance), and the G_{large} call has at most $|E(G[V \setminus P]) \setminus \bigcup_{v \in R} E(G[S_v \setminus P])|$, and these sum to $|E(G[V \setminus P])|$, as promised. This implies that the total number of edges not adjacent to a parent vertex at the next level is at most the total number at the previous level. Since the total number at the first level is m, the bound follows.

Therefore, there are $O(n)$ vertices and $\tilde{O}(m)$ edges in each recursion level. By Lemma 7.11, there are $O(\epsilon^{-1} \log^6 n)$ levels, for a total of $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges. In particular, the instances to the max-flow calls have $\tilde{O}(n\epsilon^{-1})$ vertices and $\tilde{O}(m\epsilon^{-1})$ edges in total. □

Finally, we prove Lemma 7.9, restated below.

Lemma 7.9. *For any W that is at most $(1 + \epsilon)$ times the Steiner mincut of G, we have*

$$\mathbb{E}[|P(G_{\text{large}}, U_{\text{large}}, W)|] \leq \left(1 - \Omega\left(\frac{1}{\log^4 n}\right)\right) |P(G, U, W)|,$$

where the expectation is taken over the random selection of s and the randomness in Algorithm 4.

Proof. Define D^* as the set of vertices $v \in U \setminus s$ for which there exists an (s, v)-cut in G of weight at most W whose side containing v has at most $|U|/2$ vertices in U. Let $P_{\text{ordered}}(G, U, W)$ be the set of ordered pairs $(u, v) : u, v \in V$ for which there exists a (u, v)-mincut of weight at most W with at most $|U|/2$ vertices in U on the side $S(u, v) \subseteq V$ containing u. We now state and prove the following four properties:

(a) For all $u, v \in U$, $\{u, v\} \in P(G, U, W)$ if and only if either $(u, v) \in P_{\text{ordered}}(G, U, W)$ or $(v, u) \in P_{\text{ordered}}(G, U, W)$ (or both).

(b) For each pair $(u, v) \in P_{\text{ordered}}(G, U, W)$, we have $u \in D^*$ with probability at least $1/2$,

(c) For each $u \in D^*$, there are at least $|U|/2$ vertices $v \in U$ for which $(u, v) \in P_{\text{ordered}}(G, U, W)$.

(d) Over the randomness in Algorithm 3 on $(G, U, (1 + \epsilon)\lambda)$, $\mathbb{E}[|D \cap D^*|] \geq \Omega(|D^*|/\log^4 |U|)$.

Property (a) follows by definition. Property (b) follows from the fact that $u \in D^*$ whenever $s \notin S(u, v)$, which happens with probability at least $1/2$. Property (c) follows because any vertex $v \in U \setminus S(u, v)$ satisfies $(u, v) \in P_{\text{ordered}}(G, U, W)$, of which there are at least $|U|/2$. For property (d), observe by construction of G' that for each vertex $v \in D^*$, the (s, v)-mincut has weight at most $W + |U|/2 \cdot 18\epsilon\lambda/|U|$. This is at most $(1+\epsilon)\lambda + 9\epsilon\lambda = (1+10\epsilon)\lambda$ since W is at most $(1+\epsilon)$ times the

Steiner mincut of G (which is at most λ). It follows that each $v \in D^*$ satisfies $\lambda_{G'}(s,v) \le (1+10\epsilon)\lambda$. Property (d) follows from Lemma 7.7 applied to input $(G, U, (1+10\epsilon)\lambda, s)$ and set D^*.

With properties (a) to (d) in hand, we now finish the proof of Lemma 7.9. For any vertex $u \in D$, all pairs $(u,v) \in P_{\text{ordered}}(G,U,W)$ (over all $v \in U$) disappear from $P_{\text{ordered}}(G,U,W)$, which is at least $|U|/2$ many by (c). In other words,

$$|P_{\text{ordered}}(G,U,W) \setminus P_{\text{ordered}}(G_{\text{large}}, U_{\text{large}}, W)| \ge \frac{|U| \cdot |D|}{2}.$$

Taking expectations and applying (d),

$$\mathbb{E}[|P_{\text{ordered}}(G,U,W) \setminus P_{\text{ordered}}(G_{\text{large}}, U_{\text{large}}, W)|] \ge \frac{|U| \cdot \mathbb{E}[|D|]}{2} \ge \Omega\left(\frac{|U| \cdot |D^*|}{\log^4 |U|}\right).$$

Moreover,

$$|U| \cdot |D^*| \ge \mathbb{E}\big[\big|\{(u,v) : u \in D^*\}\big|\big] \ge \frac{1}{2}|P_{\text{ordered}}(G,U,W)|,$$

where the second inequality follows by (b). Putting everything together, we obtain

$$\mathbb{E}[|P_{\text{ordered}}(G,U,W) \setminus P_{\text{ordered}}(G_{\text{large}}, U_{\text{large}}, W)|] \ge \Omega\left(\frac{|P_{\text{ordered}}(G,U,W)|}{\log^4 |U|}\right).$$

Finally, applying (a) gives

$$\mathbb{E}[|P(G,U,W) \setminus P(G_{\text{large}}, U_{\text{large}}, W)|] \ge \Omega\left(\frac{|P(G,U,W)|}{\log^4 |U|}\right).$$

Finally, we have $P(G_{\text{large}}, U_{\text{large}}, W) \subseteq P(G,U,W)$ since the (u,v)-mincut for $u, v \in U_{\text{large}}$ can only increase in G_{large} due to G_{large} being a contraction of G. Therefore,

$$|P(G,U,W)| - |P(G_{\text{large}}, U_{\text{large}}, W)| = |P(G,U,W) \setminus P(G_{\text{large}}, U_{\text{large}}, W)|,$$

and combining with the bound on $\mathbb{E}[|P(G,U,W) \setminus P(G_{\text{large}}, U_{\text{large}}, W)|]$ concludes the proof. □

7.4 Approximation

We first prove the two lemmas below before concluding the approximation guarantee.

Lemma 7.13. *For any distinct vertices $p, q \in U_{\text{large}}$, we have $\lambda_G(p,q) \le \lambda_{G_{\text{large}}}(p,q) \le (1+\gamma)\lambda_G(p,q)$.*

Proof. Since G_{large} is a contraction of G, we have $\lambda_G(p,q) \le \lambda_{G_{\text{large}}}(p,q)$. To show the other inequality, fix any (p,q)-mincut (A,B) in G. We iteratively "uncross" the cut (A,B) with each set $S_v \in \mathcal{F}$ $(v \in R)$ as follows: if $v \in A$, then replace (A,B) with $(A \cup S_v, B \setminus S_v)$, and if $v \in B$, then replace (A,B) with $(A \setminus S_v, B \cup S_v)$. By construction, the final cut is a (p,q)-cut that contains each S_v on one side of the cut, so it survives upon contraction into G_{large} and is a valid (p,q)-cut in G_{large}. We claim that the final cut has weight at most $(1+\gamma)\lambda_G(p,q)$, which would prove $\lambda_{G_{\text{large}}}(p,q) \le (1+\gamma)\lambda_G(p,q)$.

Let (A,B) be the current cut in the iterative process, and let S_v be the next cut we wish to uncross. Since S_v is a v-sided $(1+\gamma)$-fair cut on G', there is a feasible flow with no source/sink in $S_v \setminus \{v\}$ and which saturates $\partial_{G'} S_v$ up to factor $\frac{1}{1+\gamma}$ (in the direction from S_v to $V \setminus S_v$). By ignoring the flow outside $G'[S_v] \cup \partial_{G'} S_v$, we can view it as a flow from v to the boundary $\partial_{G'} S_v$ that saturates the boundary up to $\frac{1}{1+\gamma}$ factor. Decompose the flow into paths and ignore the paths ending at edges in $G' - G$ (which are all in $\partial_{G'} S_v$), obtaining a feasible flow from v to $\delta_G S_v$ that saturates $\partial_G S_v$ to factor $\frac{1}{1+\gamma}$.

Suppose first that $v \in B$. Further restrict the flow paths to only those ending at the edges in the subset $E_G(A \setminus S_v, A \cap S_v)$ of $\partial_G S_v$. Each of these paths must cross $E_G(A \cap S_v, B \cap S_v)$. There is at least $\frac{1}{1+\gamma} w(E_G(A \setminus S_v, A \cap S_v))$ flow along these paths, and they must cross a total capacity of $w(E_G(A \cap S_v, B \cap S_v))$. Since the flow is feasible, we conclude that $\frac{1}{1+\gamma} w(E_G(A \setminus S_v, A \cap S_v)) \leq w(E_G(A \cap S_v, B \cap S_v)$. In the operation that uncrosses S_v, the newly cut edges are precisely $E_G(A \setminus S_v, A \cap S_v)$, and all edges in $E_G(A \cap S_v, B \cap S_v)$ disappear. We *charge* the newly cut edges $E_G(A \setminus S_v, A \cap S_v)$ to the deleted edges $E_G(A \cap S_v, B \cap S_v)$ at a $1 + \gamma$ to 1 ratio. Finally, if $v \in A$, then the argument is symmetric by replacing A and B, and the charging is identical.

Since the sets $S_v : v \in R$ are disjoint, each edge is either charged to or charged from, but not both. If the total weight of charged-to edges is W, then the total weight of newly cut edges is at most $(1 + \gamma)W$, so the final cut has weight at most $\lambda_G(p, q) - W + (1 + \gamma)W \leq (1 + \gamma)\lambda_G(p, q)$, as promised. □

Lemma 7.14. *For any $v \in R$ and any distinct vertices $p, q \in U_v$, we have $\lambda_G(p, q) \leq \lambda_{G_v}(p, q) \leq (1 + 13\epsilon)\lambda_G(p, q)$.*

Proof. The lower bound $\lambda_G(p, q) \leq \lambda_{G_v}(p, q)$ holds because G_v is a contraction of G, so we focus on the upper bound. Fix any (p, q)-mincut in G, and let S be the side of the mincut not containing s (recall that $s \in U$ and $s \notin S_v$). Since $S_v \cup S$ is a (p, s)-cut (and also a (q, s)-cut), it is in particular a Steiner cut for terminals U, so $\delta_G(S_v \cup S) \geq (1 - \epsilon)\lambda$. Also, $\delta_G S_v \leq (1 + \gamma)(1 + 10\epsilon)\lambda \leq (1 + 11\epsilon)\lambda$ by Lemma 7.8. Together with the submodularity of cuts, we obtain

$$(1 + 11\epsilon)\lambda + \delta_G S \geq \delta_G S_v + \delta_G S \geq \delta_G(S_v \cup S) + \delta_G(S_v \cap S) \geq (1 - \epsilon)\lambda + \delta_G(S_v \cap S),$$

The set $S_v \cap S$ stays intact under the contraction from G to G_v, so $\delta_{G_v}(S_v \cap S) = \delta_G(S_v \cap S)$. Therefore,

$$\lambda_{G_v}(p, q) \leq \delta_{G_v}(S_v \cap S) = \delta_G(S_v \cap S) \leq \delta_G S + 12\epsilon\lambda = \lambda_G(p, q) + 12\epsilon\lambda.$$

Finally, $\lambda_G(p, q)$ is at least the Steiner mincut of G, which is at least $(1 - \epsilon)\lambda$, so the above is at most $\lambda_G(p, q) + 12\epsilon \cdot \lambda_G(p, q)/(1 - \epsilon) \leq (1 + 13\epsilon)\lambda_G(p, q)$, as promised. □

Combining the lemmas above, we can conclude the following.

Lemma 7.15. *Algorithm 5 outputs a $\big((1+13\epsilon)(1+\gamma)^{O(\epsilon^{-1}\log^6 n)}\big)^{\log_{1.5}|U|}$-approximate Gomory-Hu Steiner tree. With $\gamma = \epsilon^2/\log^6 n$, the approximation factor is $(1 + \epsilon)^{O(\log |U|)}$.*

Proof. To avoid clutter, define $\alpha = C\epsilon^{-1}\log^6 n$ for large enough constant $C > 0$. Consider the path down the recursion tree leading up to the current recursive instance, and let k be the number of consecutive recursive calls of type G_{large} directly preceding the current instance. We apply induction on $|U|$ and k to prove an $((1+13\epsilon)(1+\gamma)^\alpha)^{\log_{1.5}|U|}(1+\gamma)^{-k}$-approximation factor. By Lemma 7.8, we have $|U_v| \leq 2|U|/3$ for all $v \in R$, so by induction, the recursive outputs (T_v, f_v) are Gomory-Hu Steiner trees with approximation $((1 + 13\epsilon)(1 + \gamma)^\alpha)^{\log_{1.5}|U_v|} \leq ((1 + 13\epsilon)(1 + \gamma)^\alpha)^{\log_{1.5}|U|-1}$. By definition, this means that for all $s, t \in U_v$ and the minimum-weight edge (u, u') on the s–t path in T_v, letting $U'_v \subseteq U_v$ be the vertices of the connected component of $T_v - (u, u')$ containing s, we have that $f_v^{-1}(U'_v)$ is a $((1+13\epsilon)(1+\gamma)^\alpha)^{\log_{1.5}|U|-1}$-approximate (s,t)-mincut in G_v with value $w_T(u, u')$. Define $U' \subseteq U$ as the vertices of the connected component of $T - (u, u')$ containing s. By construction of (T, f) (lines 17 and 18), the set $f^{-1}(U')$ is simply $f_v^{-1}(U'_v)$ with the vertex x_v replaced by $V \setminus S_v$ in the case that $x_v \in f^{-1}(U')$. Since G_v is simply G with all vertices $V \setminus S_v$ contracted to x_v, we conclude that $\delta_{G_v}(f_v^{-1}(U'_v)) = \delta_G(f^{-1}(U'))$. By Lemma 7.14, the values $\lambda_G(s,t)$ and $\lambda_{G_v}(s,t)$ are within factor $(1+13\epsilon)$ of each other, so $\delta_G(f^{-1}(U'))$ approximates the (s,t)-mincut in G to a factor $(1+13\epsilon) \cdot ((1+13\epsilon)(1+\gamma)^\alpha)^{\log_{1.5}|U|-1}$, which we want to show is at most $((1+13\epsilon)(1+\gamma)^\alpha)^{\log_{1.5}|U|}(1+\gamma)^{-k}$. This follows by Lemma 7.11 since w.h.p., we always have $k \le C\epsilon^{-1}\log^6 n = \alpha$ for large enough constant $C > 0$. Thus, the Gomory-Hu Steiner tree condition for (T, f) is satisfied for all $s, t \in U_v$ for some $v \in R$.

We now focus on the case $s, t \in U_{\text{large}}$. By induction, the recursive output $(T_{\text{large}}, f_{\text{large}})$ is a Gomory-Hu Steiner tree with approximation $((1+13\epsilon)(1+\gamma)^\alpha)^{\log_{1.5}|U|}(1+\gamma)^{-(k+1)}$. Again, consider $s, t \in U_{\text{large}}$ and the minimum-weight edge (u, u') on the s–t path in T_{large}, and let $U'_{\text{large}} \subseteq U_{\text{large}}$ be the vertices of the connected component of $T_{\text{large}} - (u, u')$ containing s. Define $U' \subseteq U$ as the vertices of the connected component of $T - (u, u')$ containing s. By a similar argument, we have $\delta_{G_{\text{large}}}(f_{\text{large}}^{-1}(U'_{\text{large}})) = \delta_G(f^{-1}(U'))$. By Lemma 7.13, we also have $\lambda_{G_{\text{large}}}(s,t) = (1+\gamma)\lambda_G(s,t)$, so $\delta_G(f^{-1}(U'))$ is a $\big(((1+13\epsilon)(1+\gamma)^\alpha)^{\log_{1.5}|U|}(1+\gamma)^{-(k+1)} \cdot (1+\gamma)\big)$-approximate (s,t)-mincut in G, fulfilling the Gomory-Hu Steiner tree condition for (T, f) in the case $s, t \in U_{\text{large}}$.

There are two remaining cases: $s \in U_v$ and $t \in U_{v'}$ for distinct $v, v' \in R$, and $s \in U_v$ and $t \in U_{\text{large}}$; we treat both cases simultaneously. Since G has Steiner mincut at least λ, each of the contracted graphs G_{large} and G_v also has Steiner mincut at least λ. Since all edges on the approximate Gomory-Hu Steiner tree correspond to actual cuts in the graph, every edge in T_v and T_{large} has weight at least λ. By construction, the s–t path in T has at least one edge of the form $(f_v(x_v), f_{\text{large}}(y_v))$, added on line 17; this edge has weight $\delta_G S_v \le (1+\epsilon)(1+\gamma)\lambda$ by Lemma 7.8. Therefore, the minimum-weight edge on the s–t path in T has weight at least λ and at most $(1+\epsilon)(1+\gamma)\lambda$; in particular, it is a $(1+\epsilon)(1+\gamma)$-approximation of $\lambda_G(s,t)$, which fits the bound since $|U| \ge 2$. If the edge is of the form $(f_v(x_v), f_{\text{large}}(y_v))$, then by construction, the relevant set $f^{-1}(U')$ is exactly S_v, which is a $(1+\epsilon)$-approximate (s,t)-mincut in G. If the edge is in T_{large} or T_v or $T_{v'}$, then we can apply the same arguments used previously. □

Finally, we can reset $\epsilon \leftarrow \Theta(\epsilon/\log n)$ so that the $(1+\epsilon)^{O(\log|U|)}$-approximation becomes $(1+\epsilon)$. This concludes Theorem 7.2.

8 Expander Decomposition

In this section, we show how the fair cut algorithm implies a near-optimal expander decomposition algorithm, following the framework of Saranurak and Wang [SW19]. We first begin with some notation exclusive to this section. Define the *volume* of a set of vertices S as $\mathbf{vol}(S) = \sum_{v \in S} \deg(v)$, and let $G\{S\}$ denote the subgraph $G[S]$ with (weighted) self-loops added to vertices so that all vertex degrees are preserved, i.e., $\deg_G(v) = \deg_{G\{S\}}(v)$ for all $v \in S$. For a graph G, define its *conductance* as

$$\Phi_G = \min_{\emptyset \subsetneq S \subsetneq V} \frac{c(E(S, V \setminus S))}{\min\{\mathbf{vol}(S), \mathbf{vol}(V \setminus S)\}}.$$

We call G a *ϕ-expander* if $\Phi_G \geq \phi$.

Theorem 8.1 (Near-linear expander decomposition)**.** *Given a graph $G = (V, E)$ and a parameter ϕ, there is a randomized $\tilde{O}(m)$-time algorithm that with high probability finds a partitioning of V into $V_1, \ldots, V_k$ such that $\Phi_{G\{V_i\}} \geq \phi$ for all $i \in [k]$ and $\sum_i \delta(V_i) = \tilde{O}(\phi m)$.*

Note that if $G\{V_i\}$ is a ϕ-expander, then so is the induced subgraph $G[V_i]$ (which is sometimes more directly applicable). We also remark that [SW19] prove almost the exact same theorem, except their running time is $\tilde{O}(m/\phi)$, and is therefore slower for small values of ϕ.

At a high level, we use the same high-level recursive approach, except we replace the flow subroutines in their *trimming* and *cut-matching* steps of [SW19] with a fair cut computation. We note that there are known black-box reductions from expander decomposition to computing (approximately) most-balanced sparse cuts. But these reductions have some drawbacks and do not lead to near-optimal algorithms as in Theorem 8.1. The first reduction is implicit by Spielman and Teng [ST04]. However, they can only obtain a *weak* expander decomposition from most-balanced sparse cut algorithms. It is weak in the sense that each part is only guaranteed to be contained in some expanders, but may not induce an expander itself. Another reduction by Nanongkai and Saranurak [NS17] suffers from an inherent $n^{o(1)}$ factor loss in both quality and running time. By refining the non-blackbox approach of [SW19] via fair cuts, we successfully obtain the first expander decomposition algorithm that are optimal up to polylogarithmic factors.

8.1 Algorithm overview

We begin by describing the recursive algorithm of [SW19] at a high level. There are two main subroutines, *cut-matching* and *trimming*, to be described later. On input graph $G = (V, E)$ and parameter ϕ, the algorithm Decomp(G, ϕ) outputs a partition of V as follows.

1. Call Cut-Matching(G, ϕ), which either certifies that $\Phi_G \geq \phi$ or finds a cut (A, R)
2. If we certify $\Phi_G \geq \phi$, then return $\{V\}$ (the trivial partition)
3. Else if we find a relatively balanced cut (A, R), where $\mathbf{vol}(A)$ and $\mathbf{vol}(R)$ are both $\Omega(\mathbf{vol}(V)/\log^2 m)$:
   (a) Return Decomp$(G\{A\}, \phi) \cup$ Decomp$(G\{R\}, \phi)$
4. Else, suppose that $\mathbf{vol}(R) \leq O(\mathbf{vol}(V)/\log^2 m)$:
   (a) $A' =$ Trimming(G, A, ϕ)
   (b) Return $\{A'\} \cup$ Decomp$(G\{A'\}, \phi)$

If Cut-Matching and Trimming run in T time, then the entire recursive algorithm takes $\tilde{O}(T)$ time. In [SW19], the two subroutines are solved in $\tilde{O}(m/\phi)$ time. In this section, we improve both running times to $\tilde{O}(m)$ by substituting their flow subroutines with fair cuts/flows.

8.2 Trimming step

To describe the trimming step formally, we need the concept of a *nearly expander*.

Definition 8.2 (nearly ϕ-expander)**.** Given $G = (V, E)$ and a set of vertices $A \subseteq V$, we say that A is a nearly ϕ-expander in G if for all subsets $S \subseteq A$ with $\mathbf{vol}(S) \leq \mathbf{vol}(A)/2$, we have $c(E(S, V \setminus S)) \geq \phi\mathbf{vol}(S)$.

In the trimming step, we are given a set $A \subseteq V$ such that A is a nearly ϕ-expander in G, and the goal is to "trim" A to a subset $A' \subseteq A$ such that $G\{A'\}$ is a $\phi/6$-expander. The formal subroutine is described in the theorem below, copied almost identically to Theorem 2.1 of [SW19] except for the improved $\tilde{O}(m)$ running time.

Theorem 8.3 (Trimming, Theorem 2.1 of [SW19])**.** *Given graph $G = (V, E)$ and $A \subseteq V$ such that*

1. *A is a nearly ϕ-expander in G, and*
2. *$c(E(A, \overline{A})) \le \phi \mathbf{vol}(A)/10$,*

the trimming step finds $A' \subseteq A$ in time $\tilde{O}(m)$ such that $\Phi_{G\{A'\}} \ge \phi/6$. Moreover, $\mathbf{vol}(A') \ge \mathbf{vol}(A) - 4c(E(A, \overline{A}))/\phi$ and $c(E(A', \overline{A'})) \le 2c(E(A, \overline{A}))$.

Proof. Consider the following (s,t)-flow problem on a new graph $H = (V_H, E_H)$. Start from $G\{A\}$, and contract $V \setminus A$ into a single vertex and label it the source s. Next, multiply the capacity of each edge by $3/\phi$. Finally, add a new sink vertex t and connect it to each vertex $v \in A$ with an edge of capacity $\deg_{G\{A\}}(v)$. Let $\alpha = 0.1$, and compute a $(1+\alpha)$-fair cut (S, T). Let $A' = T \setminus \{t\}$, which we now show satisfies the properties of the lemma.

First, suppose for contradiction that $G\{A'\}$ is not a $\phi/6$-expander. Then, there is a violating set $U \subseteq A'$ satisfying

$$c(E(U, A' \setminus U)) \le \frac{\phi}{6}\mathbf{vol}(U).$$

Since A is a nearly ϕ-expander,

$$c(E(U, V \setminus U)) \ge \phi \mathbf{vol}(U).$$

Taking the difference of the two inequalities above,

$$c(E(U, V \setminus A')) = c(E(U, V \setminus U)) - c(E(U, A' \setminus U)) \ge \frac{5\phi}{6}\mathbf{vol}(U).$$

Since (S, T) is a $(1+\alpha)$-fair cut, there is a feasible flow f that saturates each edge of $E_H(S, T)$ to factor $\frac{1}{1+\alpha}$. Each edge (u, v) in $E(U, V \setminus A')$ corresponds to an edge in $E_H(S, T)$ of capacity $\frac{3}{\phi}c_{G\{A\}}(u, v)$, and the flow f must send at least $\frac{1}{1+\alpha} \cdot \frac{3}{\phi}c_{G\{A\}}(u, v) \ge \frac{2}{\phi}c_{G\{A\}}(u, v)$ flow along that edge (in the direction from S to T). In total, the amount of flow entering U in H is at least

$$\frac{2}{\phi}c_{G\{A\}}(E(U, V \setminus A')) \ge \frac{2}{\phi} \cdot \frac{5\phi}{6}\mathbf{vol}(U) = \frac{5}{3}\mathbf{vol}(U).$$

On the other hand, at most $\mathbf{vol}(U)$ flow can leave U along the edges incident to t, and at most

$$\frac{3}{\phi}c_{G\{A\}}(E(U, A' \setminus U)) \le \frac{3}{\phi} \cdot \frac{\phi}{6}\mathbf{vol}(U) = \frac{1}{2}\mathbf{vol}(U)$$

flow can cross from U to $A' \setminus U$. This totals at most $\frac{3}{2}\mathbf{vol}(U)$ flow that can exit U, which is strictly less than the $\ge \frac{5}{3}\mathbf{vol}(U)$ flow that enters U, a contradiction. Thus, $G\{A'\}$ is a $\phi/6$-expander.

Finally, we show the properties $\mathbf{vol}(A') \ge \mathbf{vol}(A) - 4c(E(A, \overline{A}))/\phi$ and $c(E(A', \overline{A'})) \le 2c(E(A, \overline{A}))$ promised by the lemma. Since (S, T) is a $(1+\alpha)$-fair cut, it is in particular a $(1+\alpha)$-approximate (s,t)-mincut. Since $(\{s\}, V_H \setminus \{s\})$ is an (s,t)-cut of capacity $\frac{3}{\phi}c(E(A, \overline{A}))$, it follows that the cut

(S,T) has capacity at most $(1+\alpha)\cdot\frac{3}{\phi}c(E(A,\overline{A}))$. To prove the first property above, note that each vertex $v\in A\setminus A'$ is on the S-side of the cut (S,T), and therefore contributes $\deg_{G\{A\}}(v)$ to the cut (S,T) from the edge (v,t). Summing over all $v\in A\setminus A'$, we obtain

$$\mathbf{vol}(A\setminus A')\le c_H(E(S,T))\le(1+\alpha)\cdot\frac{3}{\phi}c(E(A,\overline{A}))\le\frac{4}{\phi}c(E(A,\overline{A})),$$

which proves the first property. For the second property above, note that each edge (u,v) in $E(A',\overline{A'})$ corresponds to an edge in $E(S,T)$ with $3/\phi$ times the capacity, so summing over all such edges,

$$\frac{3}{\phi}c(E(A',\overline{A'}))\le c_H(E(S,T))\le(1+\alpha)\cdot\frac{3}{\phi}c(E(A,\overline{A})),$$

which proves the second property. □

8.3 Cut-matching step

In the cut-matching step, the goal is to either certify that the input graph is an expander, or find a low-conductance cut with a special property: either it is balanced, or if not, we guarantee that the larger side is a nearly expander. The name "cut-matching" comes from the *cut-matching game* framework [KRV09] that this step uses, though its description is not required in this section.

The formal subroutine is described in the theorem below, copied almost identically to Theorem 2.2 of [SW19] except for the improved $\tilde{O}(m)$ running time.

Theorem 8.4 (Cut-Matching, Theorem 2.2 of [SW19])**.** *Given a graph $G=(V,E)$ and a parameter ϕ, the* cut-matching *step takes $\tilde{O}(m)$ time and must end with one of the three cases:*

1. *We certify G has conductance $\Phi_G\ge\phi$.*
2. *We find a cut $(A,\overline{A})$ in G of conductance $\Phi_G(\overline{A})=O(\phi^2 m)$, and $\mathbf{vol}(A),\mathbf{vol}(\overline{A})$ are both $\Omega(m/\log^2 m)$, i.e., we find a relatively balanced low conductance cut.*
3. *We find a cut $(A,\overline{A})$ with $\Phi_G(\overline{A})\le c_0\phi\log^2 m$ for some constant c_0, and $\mathbf{vol}(\overline{A})\le m/(10c_0\log^2 m)$, and A is a nearly ϕ-expander.*

We will not present the entire proof of this theorem, since most of the steps remain unchanged from [SW19]. The only step that takes $\tilde{O}(m/\phi)$ time in [SW19] is their subroutine Lemma B.6, so it suffices to describe it and improve its running time to $\tilde{O}(m)$.

First, we introduce some notation from [SW19]. Given a graph $G=(V,E)$ and a subset of vertices $A\subseteq V$, denote by $G\{S\}$ the induced subgraph $G[S]$ but with self-loops added to vertices so that any vertex in S has the same degree as its degree in G. Given a multi-graph $G=(V,E)$, its *subdivision graph* $G_E=(V',E')$ is the graph where we put a *split node* x_e on each edge $e\in E$ (including the self-loops). Formally, $V'=V\cup X_E$ where $X_E=\{x_e\mid e\in E\}$, and $E'=\{(u,x_e),(v,x_e)\mid e=(u,v)\in E\}$. While [SW19] only defines the subdivision graph for unweighted graphs, we can extend the definition to weighted graphs by assigning the edges $(u,x_e),(v,x_e)$ ro have capacity $c(e)$ for each edge $e=(u,v)\in E$. For a split node $x_{(u,v)}$, we abuse notation and define its *capacity* $c(x_{(u,v)})$ to be the capacity $c(u,v)$ of the edge (u,v) in G. For a set of split nodes S, its total capacity $c(S)$ is the sum of the capacities of the split nodes in S.

The input to the subroutine of Lemma B.6 is

1. A set of vertices $A \subseteq V'$,
2. A set of source split nodes $A^l \subseteq A \cap X_E$ of total capacity at most $c_{G\{A\}}(A \cap X_E)/8$, and
3. A set of target split nodes $A^r \subseteq A \cap X_E$ of total capacity at least $c_{G\{A\}}(A \cap X_E)/2$.

For any graph H and positive number U, let H^U be the graph where each edge has its capacity multiplied by U. Let $U = 1/(\phi \log^2 m)$, and consider a flow problem on $(G_E\{A\})^U$ where each split node $x_{(u,v)} \in A^l$ is a source of $c(u,v)$ units of mass (where $c(u,v)$ is the original capacity in G_E, not multiplied by U) and each split node $x_{(u,v)} \in A^r$ is a sink with capacity $c(u,v)$. The task is to either find

1. A feasible flow f for the above problem, or
2. A cut S where $\Phi_{G\{A\}}(S) = O(\phi \log^2 m)$ and a feasible flow for the above flow problem when only split nodes $x_{(u,v)}$ in $A^l \setminus S$ are sources of $c(u,v)$ units.

Lemma B.6 of [SW19] uses a push-relabel or blocking-flow algorithm that runs in $O(m/(\phi \log m))$ time. Using fair cuts, we improve the running time to $\tilde{O}(m)$, independent of ϕ, in the lemma below.

Lemma 8.5. *We can solve the task above in $\tilde{O}(m)$ time.*

Proof. Let $\alpha = 0.1$, and consider the flow problem on the graph $H = (G_E\{A\})^{U/(1+\alpha)}$ instead. First, convert it to an (s,t)-flow problem by adding a source vertex s, connected to each $x_{(u,v)} \in A^l$ with capacity $c_{G\{A\}}(u,v)$, and a sink vertex t, connected to each $x_{(u,v)} \in A^r$ with capacity $c_{G\{A\}}(u,v)/(1+\alpha)$. Next, we compute a $(1+\alpha)$-fair cut (S,T) and corresponding feasible flow f' in $\tilde{O}(m)$ time. There are two cases below:

1. $S = \{s\}$. In this case, by definition of fair cuts, the flow f' sends at least $c_{G\{A\}}(u,v)/(1+\alpha)$ flow out of each edge from s. By computing a path decomposition and removing paths accordingly, we can modify f' to a new feasible flow f'' that sends *exactly* $c_{G\{A\}}(u,v)/(1+\alpha)$ flow along each edge out of s, and at most $c_{G\{A\}}(u,v)/(1+\alpha)$ flow along each edge into t. Finally, we let flow f be f'' multiplied by $(1+\alpha)$, and then restricted to graph $(G_E\{A\})^U$. Since f'' is feasible on the edges in $(G_E\{A\})^{U/(1+\alpha)}$, we conclude that f is feasible on $(G_E\{A\})^U$.
2. $S \neq \{s\}$. In this case, let $E_s \subseteq E_H(S,T)$ be the edges of the cut incident to s, let $E_t \subseteq E_H(S,T)$ be those incident to t, and let $E_m = E_H(S,T) \setminus (E_s \cup E_t)$ be the remaining cut edges. Recall that edges in E_s and E_t retain their original capacity from $G_E\{A\}$, while edges in E_m have their capacity scaled by $U/(1+\alpha)$. Also, note that E_m is, up to this scaling factor, exactly the cut $E(S \setminus \{s\}, T \setminus \{t\})$ in the original graph $G\{A\}$.[12] In other words,

$$c_{G\{A\}}(E(S \setminus \{s\}, T \setminus \{t\})) = \frac{1+\alpha}{U} \cdot c_H(E_m). \tag{7}$$

   Let $\overline{E}_s$ be the edges incident to s that are not in E_s. Since (S,T) is a $(1+\alpha)$-fair cut, there is a flow f from s to t that saturates each edge in $E_H(S,T)$ to fraction at least $\frac{1}{1+\alpha}$. In particular, this means that the sub-flow from s starting from edges $\overline{E}_s$ must saturate edges in

[12] We show later that the degenerate case $T = \{t\}$ cannot happen.

$E_H(S,T) \setminus E_s$ to fraction at least $\frac{1}{1+\alpha}$. This implies that $c_H(E_H(S,T) \setminus E_s) \le (1+\alpha)c_H(\overline{E}_s)$. Moreover, for each edge $(s, x_e) \in \overline{E}_s$, the split node x_e is on the $S \setminus \{s\}$ side of the cut $E(S \setminus \{s\}, T \setminus \{t\})$ in $G\{A\}$, so

$$\mathbf{vol}_{G\{A\}}(S \setminus \{s\}) \ge \sum_{(s,x_e)\in\overline{E}_s} \deg_{G\{A\}}(x_e) = 2c_H(\overline{E}_s) \ge \frac{2}{1+\alpha} c_H(E_H(S,T) \setminus E_s) \ge \frac{2}{1+\alpha} c_H(E_m). \tag{8}$$

Putting (7) and (8) together, we obtain

$$\mathbf{vol}_{G\{A\}}(S \setminus \{s\}) \ge \frac{2U}{(1+\alpha)^2} c_{G\{A\}}(E(S \setminus \{s\}, T \setminus \{t\})), \tag{9}$$

so we would be done as long as we show that $\mathbf{vol}_{G\{A\}}(S \setminus \{s\}) \le O(\mathbf{vol}_{G\{A\}}(T \setminus \{t\}))$.

Consider now the edges E_t. Their capacities are scaled down by $1/(1+\alpha)$, so their total original capacity is at most $(1+\alpha)^2 c_{G\{A\}}(A^l)$, which is at most $(1+\alpha)^2 c_{G\{A\}}(A \cap X_E)/8$ by property (2). On the other hand, the total capacity of edges incident to t is $c_{G\{A\}}(A^r)/(1+\alpha)$, which is at least $c_{G\{A\}}(A \cap X_E)/(2(1+\alpha))$ by property (3). It follows that at least

$$c_{G\{A\}}(A \cap X_E)/(2(1+\alpha)) - (1+\alpha)^2 c_{G\{A\}}(A \cap X_E)/8 \ge \Omega(c_{G\{A\}}(A \cap X_E))$$

total capacity of edges incident to t are not in E_t. In other words, their corresponding split nodes are on the $T \setminus \{t\}$ side of the cut $E(S \setminus \{s\}, T \setminus \{t\})$, which means that $\mathbf{vol}_{G\{A\}}(T \setminus \{t\}) \ge \Omega(c_{G\{A\}}(A \cap X_E))$. Now observe that $c_{G\{A\}}(A \cap X_E)$ is a constant fraction of the total volume of the graph $G\{A\}$, so $\mathbf{vol}_{G\{A\}}(T \setminus \{t\}) \ge \Omega(\mathbf{vol}_{G\{A\}}(A))$. Together with (9), we obtain the desired

$$\Phi_{G\{A\}}(S \setminus \{s\}) = \frac{c_{G\{A\}}(S \setminus \{s\}, T \setminus \{t\})}{\min\{\mathbf{vol}_{G\{A\}}(S \setminus \{s\}), \mathbf{vol}_{G\{A\}}(T \setminus \{t\})\}} \le O(1/U) = O(\phi \log^2 m).$$

□

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. Danupon Nanongkai was also supported by the Swedish Research Council (Reg. No. 2019-05622). Debmalya Panigrahi was supported in part by NSF grants CCF-1750140 (CAREER Award) and CCF-1955703.

References

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. 11

[AKL+21] Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Gomory-hu tree in subcubic time. *CoRR*, abs/2111.04958, 2021. 1, 3, 4

[AKT20a] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Cut-equivalent trees are optimal for min-cut queries. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 105–118. IEEE, 2020. 3

[AKT20b] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 48–61. SIAM, 2020. 3

[AKT21a] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. APMF < apsp? gomory-hu tree for unweighted graphs in almost-quadratic time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1135–1146. IEEE, 2021. 3

[AKT21b] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Friendly cut sparsifiers and faster gomory-hu trees. *CoRR*, abs/2110.15891, 2021. 3

[AKT21c] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. Subcubic algorithms for gomory-hu tree in unweighted graphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1725–1737. ACM, 2021. 3

[BBG+20] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. *arXiv preprint arXiv:2004.08432*, 2020. 4

[BHKP07] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In *39th Annual ACM Symposium on Theory of Computing*, STOC'07, pages 605–614, 2007. 3

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol 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)*, pages 1158–1167. IEEE, 2020. 3

[CH03] Richard Cole and Ramesh Hariharan. A fast algorithm for computing steiner edge connectivity. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 167–176. ACM, 2003. 3

[CKL+22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. March 2022. 1, 4

[CLP22] Ruoxu Cen, Jason Li, and Debmalya Panigrahi. Augmenting edge connectivity via isolating cuts. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2022. 1

[CQ21a] Chandra Chekuri and Kent Quanrud. Isolating cuts, (bi-)submodularity, and faster algorithms for connectivity. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 50:1–50:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 1

[CQ21b] Chandra Chekuri and Kent Quanrud. Isolating cuts, (bi-)submodularity, and faster algorithms for connectivity. In *ICALP*, volume 198 of *LIPIcs*, pages 50:1–50:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 4

[CRJ17] Jaime Cohen, Luiz A. Rodrigues, and Elias P. Duarte Jr. Parallel cut tree algorithms. *J. Parallel Distributed Comput.*, 109:1–14, 2017. 3

[CS19] Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 66–73, 2019. 1, 4, 44

[DV94] Yefim Dinitz and Alek Vainshtein. The connectivity carcass of a vertex subset in a graph and its incremental maintenance. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 716–725. ACM, 1994. 3

[GG18] Barbara Geissmann and Lukas Gianinazzi. Parallel minimum cuts in near-linear work and low depth. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 1–11, 2018. 45

[GKK+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 81–90, 2015. 1

[GLP21] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. *FOCS*, 2021. 1

[GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998. 4

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021. 44

[HKP07] Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Efficient algorithms for computing all low s-t edge connectivities and related problems. In *Proceedings of* *the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 127–136, 2007. 3

[Kar00] David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000. 45

[KLOS14] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014. 1, 4

[KRV09] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):1–15, 2009. 37

[LF80] Richard E Ladner and Michael J Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980. 45

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *STOC*, pages 317–329. ACM, 2021. 1, 4

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE Computer Society, 2020. 1, 3, 6, 23

[LP21] Jason Li and Debmalya Panigrahi. Approximate Gomory-Hu tree is faster than $n-1$ max-flows. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing*, 2021. 3, 6, 24, 25, 27

[LPS21] Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1124–1134. IEEE, 2021. 3

[LS20] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814. ACM, 2020. 1

[Mad11] Aleksander Madry. *From graphs to matrices, and back: new techniques for graph algorithms*. PhD thesis, Massachusetts Institute of Technology, 2011. 1

[MCJ20] Charles Maske, Jaime Cohen, and Elias P. Duarte Jr. Speeding up the gomory-hu parallel cut tree algorithm with efficient graph contractions. *Algorithmica*, 82(6):1601–1615, 2020. 3

[MN20] Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *STOC*, pages 496–509. ACM, 2020. 4

[MN21] Sagnik Mukhopadhyay and Danupon Nanongkai. A note on isolating cut lemma for submodular function minimization. *CoRR*, abs/2103.15724, 2021. 1

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o($n^{1/2 - \epsilon}$)-time. In *STOC*, pages 1122–1129. ACM, 2017. 4, 35

[NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961. IEEE Computer Society, 2017. 4

[Pen16] Richard Peng. Approximate undirected maximum flows in o (m polylog (n)) time. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1862–1867. SIAM, 2016. 1, 3, 7, 8

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 227–238. SIAM, 2014. 3, 8

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269. IEEE, 2013. 1, 4, 5

[She17] Jonah Sherman. Area-convexity, l_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017. 3

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90, 2004. 4, 35

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. 2019. To appear in SODA'19. 4, 7, 34, 35, 36, 37, 38

[vdBLL+21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. 2021. arXiv:2101.05719. 1

[vdBLN+20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *FOCS*, pages 919–930. IEEE, 2020. 1

[Wul17] Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *STOC*, pages 1130–1143. ACM, 2017. 4

[Zha21a] Tianyi Zhang. Faster cut-equivalent trees in simple graphs. *CoRR*, abs/2106.03305, 2021. 3

[Zha21b] Tianyi Zhang. Gomory-hu trees in quadratic time. *CoRR*, abs/2112.01042, 2021. 3

A Parallel Algorithms

The goal of this section to prove Theorem 1.5. Along the way, we will show that all algorithmic components we use and develop can be parallelized.

A.1 Congestion Approximators

The first thing we need is a parallel construction of congestion approximators (see Theorem 3.2).

Theorem A.1 (Parallel Congestion approximator)**.** *There is a randomized algorithm that, given an unweighted graph $G = (V, E)$ with n vertices and m edges, constructs in $m^{1+o(1)}$ work and $m^{o(1)}$ depth with high probability same laminar as in Theorem 3.2 except that $\gamma_{\mathcal{S}} = n^{o(1)}$.*

We only state the result for unweighted graphs as it follows quite easily from [CS19, GRST21]. We believe that known techniques also imply the same for weighted graphs. Below, we sketch the proof of Theorem A.1.

First, we need a definition of *boundary-linked expander decomposition* introduced in [GRST21]. For any graph $G = (V, E)$ and any set $S \subset V$, let $G[S]$ denote the subgraph of G induced by S. For any $w \geq 0$, let $G[S]^w$ be obtained from $G[S]$ by adding w self-loops to each vertex $v \in S$ for every boundary edge (v, x), $x \notin S$.

Definition A.2. For any graph $G = (V, E)$ with m edges, a (ϵ, ϕ, α)-*boundary-linked expander decomposition* is partition $\mathcal{U} = (U_1, \ldots, U_k)$ of vertex set V such that $\sum_i |E(U_i, V \setminus U_i)| \leq \epsilon m$ and $G[U_i]^{\alpha/\phi}$ is a ϕ-expander for all i.

Note that $(\epsilon, \phi, 0)$-boundary-linked expander decomposition is the standard (ϵ, ϕ)-expander decomposition. A parallel algorithm for computing an expander decomposition of an unweighted graph was explicitly shown in [CS19]. In fact, the algorithm works even in the distributed model called CONGEST.

Theorem A.3 ([CS19])**.** *For any positive integer k, $\epsilon \in (0, 1)$, and $\phi \geq (\epsilon/\log n)^{2^{O(k)}}$, there is an algorithm for computing an (ϵ, ϕ)-expander decomposition of an unweighted graph in CONGEST in $O(n^{2/k}\text{poly}(1/\phi, \log n))$ rounds w.h.p. In fact, this algorithm has $n^{1/O(\log\log\log n)}$-depth and $m^{1+o(1)}$ work.*

We will choose $k = \log\log\log n$ from now on. This algorithm can be easily extended to compute a $(\epsilon, \phi, \epsilon)$-boundary-linked expander decomposition. The idea is as follows: whenever we find a ϕ-sparse cut, for each cut edge (u, v), we add (α/ϕ) self-loops on both u and v before recursing on both sides. The largest boundary-linked parameter α we can get can be derived by setting $\epsilon = 1/O(\log n)$ and see the largest value of ϕ we can get. In this case, it is $1/2^{\Theta(\log\log n)^2)}$ when $\epsilon = 1/O(\log n)$ and $k = \log\log\log n$. From this, it implies the following:

Theorem A.4. *When $\epsilon = 1/2^{\Theta(\sqrt{\log n})}$, $\phi \geq (\epsilon/\log n)^{2^{O(\log\log\log)}} \geq 1/2^{\Theta(\sqrt{\log n}\cdot\log\log n)}$, and $\alpha \geq 1/2^{\Theta(\log\log n)^2)}$, there is an algorithm that w.h.p. computes a (ϵ, ϕ, α)-boundary-linked expander decomposition in $n^{1/O(\log\log\log n)}$-depth and $m^{1+o(1)}$ work. (In fact, the algorithm is implementable in CONGEST in $n^{1/O(\log\log\log n)}$ rounds.)*

In [GRST21], it is shown that constructing congestion approximators can be reduced to computing boundary-linked expander decomposition a few times, which is summarized as follows:

Lemma A.5. *By calling an algorithm for computing a (ϵ, ϕ, α)-boundary-linked expander decomposition for $O(\log_{(1/\epsilon)} m)$ times, one can construct a congestion approximator $\mathcal{S}$ with quality $\gamma_{\mathcal{S}} = O((1/\phi) \cdot (1/\alpha)^{\log_{(1/\epsilon)} m})$.*

Plugging Theorem A.4 into the above lemma, this implies an algorithm for Theorem A.1 where $n^{1/O(\log\log\log n)}$ depth and $m^{1+o(1)}$ work that computes a congestion approximator $\mathcal{S}$ with quality $\gamma_{\mathcal{S}} = 2^{\Theta(\sqrt{\log n}\cdot(\log\log n)^2)} = n^{o(1)}$.

A.2 Fair Cuts

Given the above parallel construction for congestion approximator, we can obtain the following parallel fair cut algorithm:

Theorem A.6 (Parallel Fair Cut)**.** *Given an unweighted graph $G = (V, E)$, two vertices $s, t \in V$, and $\epsilon \in (0, 1]$, we can compute with high probability a $(1 + \epsilon)$-fair (s, t)-cut in $n^{o(1)}/\text{poly}(\epsilon)$ depth and $m^{1+o(1)}/\text{poly}(\epsilon)$ work.*

Before proving the above theorem, we first argue how to obtain a parallel version of the AlmostFair algorithm.

See the running time analysis of AlmostFair in Section 4.3. We can parallelize it as follows. We initialize by computing a congestion approximator $\mathcal{S}$ with quality $\gamma_{\mathcal{S}} = n^{o(1)}$ via Theorem A.1. The other initialization steps consist of elementary operations which can be parallelized in $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work.

For each round of the multiplicative weight update algorithm, the only non-trivial step is to a compute the "deletion set" D^i via a sweep cut (Lemma 4.9).

We will prove the below claim at the end.

Claim A.7. *Lemma 4.9 admits a parallel implementation with $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work.*

Since our multiplicative weight update algorithm consists of $T = O(\log(n)/\alpha^2) = m^{o(1)}/\text{poly}(\epsilon)$ rounds (recall that $\alpha = \epsilon/\gamma_{\mathcal{S}}$), we can implement the AlmostFair algorithm from Theorem 4.1 in $m^{o(1)}/\text{poly}(\epsilon)$ depth and $m^{1+o(1)}/\text{poly}(\epsilon)$ work.

Given the parallel implementation of the AlmostFair algorithm, we are almost done. The algorithm for computing fair cuts in Section 5.1 simply calls the AlmostFair subroutine for $O(\log(C/\beta)/\beta)$ times where we set $\beta = \Theta(\alpha/\log n)$. Therefore, the algorithm require $m^{o(1)}/\text{poly}(\epsilon)$ depth and $m^{1+o(1)}/\text{poly}(\epsilon)$ work. This concludes Theorem A.6.

Proof of Claim A.7. Recall that the problem is to compute x^* which is the largest x such that $\Delta|_{V^{i-1}}(V_{>x}) - \delta_H(V_{>x}) > 0$ where $V_{>x} = \{v \in V(H) : \phi^i_v > x\}$.

We start by parallel sorting vertices v according to their potential ϕ^i_v in decreasing order. Let $v_1, \dots, v_n$ be the vertices after sorting. Let $S_k = \{v_1, \dots, v_k\}$. We can compute the list of values of $\Delta|_{V^{i-1}}(S_k)$ for all $k \in [n]$ in $O(\log n)$ depth and $O(n)$ work using a classic parallel prefix sum algorithm [LF80].

Observe that our goal is equivalent to finding the largest k where $\delta_H(S_k) - \Delta|_{V^{i-1}}(S_k) < 0$. By binary search, we can reduce the problem to checking if there is k where $\delta_H(S_k) - \Delta|_{V^{i-1}}(S_k) < 0$.

Now, this problem can be solved using a parallel 1-respecting mincut algorithm by Karger [Kar00] (see also Lemma 11 of [GG18]) with $O(\log n)$ depth and $O(m)$ work. The reduction is as follows. Let H' be the graph obtained from H by inserting the tree $P = (v_1, \dots, v_n)$, which is a path. Let M be a big number such that $M - \Delta|_{V^{i-1}}(S_k) > 0$. Each tree edge $(v_k, v_{k+1}) \in P$, we set its weight to be $M - \Delta|_{V^{i-1}}(S_k)$. By computing a mincut in H' that 1-respect the tree P, we will obtain k such that $\delta_{H'}(S_k)$ is minimized. Since $\delta_{H'}(S_k) = \delta_H(S_k) + M - \Delta|_{V^{i-1}}(S_k)$, we can just check if $\delta_{H'}(S_k) - M < 0$. □

A.3 Isolating Cuts and Gomory-Hu Tree

Here, we finally prove Theorem 1.5. We first briefly explain how the approximate isolating cuts algorithm (Algorithm 1) and Gomory-Hu tree algorithm (Algorithm 5) can be parallelized to run in $\tilde{O}(m)$ work and $\text{polylog}(n)$ parallel time.

For approximate isolating cuts, Phase 1 of Algorithm 1 requires $O(\log n)$ many calls to $(1+\gamma)$-fair cut, which has a parallel algorithm by Theorem A.6. For Phase 2, the sets S_t and graphs G_t can be constructed independently for different t in parallel, and for the $(1+\beta)$-approximate minimum cut computation, we can use the parallel $(1+\beta)$-fair cut algorithm of Theorem A.6, which is also a $(1+\beta)$-approximate minimum cut.

For Gomory-Hu tree, there are a few additional algorithms that need to be investigated. For the "Cut Threshold Step" algorithm (Algorithm 3), the $O(\log n)$ independent iterations can be executed in parallel, so the entire algorithm can as well. The $(1+\gamma)$-approximate Gomory-Hu Steiner tree "step" (Algorithm 4) makes $O(\log^3 n)$ (sequential) calls to Algorithm 3, so it can also be parallelized. The Gomory-Hu tree algorithm itself (Algorithm 5) makes one call to Algorithm 4 and, aside from the recursive call on line 11, consists of elementary operations that can directly be parallelized. For the recursive calls, we use Lemma 7.11 to argue that the recursion tree has depth $\text{polylog}(n)$ w.h.p., so the recursive calls can be parallelized as well. (We stop the recursion after a large enough $\text{polylog}(n)$ many recursive calls, which is all we need w.h.p.)

B Proof of Uncrossing Property

Here, we prove the uncrossing property (Lemma 1.2), restated below. We remark that the proof follows the same outline as the proof of Lemma 6.3 for approximate isolating cuts.

Lemma 1.2 (Approximate Uncrossing Property)**.** *For any vertices s and t, let (S,T) be an α-fair (s,t)-mincut. Then, for any $u, v \in S$, there exists $R \subset S$ such that $(R, V \setminus R)$ is an α-approximate (u,v)-mincut.*

Proof. Let $(U, V\setminus U)$ be a (u,v)-mincut. Without loss of generality, assume that $t \notin U$. (Otherwise, we can swap u and v and use $V\setminus U$ in place of U.) Our goal is to show that $U\cap S$ is an α-approximate (u,v)-mincut contained in S, so that setting $R = U\cap S$ proves the lemma. Equivalently, we want to show that $\delta(U\cap S) \le \alpha \cdot \delta(U)$.

Using the notation $\uplus$ for disjoint union, we can write

$$E(U, V\setminus U) = E(U\cap S, V\setminus (U\cup S)) \uplus E(U\cap S, S\setminus U) \uplus E(U\setminus S, V\setminus U)$$
$$E(U\cap S, V\setminus (U\cap S)) = E(U\cap S, V\setminus (U\cup S)) \uplus E(U\cap S, S\setminus U) \uplus E(U\cap S, U\setminus S).$$

Since the first two sets are identical, we only need to compare the third sets $E(U\setminus S, V\setminus U)$ and $E(U\cap S, U\setminus S)$. Since (S,T) is an α-fair (s,t)-cut, there is a feasible flow from s to t that, for each edge in $E(S,T)$, sends at least $1/\alpha$ times capacity in the direction from S to T. Now, consider the flow on the subset of edges $E(U \cap S, U \setminus S) \subseteq E(S, T)$. This flow must reach t eventually, and it must exit $U \setminus S$ along the edges in $E(U \setminus S, V \setminus (U \cup S))$. Thus,

$$\delta(U \cap S, U \setminus S) \leq \alpha \cdot \delta(U \setminus S, V \setminus (U \cup S)) \leq \alpha \cdot \delta(U \setminus S, V \setminus U).$$

It follows that $\delta(U \cap S) \leq \alpha \cdot \delta(U)$, which proves the lemma. $\square$