Deterministic Near-Linear Time Minimum Cut in Weighted Graphs

Monika Henzinger* Jason Li† Satish Rao‡ Di Wang§

Abstract

In 1996, Karger [Kar96] gave a startling randomized algorithm that finds a minimum-cut in a (weighted) graph in time $O(m \log^3 n)$ which he termed near-linear time meaning linear (in the size of the input) times a polylogarthmic factor. In this paper, we give the first deterministic algorithm which runs in near-linear time for weighted graphs.

Previously, the breakthrough results of Kawarabayashi and Thorup [KT19] gave a near-linear time algorithm for simple graphs (which was improved to have running time $O(m \log^2 n \log\log n)$ in [HRW20].) The main technique here is a clustering procedure that perfectly preserves minimum cuts. Recently, Li [Li21] gave an $m^{1+o(1)}$ deterministic minimum-cut algorithm for weighted graphs; this form of running time has been termed "almost-linear". Li uses almost-linear time deterministic expander decompositions which do not perfectly preserve minimum cuts, but he can use these clusterings to, in a sense, "derandomize" the methods of Karger.

In terms of techniques, we provide a structural theorem that says there exists a sparse clustering that preserves minimum cuts in a weighted graph with $o(1)$ error. In addition, we construct it deterministically in near linear time. This was done exactly for simple graphs in [KT19, HRW20] and with polylogarithmic error for weighted graphs in [Li21]. Extending the techniques in [KT19, HRW20] to weighted graphs presents significant challenges, and moreover, the algorithm can only polylogarithmically approximately preserve minimum cuts. A remaining challenge is to reduce the polylogarithmic-approximate clusterings to $1 + o(1/\log n)$-approximate so that they can be applied recursively as in [Li21] over $O(\log n)$ many levels. This is an additional challenge that requires building on properties of tree-packings in the presence of a wide range of edge weights to, for example, find sources for local flow computations which identify minimum cuts that cross clusters.

*Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria. See funding information in the acknowledgement section. email: `monika.henzinger@ista.ac.at`

†Simons Institute, UC Berkeley. email: `jmli@alumni.cmu.edu`

‡UC Berkeley. email: `satishr@berkeley.edu`

§Google Research. email: `wadi@google.com`

1 Introduction

The minimum cut problem of finding the partition in a graph with n nodes and m edges which minimizes the number (or weight) of edges between the parts has a long history and many applications. Algorithms were initially derived from algorithms for the minimum $s-t$ cut problem, which finds a partition that separates s and t and minimizes the edge number or weight between the pieces; here one can simply use $n-1$ instances of the minimum $s-t$ cut to find a minimum cut. It is interesting to remind the reader that the minimum $s-t$ cut problem has as a dual, the maximum $s-t$ flow problem, whose flow value equals the minimum cut value, and algorithms are inextricably linked for both.

For minimum cut itself, a directed version of a dual was developed in 1962 by Nash-Williams [NW61], who showed that a packing of arborescences could be found in a directed graph whose size is equal to the size of a related cut. This packing figures in modern algorithms. In particular, the packing algorithm of Gabow [Gab95] makes this duality algorithmically effective. We also use it in our algorithm. Approximate versions using the multiplicative weights methods [PST95] of these ideas are directly useful in our work.

A linear-time deterministic method for producing a "sparse" graph with $O(\lambda n)$ edges that preserves all minimum cuts for a graph with minimum cut size λ was given by Nagamochi and Ibaraki [NI92]. Another structural breakthrough was given by Karger [KAR93] who showed that the remarkably simple randomized algorithm of repeatedly choosing a random edge to collapse, returns a minimum cut with probability $1/n^2$. This yields a proof that the number of minimum cuts is at most $O(n^2)$ which is matched by the number of minimum cuts on a simple cycle. The method can also be used to bound the number of near-minimum cuts. This led to a sequence of randomized algorithms [KS96, BK96, Kar00] with improved running times over the deterministic maximum-flow based approaches.

Furthermore, Karger [Kar00] gave a randomized algorithm that ran in time $O(m \log^3 n)$, which yielded a minimum cut with high probability. The algorithm makes use of the structural insights on the number of cuts in a graph, on tree packings, and on analyzing sampling methods to yield a weighted graph with few edges that almost preserved all minimum cuts. This, along with a packing of trees and a clever dynamic program yielded Karger's bounds.

For deterministic algorithms, the state of the art remained largely tied to maximum flow based methods until the breakthrough work of Kawarabayashi and Thorup [KT19] who developed graph decomposition techniques to give a $\widetilde{O}(m)$ deterministic algorithm for simple graphs, which was improved by Henzinger, Rao, and Wang [HRW20] to $O(m \log^2 n \log\log n)$[1]. Interestingly, the decomposition method's closest progenitor was perhaps the work of Spielman and Teng [ST04] who gave *randomized* near-linear time algorithms for solving linear systems. Moreover, these graph decomposition methods are a central ingredient in the recent *randomized* almost-linear time algorithms for maximum flow [CKL+22]. We note that a central tool in [KT19] was the page-rank algorithm which was replaced by local flow algorithms in [HRW20]. The usefulness of decompositions in both randomized and deterministic settings is notable.

The main weakness of the method of [KT19, HRW20] is that it is limited to *simple* graphs due to several issues, one of which is the fact that any minimum cut of size λ must consist of either a single vertex or must have at least λ vertices. When λ is large[2], this allows the decomposition procedure of [KT19, HRW20] to find clusterings which are "perfect" in the sense that they don't "cross" any cut whose size is close to a minimum cuts. This is one of several obstacles that we must overcome in the weighted setting.

Li [Li21] produced an $m^{1+o(1)}$ deterministic algorithm for *weighted* graphs using deterministic constructions of expander decompositions [LS21, CGL+20] to, in a sense, derandomize Karger's [Kar00] approach to minimum cut. His approach is limited by both the time for constructing hierarchical expander decompositions and the "inefficiency" (relative to [KT19]) of the resulting decompositions, in the sense that the minimum cut in the corresponding collapsed graph might be a factor of $\Omega(\log n)$ larger per level in the hierarchical decomposition than the minimum cut in the original graph. This leads to a super-polylogarithmic multiplicative factor in the size of the minimum cut in the collapsed graph over all levels. Fortunately, through the framework of Karger's randomized algorithm, this multiplicative approximation can be paid for in the running time to compute a so-called *skeleton graph*, from which an exact minimum cut can be computed.

[1]While the running time is better than Karger's randomized algorithm, subsequent work has provided an $O(m \log^2 n)$ time randomized algorithm for minimum cut in simple graphs [GMW20].

[2]When λ is small, the minimum cut can be found using [Gab95].

In this paper, we give an $\tilde{O}(m)$ deterministic algorithm for minimum cut in weighted graphs. While we combine the approaches in [Li21] and those in [KT19, HRW20], we have to address a number of issues in each by devising more efficient algorithms for weighted graphs as well as deal with the more complicated structure of minimum cuts in weighted graphs as compared to simple graphs.

Of independent interest, we provide a structural theorem that there exists a sparse clustering, i.e., a partition of the vertices into set such that there are few edges between the sets, that preserves minimum cuts in weighted graphs with $(1 + 1/\operatorname{polylog}(n))$ multiplicative error. This extends the theorem of [KT19] which gives a perfect clustering for simple graphs and the theorem of [Li21] which gives a clustering with a logarithmic error.

1.1 Our techniques and related work

We discuss in more detail some of the techniques that we build upon along with a description of our contributions.

A classical result of Nash and Williams [NW61] showed that any graph G with minimum cut λ contains a packing of λ arborescences (directed trees) in the directed graph where each edge in G is replaced by an edge in each direction. Gabow [Gab95] provided an efficient $\tilde{O}(\lambda m)$ algorithm that produced the packing. There are randomized approximation algorithms that produce fractional and approximate packings as well [PST95].

As noted previously, in 1996, Karger [Kar96] gave an $O(m \log^3 n)$ randomized min-cut algorithm that takes a graph G and produces a weighted graph with $O(n \log n)$ edges where the minimum cuts are preserved to within a $(1 + \epsilon)$ factor.[3] Then Karger builds a tree packing using the algorithm of Gabow [Gab95] in that weighted graph that consists of $O(\log n)$ trees. In such a tree packing, there must be a tree which intersects the minimum-cut at most twice, and Karger provides an efficient dynamic programming algorithm that given that tree produces a minimum cut.

Karger produces the sparse graph by sampling edges inversely to their "strength", a measure of the connectivity of the endpoints, in a manner that preserves all small cuts and yields a sparse graph. Specifically, the sampling introduces an edge of weight $1/p$ if an edge is sampled with probability p, which preserves the edge on expectation, and sampling by strength preserves all cuts with high probability, which can be argued using the bounds on the number of minimum-cuts and near minimum-cuts as follows from [KS96]. This sampling procedure is very much randomized and an efficient deterministic construction has remained elusive. We note that Karger's algorithm works naturally for weighted graphs.

The deterministic algorithms for simple graphs in [KT19, HRW20] instead generate a sparse graph that preserves all non-trivial minimum cuts [4] by producing a hierarchical clustering. At each level, a set of clusters is produced such that (a) no non-trivial minimum-cut separates any cluster and (b) there are few intercluster edges. Thus, collapsing the graph and recursing ensures that all non-trivial minimum cuts are preserved. The collapsing can be repeated until the graph has $m' = \tilde{O}(m/\delta)$ edges, where δ is the minimum degree, and then Gabow's algorithm [Gab95] is run to find a minimum cut in $\tilde{O}(\lambda \cdot m/\delta) = \tilde{O}(m)$ time.

An example is useful here. First, note that a simple cycle is "easy" in the sense that it is sparse and one can simply run Gabow [Gab95] in linear time. Replacing each vertex with a clique on δ vertices and connecting adjacent cliques along the cycle with matchings produces a "dense" graph with minimum degree $\delta + 2$ and minimum cut size δ. Note that the cliques are not split by any non-trivial minimum cut. The algorithms in [KT19, HRW20] produce a final graph such that each node is created by collapsing a clique. The number of edges, including parallel edges, in the collapsed graph is $O(m/\delta)$ and this graph contains all non-trivial minimum cuts in the original graph. Note that each collapsed graph induces a vertex clustering or partition, where each node in the collapsed graph induces a *piece* in the partition. Inspired by this example, we say a vertex clustering is *non-crossing* if there exists a minimum cut that does not cut any of the pieces. Thus, the algorithms in [KT19, HRW20] produce a non-crossing vertex clustering.

To proceed it is useful to provide a little more detail. The algorithms of [KT19, HRW20] produce the non-crossing clusterings *incrementally.* That is, their algorithm begins with a partition where any minimum cut only overlaps (i.e., crosses) any piece by some volume s, which is initially $m/2$, and then proceeds to iteratively refine that clustering, in each iteration decreasing s by a factor of 2. Once s is smaller than $\Theta(\delta)$, where δ is the minimum vertex degree, they clean up the resulting partition to obtain the property that a non-trivial minimum cut does not cross any of the resulting clusters at all. This last step of obtaining a non-crossing clustering relies

[3]This work is inspired by a sequence of previous papers on randomized approaches that remained far from near linear time. [KS96, BK96, BK15].

[4]A trivial cut consists of just one vertex.

crucially on a local expansion property of simple graphs: that any minimum cut in a simple graph consists of either a single vertex or at least δ vertices and, thus, any non-trivial minimum-cut is extremely sparse for simple graphs. However, this does not hold for weighted graphs: In a weighted graph a non-trivial minimum cut can contain just two vertices for arbitrary values of δ.

Moreover, the recursive techniques face substantial issues for weighted graphs in terms of complexity. The refinement process in [HRW20] performs local flow computations where the progress is proportional to the total weight of the edges that are touched. In simple graphs, the weight and the number of edges are the same and, thus, the total work is near-linear in the number of edges. For weighted graphs, the number of edges and the total weight differ and we, thus, need to devise new algorithms for the refinement process. Another issue in the refinement process is the allocation of sources for the local flows, which also is more complex as weighted graphs do not "expand" with respect to weights as fast as simple graphs expand with respect to degrees. E.g., a degree δ simple graph has δ-neighbors, where a vertex of weighted degree of δ may have only a single neighbor. This is problematic for the local flow computation.

This expansion issue creates an even bigger obstruction for going from "almost" non-crossing as defined by s to completely non-crossing. This barrier remains and we do not produce a clustering such that every cluster is non-crossing in this work.

Fortunately, Li [Li21] provides an ingenious approach for utilizing "approximately non-crossing" clusterings to derandomize the sampling approach of Karger [KS96]. He first shows how to use expander decompositions to produce approximately non-crossing clusterings[5], the deterministic constructions of which take $m^{1+o(1)}$ time. He introduced another measure of non-crossing of cuts for the partition in addition to the volume s of overlap. In particular, his clusterings ensuring that making a cut consistent with a clustering would only increase the size of the cut by a multiplicative $n^{o(1)}$ factor. The idea is to "shift" the cut by adding or removing a small set of vertices across the cut. This is what he calls *uncrossing* a cut. This is how he produces a hierarchical clustering where every minimum cut is well approximated by an arithmetic expression using a *linear* number of the clusters in the hierarchy.

He argued further that sampling a subset of edges that approximately preserves the boundary size of the near-linear number of clusters would then preserve approximately the minimum cuts as well. This contrasts with Karger's sampling which argued about an *exponential* number of cuts. The linear number of cuts allows one to derandomize the sampling as one can explicitly track whether all the cuts are properly represented as one chooses edges to include in the sample. One method to sample edges is to satisfy a weighted sum of the near-linear number of cuts using a multiplicative weights method to ensure that all cuts are properly represented. This method was introduced in [Rag88].

The obstruction for Li [Li21] to obtaining a $\tilde{O}(m)$ running time is both in the use of the expander decomposition and the fact that even a constant multiplicative factor increase in boundary size, much less a polylogarithmic, induced by uncrossing cuts limits the number of recursive levels to a constant. Indeed, a recursion of depth $k = O(\log n)$ would give a disastrous $\log n^{O(\log n)}$ multiplicative increase in the size of the minimum cut, while fewer recursive levels are insufficient to sufficiently sparsify the graph. We note that the use of expander decompositions requires at least a $\Omega(\log n)$ multiplicative increase of the cut size per level of the recursion without a fundamental new understanding of finding sparsest cuts.

We address these problems by using a modified version of the techniques from [KT19, HRW20] which runs faster and is better in that the uncrossing of minimum cuts only introduces an error of $(1 + \epsilon)$ at each level where $\epsilon = 1/\operatorname{polylog}(n)$. Thus, a subset of the edges (sampled deterministically as sketched above) that approximates the cuts in the resulting decomposition yields an accurate representation of all minimum cuts.

As before, we produce these clusterings by first recursively reducing the overlap, s, of any minimum cut to have weighted volume $\tilde{O}(\lambda)$. Producing clustering with overlap of s needs modification, as does understanding the cost of uncrossing at the detail level. The recursion needs significant changes due to the running time analysis having to depend on the number of edges rather than their weight as mentioned above. At this point, cuts that do not cross any of the clusters can increase the size of the minimum cut by a polylogarithmic factor.

We then need to refine these clusters so that the size of the minimum cut increases only by a $(1 + \epsilon)$ factor. To do so, we appeal to several structural properties of minimum cuts to deal with a spectrum of edge

[5]The technical definitions of approximately non-crossing are a bit different than those in [KT19, HRW20] and here, but are similar in consequence.

weights. For "small" clusters (with a polylogarithmic number of vertices), we can use bounds on the number of minimum cuts along with our flow procedures to produce a better clustering and charging arguments like those in [KT19, HRW20]. For large clusters, we have the property that minimum cuts do not overlap them very much which implies that overlapping minimum cuts contain edges with large weights. This allows us to pack few forests into these large weight edges which, in turn, allows us to appropriately select small sets of sources for the local flow procedures that find problematic overlapping minimum cuts. In the end, we prune off overlapping small cuts from a large cluster until the large cluster is certified to have no bad overlapping mininum cuts. We note that the localized flow methods are similar to those used in randomized constructions of expander decompositions [SW19] in addition to those in [HRW20].

2 Preliminaries

We are given an undirected, weighted graph $G = (V, E, w)$. Let λ be the minimum cut in G and let W be the sum of the weights of all edges. We assume that W is polynomial in n. If this is not the case, the stated running time increases by a factor of $O(\log W)$. We use *component* to denote a connected component of $G \setminus E'$ for some $E' \subseteq E$. Given a component C we use ∂C to denote the set of edges called *boundary edges* with exactly one endpoint in C.

Weights. Given two non-overlapping vertex sets A, B, we use $w_G(A, B)$ (or $w(A, B)$ if the graph we refer to clear from the text) to denote the sum of the weight of the edges between A and B and we use $w_G(A)$ (or $w(A)$) to denote the sum of the weighted edge degrees of all vertices in A. Furthermore, we adapt the notations to weighted graphs by explicitly using superscript W to denote the weighted quantities, e.g. $d(v)$ is the number of edges incident to v, $d^W(v)$ is the total weight of edges incident to v, while $\mathbf{vol}(V) = \sum_{v \in V} d(v)$, $\mathbf{vol}^W(V) = \sum_{v \in V} d^W(v)$ for any vertex set V.

Subgraphs. We use $G[A]$ to denote the subgraph of G induced by the vertices in A, while $G\{A\}$ denotes the graph $G[A]$ together with, for every node $u \in A$ with $w_G(u, G \setminus A) > 0$, an added self-loop of u with weight $w_G(u, G \setminus A)$. We will use the subgraph $G\{A\}$ by default when we work on a component A, and the self-loops make $d^W(v)$ the same in $G\{A\}$ as in G for all $v \in A$, so there is no ambiguity when we talk about the weighted degree or volume in a subgraph.

Definition 2.1. (**Strength**) *Let $s, \tilde{\delta} \geq 0$. A component C in a graph G is s-*strong for $\tilde{\delta}$ *if every cut S with cut-size $w(S, G \setminus S) \leq 1.1\tilde{\delta}$ satisfies $\min\left(\mathbf{vol}^W_{G\{C\}}(S \cap C), \mathbf{vol}^W_{G\{C\}}(C \setminus S)\right) \leq s$. We call s the* strength *of a s-strong component. Note that s-strong is always defined with respect to some parameter $\tilde{\delta}$ even though we frequently drop $\tilde{\delta}$ and just use s-strong when it is clear from the context.*

Fact 2.1. *Let $s, \tilde{\delta} \geq 0$. Any component that is a subset of an s-strong component is s-strong, where both use the same $\tilde{\delta}$ in the definition of s-strong.*

Definition 2.2. *For a component $A \subset V$ and parameter $\beta \leq 1$, a set $U \subseteq A$ is β-boundary-sparse in A iff*

$$w(U, A \setminus U) < \beta \min\{w(U, V \setminus A), w(A \setminus U, V \setminus A)\}.$$

Otherwise it is non-β-boundary-sparse in A.

Note that if U is β-boundary-sparse in A, then $A \setminus U$ is also β-boundary-sparse in A, and that both $\emptyset$ and A are non-β-boundary-sparse in A.

Crossing. Given two vertex sets A, B, we say that A and B *cross* if the sets $A \cap B$, $A \cap (V \setminus B)$, $(V \setminus A) \cap B$, and $(V \setminus A) \cap (V \setminus B)$ are all non-empty. Informally, when we say we wish to *uncross* a set A with a set B, we mean replacing A with another set A' that does not cross B.

3 Overview of the algorithm

At a high level, we follow the approach of the $O(|E|^{1+o(1)})$ deterministic minimum cut algorithm by Li [Li21], which first performs a hierarchical expander decomposition of the input graph G, and uses the structure uncovered by the decomposition to derandomize the sampling procedure from Karger's classical randomized near-linear time minimum cut algorithm [Kar00] to construct a skeleton graph. Informally speaking, the skeleton graph approximately preserves the minimum cut from the original graph while being much sparser. Finally, one can use the (deterministic) tree packing and dynamic programming techniques from Karger's result to find the minimum cut. While all other steps take time $\tilde{O}(|E|)$, the bottleneck of Li's algorithm is the deterministic expander decomposition and we replace it with a customized graph decomposition inspired by ideas from minimum cut algorithms for unweighted graphs [KT19, HRW20]. Although our decomposition doesn't provide all the guarantees of an expander decomposition, it takes time $\tilde{O}(|E|)$ and we show that it captures all the essential properties required by the subsequent steps in Li's algorithm. This improves the time complexity of the whole algorithm to be nearly linear. Concretely, the goal of the decomposition is to partition the vertices into subsets, such that there exists a cut of size $(1+\epsilon)\lambda$ that does not cross any of the subsets, where $\epsilon = 1/\operatorname{polylog}|V|$. In the rest of this section, we discuss our graph decomposition routines and outline the main steps of our algorithm. The remaining sections then fill in the details for these steps.

First we get a close approximation of $\tilde{\lambda}$ of λ (i.e. $\lambda \leq \tilde{\lambda} \leq 1.01\lambda$) by guessing the value of λ and then start decomposing the graph using three different procedures. The first decomposition routine is similar to [KT19, HRW20] and decomposes the graph G' into s_0-*strong* components for parameters $\tilde{\delta} = \tilde{\lambda}/1.01$ and $s_0 = O(\tilde{\delta}\tau^2)$ with $\tau = \operatorname{polylog}|V|$. We intuitively think of s_0-strong components as having similar guarantees for near-minimum cuts as expanders in the expander decomposition. Our decomposition is summarized in the following lemma, whose running time is near-linear in the unweighted number of edges in the graph.

Lemma 5.1. (s_0-strong partition) *Given a weighted graph $G = (V, E, w)$, a parameter $\tilde{\delta}$ such that $\tilde{\delta} \leq \min_{v\in V} d^W(v)$ and a parameter $s_0 = \Theta(\tilde{\delta}\log^c|V|)$ for some $c \geq 2$, there is an algorithm that runs in $\tilde{O}(|E|)$ time and partitions the vertex set V into s_0-strong components for $\tilde{\delta}$ such that the total weight of the edges between distinct components is at most $O(\frac{\sqrt{\tilde{\delta}}\log|V|}{\sqrt{s_0}}) \cdot \mathbf{vol}^W(V)$.*

To prove this lemma we need to extend the techniques from [HRW20] to weighted graphs, which poses the challenges discussed in Subsection 1.1. Sections 4 and 5 shows how we overcome them.

In the algorithm we choose $\tilde{\delta} = \tilde{\lambda}/1.01$. In the following we always use

$$s_0 = 10^{11}\tilde{\delta}\tau^2 = 10^{11}\tilde{\lambda}\tau^2/1.01$$

and we use τ to denote $\Theta(\log^{c/2}|V|)$ for some $c \geq 2$. We will call an s_0-strong component a *cluster*. To emphasize that a cluster is s_0-strong, we will often write s_0-strong cluster, even though this is redundant. We then decompose these clusters further. By Fact 2.1 every refinement of a cluster is a cluster so that the two subsequent decompositions are only applied to clusters. These decompositions are novel and constitute our main technical contributions. They are presented in Section 6. The first decomposes each cluster into at most one large-volume cluster and a collection of small-volume clusters. It is summarized in the following lemma, which is proven in Subsection 6.2.

Lemma 6.4. (Large cluster decomposition) *Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.1$ be a parameter and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a (potentially "large") set A_0 and "small" sets $A_1, \ldots, A_r$ such that*

1. *For each $i \in \{1, 2, \ldots, r\}$, we have $\mathbf{vol}^W_G(A_i) \leq (\frac{10s_0}{\epsilon\tilde{\lambda}}) \cdot s_0$.*
2. *The total weight $w(A_0, A_1, \ldots, A_r)$ of inter-cluster edges is at most $O(\epsilon^{-1}) \cdot \partial A$.*
3. *For any set $U \subseteq A$ with $\partial_{G[A]}U \leq 1.01\tilde{\lambda}$ and $\mathbf{vol}^W_G(U) \leq s_0$, there exists $U^* \subseteq U$ with $U^* \cap A_0 = \emptyset$ and $w(U \setminus U^*, V \setminus A) + \partial_{G[A]}U^* \leq (1+\epsilon)\partial_{G[A]}U$.*

The algorithm runs in time $\tilde{O}((\frac{s_0}{\epsilon\tilde{\lambda}})^{O(1)}|E(G[A])|)$.

By our choice of s_0 the running time is $\tilde{O}(|E(G[A])|)$.

By Fact 2.1 every set A_i is s_0-strong. We now call a cluster *small* if it has volume $(\frac{10s_0}{\epsilon\tilde{\lambda}}) \cdot s_0$ and *large* otherwise. As we will choose $\epsilon = O(1/\log^{1.1}|V|)$, a small cluster has volume $O(\tilde{\delta}\tau^4\log^{1.1}|V|) = \tilde{O}(\lambda)$. Since every vertex has degree at least λ, it follows that small clusters contain only $\tilde{O}(1)$ many vertices. Note that the lemma states that the potentially large cluster A_0 has been uncrossed, i.e., every cut U of size at most $1.01\tilde{\lambda}$ can be shifted to a cut U^* such that U^* does not cross A_0 and such that the size of U^* is only a factor of $(1+\epsilon)$ larger than the size of U.

Thus we are left with potentially further decomposing the small clusters, for which we use the second decomposition. It runs in time polynomial in the number of vertices of the cluster, which is why we need the clusters to have only polylog $|V|$ vertices. We will not define the concept of $(1+\epsilon)$-*uncrossable* in property (1) in this section, but informally, for two sets S and A, we say that $S \cap A$ is $(1+\epsilon)$-uncrossable with A if either $\partial(S \setminus A) \approx \partial S$ or $\partial(S \cup A) \approx \partial S$. In other words, we can "uncross" S with A at little additional cost to the size of the boundary.

Lemma 6.1. (Small cluster decomposition) *Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.01$ be a parameter, and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a disjoint union of clusters $A_1 \cup A_2 \cup \cdots \cup A_k = A$ such that*

1. *For any set $S \subseteq A$ of G with $\partial_{G[A]}S \leq 1.01\tilde{\lambda}$, there is a partition $\mathcal{P}$ of the set $\{A_1, A_2, \ldots, A_k\}$ such that for each part $P = \{A_{i_1}, \ldots, A_{i_\ell}\} \in \mathcal{P}$, the set $S \cap \bigcup_{A' \in P} A'$ is non-$(1-\epsilon)$-boundary-sparse in $\bigcup_{A' \in P} A'$.*

2. *The sum of boundaries $\sum_i \partial A_i$ is at most $O(\epsilon^{-3}(\log|A|)^{O(1)}\partial A)$.*

The running time of the algorithm is $O(|A|^9 \operatorname{polylog}|A|)$.

Note that if A is a small cluster, the running time is $\tilde{O}(1)$.

With these decomposition routines, we now outline the algorithm for a weighted graph $G = (V, E)$. The goal of Step 1 and the outer loop of Step 2 is to find a good enough estimation $\tilde{\lambda}$ of the minimum cut value λ. Step 1 find a value λ_0 with $\lambda \leq \lambda_0 \leq 3\lambda$. As Step 2 iterates over all powers of 1.01 between $\lambda_0/3$ and $1.01\lambda_0$, the iteration is executed $O(1)$ times and $\tilde{\lambda}$ can be seen as a "guess" of λ that will at least once assume a value in $[\lambda, 1.01\lambda]$. If $\tilde{\lambda}$ does not belong to $[\lambda, 1.01\lambda]$, an arbitrary cut will be returned, but as the algorithm returns at the end the smallest over all cuts found for all guesses, it suffices that the algorithm finds a correct minimum cut in the iteration when $\tilde{\lambda} \in [\lambda, 1.01\lambda]$. Our graph decomposition procedures are featured in Step 2b, and the subsequent Steps 2c and 2d are mostly the same as the original $O(|E|^{1+o(1)})$ algorithm by Li [Li21] and a presented in Section 7.

1. We first run Matula's $(2+\epsilon)$-approximation algorithm [Mat93] (which can be easily extended to the weighted setting) to get a value λ_0 such that $\lambda \leq \lambda_0 \leq 3\lambda$.

2. For all powers $\tilde{\lambda}$ of 1.01 between $\lambda_0/3$ and $1.01\lambda_0$

   (a) Initialize $j \leftarrow 0$ and $G_0 \leftarrow G$ with vertex set V_0 and edge set E_0

   (b) While there are at least 2 nodes in $G_j = (V_j, E_j)$:

      i. Partition V_j into subsets called clusters that are s_0-strong using the algorithm of Lemma 5.1, for parameters $\tilde{\delta} = \tilde{\lambda}/1.01$ and $s_0 = 10^{11}\tilde{\delta}\tau^2$. Let $\mathcal{C}_j^{s_0}$ be the set of s_0-strong clusters.

      ii. Let $\epsilon = 1/\log^{1.1}|V|$.

      iii. For each resulting cluster C, we further decompose C according to Lemma 6.4 with values $\tilde{\lambda}$ and ϵ. Note that all clusters except possibly A_0 (if it exists) is a small cluster.

      iv. For each small cluster, we further decompose it according to Lemma 6.1 with values $\tilde{\lambda}$ and ϵ.

      v. Let $\mathcal{C}_j = \{C_{j,1}, C_{j,2}, \ldots\}$ be the resulting set of clusters. Collapse every cluster $C \in \mathcal{C}_j$ with $|C| > 1$ into one node and call the resulting graph $G_{j+1} = (V_{j+1}, E_{j+1})$.

      vi. $j \leftarrow j + 1$

   (c) Build a skeleton graph H using the clusters $\{\mathcal{C}_j\}_j$ using the algorithm from Section 7.

   (d) Run Karger's tree packing and dynamic programming algorithm on the skeleton H to find a cut S.

3. Output the cut S with the minimum value found over all guesses of $\tilde{\lambda}$.

Note that every iteration of Step 2b takes time $\tilde{O}(m)$. We will show in Lemma 3.1 that the clustering $\mathcal{C}_j$ guarantees that $\mathbf{vol}^W_{G_{j+1}}(V_{j+1}) \le \mathbf{vol}^W_{G_j}(V_j)/2$ for every $j \ge 0$, so there are $O(\log \mathbf{vol}^W_G(V))$ loop iterations of total time $\tilde{O}(m)$.

For correctness let us first assume that $\tilde{\lambda}$ is between λ and 1.01λ. In that case we will show, for each j, that after Step 2(b)iv, every approximately minimum cut can be "shifted" (or "uncrossed") into another cut that does not intersect any cluster and whose cut value grows at most by factor $(1 + o(1/\log|V|))$. In particular, collapsing such a cluster increases the size of the minimum cut by at most a factor of $(1 + o(1/\log|V|))$, which implies that the minimum cut of G_{j+1} is at most a factor of $(1 + o(1/\log|V|))$ larger than the minimum cut of G_j. Thus, throughout all $O(\log|V|)$ iterations of Step 2b the minimum cut in G_j is at most $1.01\tilde{\lambda}$, which is a property needed for the decomposition algorithms in steps2(b)iii and 2(b)iv to correctly compute the clustering $\mathcal{C}_j$. Using the guarantees of the clusters $\mathcal{C}_j$, the algorithm then constructs a skeleton graph H following the pessimistic estimator approach of [Li21]. Finally, the algorithm runs (the remainder of) Karger's near-linear time minimum cut algorithm on the skeleton H, which is deterministic and returns a minimum cut of G.

The lemmata below summarize the two key measures of progress. First, the total volume halves on each iteration j. Second, any near-minimum cut S can be shifted by $O(\tau^2) = \text{polylog}(n)$ vertices into a near-minimum cut $\widetilde{S}$ that does not cross any cluster in the partition, and, hence, survives the contraction on step 2(b)v.

Lemma 3.1. (Progress per iteration) *For each iteration j, the total volume drops by a factor of at least 2 in each iteration: $\mathbf{vol}^W_{G_{j+1}}(V_{j+1}) \le \mathbf{vol}^W_{G_j}(V_j)/2$.*

Proof. The value of $\tilde{\delta}$ used in Step 2(b)i fulfills the condition of by Lemma 5.1, and, thus, with $c = 6$ the lemma implies that the total weight of edges between clusters is at most $O(\frac{\sqrt{\tilde{\delta}}\log|V|}{\sqrt{s_0}}) \cdot \mathbf{vol}^W_{G'_j}(V_j)$ for $s_0 = \Theta(\tilde{\delta}\tau^2)$ with $\tau = \Theta(\log^3|V|)$. In particular, the sum of the boundaries $\sum_A \partial A$ is at most $O(\frac{\sqrt{\tilde{\delta}}\log|V|}{\sqrt{s_0}}) \cdot \mathbf{vol}^W_{G'_j}(V_j)$.

For each such cluster A, by Lemma 6.4, Step 2(b)iii produces a partition into sub-clusters $A_0, A_1, \dots, A_r$ such that $\sum_i \partial A_i \le O(\epsilon^{-1}) \cdot \partial A$. Therefore, after Step 2(b)iii, the new sum of boundaries of all clusters is $O(\frac{\sqrt{\tilde{\delta}}\log|V|}{\epsilon\sqrt{s_0}}) \cdot \mathbf{vol}^W_{G'_j}(V_j)$. Finally on Step 2(b)iv, by Lemma 6.1, each small cluster A is divided one last time into clusters whose sum of boundaries is at most $O(\epsilon^{-3}(\log|A|)^{O(1)}\partial A)$, bringing the new total sum of boundaries to $O(\frac{\sqrt{\tilde{\delta}}\log|V|(\log\log|V|)^{O(1)}}{\epsilon^4\sqrt{s_0}}) \cdot \mathbf{vol}^W_{G'_j}(V_j)$. Putting everything together, we conclude that the total weight of inter-cluster edges is at most

$$O\left(\frac{\sqrt{\tilde{\delta}}\log|V|(\log\log|V|)^{O(1)}}{\epsilon^4\sqrt{s_0}}\right) \cdot \mathbf{vol}^W_{G_j}(V_j).$$

Recall that $s_0 = 10^{11}\tilde{\delta}\tau^2$, $\tau = O(\log^3|V|)$. Using a sufficiently large constant factor for τ, this bound is at most $\mathbf{vol}^W_{G_j}(V_j)/2$, as promised. ∎

Lemma 3.2. (Structure lemma) *Suppose that Step 2 is run with $\tilde{\lambda} \in [\lambda, 1.01\lambda]$, and fix an iteration j. For any cut $S \subseteq V_j$ with $\partial_{G_j} S \le 1.01\lambda$, there exists a set $S' \subseteq V_j$ such that*

1. *S' does not cross any cluster in $\mathcal{C}_j$,*
2. *$\partial_{G_j} S' \le (1 + 3\epsilon)\partial_{G_j} S$, and*
3. *$\mathbf{vol}^W_{G_j}(S \triangle S') \le O(s_0^2/(\epsilon\lambda))$.*

For the remainder of this section, we prove Lemma 3.2.

Since $\partial_{G_j} S \le 1.01\lambda \le 1.01\tilde{\lambda} \le 1.1\tilde{\delta}$ and each cluster $A \in \mathcal{C}^{s_0}_j$ is s_0-strong for $\tilde{\delta}$, we have $\min\{\mathbf{vol}^W_{G_j}(S \cap A), \mathbf{vol}^W_{G_j}(A \setminus S)\} \le s_0$ for all clusters $A \in \mathcal{C}^{s_0}_j$. For each such cluster A that crosses S, apply property (3) of Lemma 6.4 on either $U = S \cap A$ or $U = A \setminus S$, whichever has smaller volume. By assumption, $\partial_{G_j[A]} U \le 1.01\lambda \le 1.01\tilde{\lambda}$ and $\mathbf{vol}^W_{G_j}(U) \le s_0$, so property (3) of Lemma 6.4 guarantees a set U^* with $U^* \cap A_0 = \emptyset$ and $w_{G_j}(U \setminus U^*, V_j \setminus A) + \partial_{G_j[A]} U^* \le (1+\epsilon)\partial_{G_j[A]} U$. To distinguish the sets among different A, we add a subscript A to U, U^*.

We now shift S into a new set S^* by, intuitively, replacing each of the above U_A with U_A^*. Some care is needed to distinguish the cases $U_A = S \cap A$ and $U_A = A \setminus S$. If $U_A = S \cap A$, then we remove the vertices of $U_A \setminus U_A^*$ from S. If $U_A = A \setminus S$, then we add the vertices of $U_A \setminus U_A^*$ to S. By construction, we only add or remove vertices in U_A of total volume at most $\mathbf{vol}^W_{G_j}(U_A) \le s_0$.

Claim 3.1. $\partial_{G_j} S^* \le (1+\epsilon)\partial_{G_j} S$.

Proof. Consider a cluster $A \in \mathcal{C}_j^{s_0}$ with $U_A = S \cap A$. If $U_A \neq U_A^*$, then removing vertices of $U_A \setminus U_A^*$ results in a subset of edges of $E_{G_j}(U \setminus U^*, V_j \setminus A)$ being added to $\partial_{G_j} S^*$, and within cluster A, the net difference of edges is exactly $\partial_{G_j[A]} S^* - \partial_{G_j[A]} S$. The total net difference of edges from processing cluster A is at most $w_{G_j}(U \setminus U^*, V_j \setminus A) + \partial_{G_j[A]} S^* - \partial_{G_j[A]} S$, which is at most $\epsilon \partial_{G_j[A]} S$ by property (3) of Lemma 6.1. A symmetric argument holds for the case $U_A = A \setminus S$. Finally, over all clusters A, the net differences sum to at most $\epsilon \partial_{G_j} S$. ∎

Claim 3.2. *There are $O(1)$ many clusters for which $U_A^* \neq U_A$.*

Proof. Consider the value of the cut $\partial_{G_j}(U_A \setminus U_A^*) \ge \lambda_{G_j} \ge \lambda$. We can upper bound $\partial_{G_j}(U_A \setminus U_A^*)$ as

$$\begin{aligned}\partial_{G_j}(U_A \setminus U_A^*) &= w_{G_j}(U_A \setminus U_A^*, V_j \setminus A) + \partial_{G_j[A]}(U_A \setminus U_A^*)\\ &\le w_{G_j}(U_A \setminus U_A^*, V_j \setminus A) + \partial_{G_j[A]} U_A + \partial_{G_j[A]} U_A^*\\ &\le (2+\epsilon)\partial_{G_j[A]} U_A,\end{aligned}$$

where the last inequality is by property (3) of Lemma 6.1. It follows that $\partial_{G_j[A]} U_A \ge \frac{1}{2+\epsilon}\partial_{G_j}(U_A \setminus U_A^*) \ge \frac{1}{2+\epsilon}\lambda$. Finally, the edges $\partial_{G_j[A]} U_A$ are disjoint among different clusters A, so there are at most $\partial_{G_j} S / (\frac{1}{2+\epsilon}\lambda) = O(1)$ many clusters for which $U_A^* \neq U_A$. ∎

Claim 3.3. $\mathbf{vol}^W_{G_j}(S \triangle S^*) \le O(s_0)$.

Proof. For each cluster A, we add or remove a subset of U_A, which has volume at most $\mathbf{vol}^W_{G_j}(U_A) \le s_0$. By Claim 3.2, there are at most $O(1)$ many clusters for which $U_A^* \neq U_A$. In each such cluster, we add/remove vertices with total volume at most s_0. For the clusters A with $U_A^* = U_A$, no action is needed. Altogether, we obtain $\mathbf{vol}^W_{G_j}(S \triangle S^*) \le O(1) \cdot s_0$. ∎

After step (2(b)iii), by construction of S^* and Lemma 6.4, the set S^* only crosses small clusters, which have volume $\mathbf{vol}^W_{G_j}(A) \le O(s_0^2/(\epsilon\lambda))$. Each of these clusters is then processed according to Lemma 6.1 in Step 2(b)iv as follows. We shift S^* into a new set S' by, intuitively, uncrossing with each cluster according to Lemma 6.1. For each cluster A that crosses S^*, let $\mathcal{P}$ be the partition guaranteed by property (1) of Lemma 6.1. Recall from Definition 2.2 that a set U is β-boundary-sparse in A if $w(U, A \setminus U) < \beta \min\{w(U, V_j \setminus A), w(A \setminus U, V_j \setminus A)\}$. For each $P \in \mathcal{P}$, define $A_P = \bigcup_{A' \in P} A'$ to avoid clutter. If S^* crosses A_P, then the set $U_P = S^* \cap A_P$ is not $(1-\epsilon)$-boundary-sparse in A_P, so either $w(U_P, A_P \setminus U_P) \ge (1-\epsilon) w(U_P, V_j \setminus A_P)$ or $w(U_P, A_P \setminus U_P) \ge (1-\epsilon) w(A_P \setminus U, V_j \setminus A_P)$. If $w(U_P, A_P \setminus U_P) \ge (1-\epsilon) w(U_P, V_j \setminus A_P)$, then update $S^* \leftarrow S^* \triangle U_P$, i.e., we remove the vertices of U_P from S^*. If $w(U_P, A_P \setminus U_P) \ge (1-\epsilon) w(A_P \setminus U_P, V_j \setminus A_P)$, then update $S^* \leftarrow S^* \triangle (A_P \setminus U_P)$, i.e., we add the vertices of $A_P \setminus U_P$ to S^*.

By construction, we only add or remove vertices in A of total volume at most $\mathbf{vol}^W_{G_j}(A) \le O(\frac{s_0}{\epsilon\lambda}) \cdot s_0$.

Claim 3.4. $\partial_{G_j} S' \le \frac{1}{1-\epsilon}\partial_{G_j} S^*$.

Proof. Consider a cluster A and its partition $\mathcal{P}$ guaranteed by Lemma 6.4. If $w(U_P, A_P \setminus U_P) \ge (1-\epsilon) w(U_P, V_j \setminus A_P)$, then the update $S^* \leftarrow S^* \triangle U_P$ results in a subset of edges of $E_{G_j}(U_P, V_j \setminus A)$ being added to $\partial_{G_j} S^*$, and within A_P, all edges of $E_{G_j}(U_P, A_P \setminus U_P)$ are no longer in $\partial_{G_j} S^*$. So the net increase of $\partial_{G_j} S^*$ is at most $w(U_P, V_j \setminus A_P) - w(U_P, A_P \setminus U_P) \le \frac{1}{1-\epsilon} w(U_P, A_P \setminus U_P) - w(U_P, A_P \setminus U_P) \le (\frac{1}{1-\epsilon} - 1) w(U_P, A_P \setminus U_P)$. The argument for the update $S^* \leftarrow S^* \triangle (A_p \setminus U_P)$ is symmetric and has the same upper bound of $(\frac{1}{1-\epsilon} - 1) w(U_P, A_P \setminus U_P)$ for the net increase of $\partial_{G_j} S^*$.

Summed over all clusters A and parts $P \in \mathcal{P}$, the total net increase of $\partial_{G_j} S^*$ is at most $(\frac{1}{1-\epsilon} - 1)\sum_{A,P} w(U_P, A_P \setminus U_P)$, which is at most $(\frac{1}{1-\epsilon} - 1)\partial_{G_j} S^*$ since the edges in $E_{G_j}(U_P, A_P \setminus U_P)$ are disjoint over all A, P. It follows that $\partial_{G_j} S' \le \frac{1}{1-\epsilon}\partial_{G_j} S^*$. ∎

Claim 3.5. $\mathbf{vol}^W_{G_j}(S'\triangle S^*) \le O(s_0^2/(\epsilon\lambda))$.

Proof. Suppose that S^* crosses some A_P for a small cluster A and a partition $\mathcal{P}$ returned by Step 2(b)iv. Recall that $U_P = S^* \cap A_P$ and that Lemma 6.1 guarantees that either $w(U_P, A_P \setminus U_P) \ge (1-\epsilon)w(U_P, V_j \setminus A_P)$ or $w(U_P, A_P \setminus U_P) \ge (1-\epsilon)w(A_P \setminus U, V_j \setminus A_P)$. In the first case, consider the value of the cut $\partial_{G_j} U_P \ge \lambda_{G_j}$. We can upper bound $\partial_{G_j} U_P$ as

$$\partial_{G_j} U_P = w(U_P, V_j \setminus A_P) + w(U_P, A_P \setminus U_P) \le \frac{1}{1-\epsilon} w(U_P, A_P \setminus U_P) + w(U_P, A_P \setminus U_P) = \frac{2-\epsilon}{1-\epsilon} w(U_P, A_P \setminus U_P).$$

It follows that $w(U_P, A_P \setminus U_P) \ge \frac{1-\epsilon}{2-\epsilon}\partial_{G_j} U_P \ge \frac{1-\epsilon}{2-\epsilon}\lambda_{G_j} \ge \frac{1-\epsilon}{2-\epsilon}\lambda$.

The edges $E(U_P, A_P \setminus U_P)$ are disjoint among pairs (A, P), so there are at most $\partial_{G_j} S^*/(\frac{1-\epsilon}{2-\epsilon}\lambda) \le (1+\epsilon)\partial_{G_j} S/(\frac{1-\epsilon}{2-\epsilon}\lambda) = O(1)$ many pairs (A, P) for which any action is needed. For each, we add or remove vertices of total volume at most $\mathbf{vol}^W_{G_j}(A) = O(s_0^2/(\epsilon\tilde{\lambda}))O(s_0^2/(\epsilon\lambda))$, and the claim follows. ◻

We now verify the requirements of Lemma 3.2:

1. By construction, S' does not cross any cluster in $\mathcal{C}_j$.
2. $\partial_{G_j} S' \le \frac{1}{1-\epsilon}\partial_{G_j} S^* \le \frac{1}{1-\epsilon}\cdot(1+\epsilon)\partial_{G_j} S \le (1+3\epsilon)\partial_{G_j} S$.
3. $\mathbf{vol}^W_{G_j}(S\triangle S') \le \mathbf{vol}^W_{G_j}(S\triangle S^*) + \mathbf{vol}^W_{G_j}(S^*\triangle S') \le O(s_0) + O(s_0^2/(\epsilon\lambda)) \le O(s_0^2/(\epsilon\lambda))$.

This concludes the proof of Lemma 3.2

4 Unit-Flow Algorithm

We will consider flow problems extensively in our graph decomposition procedure, and we start with some notations. Formally, a *flow problem* is defined with a *source function*, $\Delta : V \to \mathbb{R}_{\ge 0}$, a *sink function*, $T : V \to \mathbb{R}_{\ge 0}$, and edge capacities $c(\cdot)$. We say that v is *a sink of capacity* x if $T(v) = x$. To avoid confusion with the way flow is used, we use *supply* to refer to the substance being routed in flow problems.

For the sake of efficiency, we will not typically obtain a full solution to a flow problem. We will compute a *pre-flow*, which is a function $f : V \times V \to \mathbb{R}$, where $f(u,v) = -f(v,u)$. A pre-flow f is *source-feasible* with respect to source function Δ if $\forall v : \sum_u f(v,u) \le \Delta(v)$. A pre-flow f is *capacity-feasible* with respect to $c(\cdot)$ if $|f(u,v)| \le c(e)$ for $e = \{u,v\} \in E$ and $f(u,v) = 0$ otherwise. We say that f is a *feasible pre-flow* for flow problem Π, or simply a *pre-flow* for Π, if f is both source-feasible and capacity-feasible with respect to Π.

For a pre-flow f and a source function $\Delta(\cdot)$, we extend the notation to denote $f(v) \overset{\text{def}}{=} \Delta(v) + \sum_u f(u,v)$ as *the amount of supply ending at* v *after* f. Note that $f(v)$ is non-negative for all v if f is source-feasible. When we use a pre-flow as a function on vertices, we refer to the function $f(\cdot)$, and it will be clear from the context what $\Delta(\cdot)$ we are using. If in addition, $\forall v : f(v) \le T(v)$, the pre-flow f will be a *feasible flow (solution)* to the flow problem Π.

We denote $\text{ex}(v) \overset{\text{def}}{=} \max(f(v) - T(v), 0)$ as the excess supply at v, and we call the part of the supply below $T(v)$ as the supply routed to the sink at v, or *absorbed* by v. We call the sum of all the supply absorbed by vertices, $\sum_v \min(f(v), T(v))$, the total supply routed to sinks. Finally, given a source function $\Delta(\cdot)$, we define $|\Delta(\cdot)| \overset{\text{def}}{=} \sum_v \Delta(v)$ as the total amount of supply in the flow problem. Note the total amount of supply is preserved by any pre-flow routing, so $\sum_v f(v) = |\Delta(\cdot)|$ for any source-feasible pre-flow f.

In this section, we consider a particular family of flow problems that is useful in our graph decomposition, and discuss the algorithm we use to compute pre-flows. The flow problem we consider comes with a source function $\Delta(\cdot)$, a weighed undirected graph G with edge weights w_e's, an edge capacity parameter U such that $c_e = w_e \cdot U$ for all edges, and $T(v) = d^W(v)$ for all vertices. As a side note, for the actual flow problems we solve in our graph decomposition procedure, all the quantities (i.e. source supply, edge and sink capacities) are in the same unit whose value can vary across different flow problems, but the flow algorithm should be completely oblivious to the value of the unit. Nonetheless, in the discuss of our flow computation we may refer to supply of amount x as x units of supply.

To solve our flow problems, we use the *Unit-Flow* algorithm, which was designed in [HRW20] for unweighted graphs, and we adapt it to weighted graphs. The algorithm follows the classic Push-belabel framework by Goldberg and Tarjan [GT88], which at a high level works as follows. The algorithm maintains an integer label (or height) $l(v)$ for each node v starting at 0. If there is excess on any node, the algorithm picks one such node and tries to send its excess to any neighbour of smaller label while obeying the edge's residual capacity (a.k.a *Push*), and if there is no possible Push for a node with excess, the algorithm raises the label of the node by 1 (a.k.a *Relabel*). The standard Push-relabel algorithm for max flow keeps carrying out Push or Relabel until it finds a flow (i.e. no excess supply on any node).

The Unit-Flow algorithm works exactly the same except one difference. In particular, Unit-Flow takes as input a cap h on the height, and when Push-relabel needs to raise a node's label from h to $h+1$, Unit-Flow will instead leave the node at height h without working further on the node's excess supply. Note this is also the only case that Unit-Flow leaves excess on nodes, so when the algorithm stops, if there is any node with excess, the node must have label h. Since the algorithm description and the running time analysis follow quite mechanically from the Push-relabel algorithm in [GT88] using the dynamic-tree data structuer [ST81], in this section we only summarize the result of Unit-Flow that we will use in our later graph decomposition. The proof of the main theorem also follows largely from the original analysis of unweighted Unit-Flow in [HRW20], so we defer most of the implementation details and analysis of our weighted Unit-Flow to the Appendix A.

Upon termination of the algorithm, it returns (1) a pre-flow f, (2) labels $l(\cdot)$ (i.e. height) on vertices, and (3) a possibly empty cut A. More specifically, *Unit-Flow* will either successfully route a large amount of supply to sinks and return an empty cut, or it returns a low conductance cut using the labels. Formally, we have the following theorem.

THEOREM 4.1. *Given $G, \Delta, h, U, \beta \geq 2$ such that $\Delta(v) \leq \beta d^W(v)$ for all v,* Unit-Flow *terminates with a pre-flow f, where we must have one of the following three cases*

(1) f is a feasible flow, i.e. $\forall v : \mathrm{ex}(v) = 0$. All units of supply are absorbed by sinks.

(2) f is not a feasible flow, but $\forall v : f(v) \geq d^W(v)$, i.e., at least $\mathbf{vol}^W(G)$ units of supply are absorbed by sinks.

(3) If f satisfies neither of the two cases above, denote T as the set of nodes whose labels are h (i.e. largest possible label), and assume $h \geq \ln \mathbf{vol}^W(G)$, then we can find a cut $(A, \bar{A})$ such that

(a) $T \subseteq A$, and $(\beta + U)d^W(v) \geq f(v) \geq d^W(v)$ for all $v \in T$, and $f(v) = d^W(v)$ for all $v \in A \setminus T$, and $f(v) \leq d^W(v)$ for all $v \notin A$.

(b) The cut's conductance $\Phi(A) = \frac{W(A, V\setminus A)}{\min\left(\mathbf{vol}^W(A), \mathbf{vol}^W(G) - \mathbf{vol}^W(A)\right)} \leq \frac{20 \ln \mathbf{vol}^W(G)}{h} + \frac{\beta}{U}$.

(c) Let $\mathrm{ex}(T)$ be the total amount of excess supply (which can only be on nodes in T), i.e. $\mathrm{ex}(T) = \sum_v \max\left(0, f(v) - d^W(v)\right)$, we have $\mathbf{vol}^W(A) \geq \frac{\mathrm{ex}(T)}{(\beta-1)+10U\ln\left(\mathbf{vol}^W(G)\right)/h}$

Let $C = \left\{v : f(v) \geq d^W(v)\right\}$ be the set of saturated nodes when Unit-Flow finishes, the running time for Unit-Flow *is $O(\mathbf{vol}(C) \cdot h \cdot \log \mathbf{vol}(C))$. Moreover, the above guarantees hold in the case when Δ is incremental, i.e., after Unit-Flow finishes working with the supply from the current $\Delta(\cdot)$, new source supply can be added on nodes to increase $\Delta(\cdot)$, and the theorem holds when Unit-Flow finishes with any interim $\Delta(\cdot)$.*

Note that the running time only depends on the *unweighted* volume $\mathbf{vol}(C)$ and *not* on the weighted volume $\mathbf{vol}^W(C)$.

5 Guaranteeing s_0-strength

Our procedure will make progress on the graph by breaking it into s_0-strong components, i.e., components such that if we consider any nearly min cut in the graph, any component will be mostly on one side of the cut up to some small error. We prove in this section the following result.

LEMMA 5.1. (s_0-STRONG PARTITION) *Given a weighted graph $G = (V, E, w)$, a parameter $\tilde{\delta}$ such that $\tilde{\delta} \leq \min_{v \in V} d^W(v)$ and a parameter $s_0 = \Theta(\tilde{\delta} \log^c |V|)$ for some $c \geq 2$, there is an algorithm that runs in $\tilde{O}(|E|)$ time and partitions the vertex set V into s_0-strong components for $\tilde{\delta}$ such that the total weight of the edges between distinct components is at most $O(\frac{\sqrt{\tilde{\delta}} \log |V|}{\sqrt{s_0}}) \cdot \mathbf{vol}^W(V)$.*

Throughout this section, when we talk about s-strong components for any $s \geq s_0$, the definition of strength is with respect to the parameter $\tilde{\delta}$, so we only worry about any cut S with $w(S, G \setminus S) \leq 1.1\tilde{\delta}$. We denote $\tau = \Theta(\sqrt{s_0/\tilde{\delta}}) = \Theta(\ln^{c/2}|V|)$ as the parameter, such that in the section we always break components along cuts of conductance at most $O(1/\tau)$. Moreover, we need the following definitions.

Definition 5.1. *Given a component C in a graph G we say that a cut $(A, C \setminus A)$ is* relatively balanced *if $\min\left(\mathbf{vol}^W(A), \mathbf{vol}^W(C \setminus A)\right) \geq \Omega\left(\frac{\mathbf{vol}^W(C)}{\tau^2 \ln m_G}\right)$ where all volumes refer to the graph $G\{C\}$ (which are the same as in G due to the added self-loops).*

We start with some input graph G with m_G edges and minimum weighted node degree at least $\tilde{\delta}$. By definition G is $\mathbf{vol}^W(G)/2$-strong. For a component C that is already certified to be s-strong for some $s \geq s_0$, our algorithm will try to certify C is $s/2$-strong, but during the process it may find a low conductance cut in C and break up C. When we break a s-strong component C into $(C', C \setminus C')$, by definition we already certify C' is $\min(s, \mathbf{vol}^W(C')/2)$-strong (and similarly for $C \setminus C'$). The running time to certify that C is $s/2$-strong or to find a low conductance cut will be $\tilde{O}(m_C)$, where m_C is the number of edges in C. We will iteratively work on the graph (by cutting or certifying) and stop when all connected components are certified to be s_0-strong where for concreteness we use the explicit value of $s_0 = 10^{11}\tilde{\delta}\tau^2$.

Moreover, for the low conductance cut we use to divide a component C, either the cut is relatively balanced (i.e., the weighted volume of both C' and $C \setminus C'$ is at most $(1 - \frac{1}{\tau^2 \ln m_G})\mathbf{vol}^W(C)$), or we can certify that the larger side of the cut is $s/2$-strong. This guarantees that the total running time to decompose the graph G into s_0-strong components is near-linear in the number of edges of the graph (i.e. m_G).

For now, we will work with the graph $G\{C\}$ of one connected component C that is certified to be s-strong for some $s > s_0$. We use m_C, W_C to denote the number of edges and weighted volume of $G\{C\}$ respectively.

5.1 Flow Problems and Excess Scaling

We will construct a sequence of flow problems, and have a total of (up to) $L = O(\ln m_C)$ stages for each flow problem. The basic structure is that each flow problem will start at stage 0 and be associated with a source node at stage 0.

Technically for each flow problem, we need its associated source node to reserve a distinct $\tilde{\delta}$ adjacent edge weight for that flow problem, and thus we can (and may have to) use the same node v as the source node of multiple flow problems if $d^W(v) \gg \tilde{\delta}$ to accommodate a large enough number of flow problems in total. However, for the sake of notation simplicity, it is easier if we can associate each flow problem with a distinct source node (and thus refer to it by its source node), and to make sure the graph can still support enough flow problems, we assume each node v has $\tilde{\delta} \leq d^W(v) \leq 2\tilde{\delta}$. The lower bound automatically holds, and for any node v with $d^W(v) > 2\tilde{\delta}$, we can view it as several virtual nodes such that they (virtually) partition the total weight of v's adjacent edges in a way that the weighted degree of each is between $\tilde{\delta}$ and $2\tilde{\delta}$. This guarantees that we have at least $\frac{W_C}{2\tilde{\delta}}$ distinct (virtual) nodes to use as source nodes. Consequently, we WLOG make the following assumption.

Assumption 5.1. *We can pick up to $\frac{W_C}{2\tilde{\delta}}$ distinct nodes in the graph as source nodes of our flow problems in stage 0, and all source nodes have weighted degree at least $\tilde{\delta}$.*

Stages. For the flow problem associated with a node v, we will inject a certain amount of source supply at v in stage 0 and try to spread it to sinks iteratively in stages. In each stage we use Unit-Flow, which may terminate with some excess supply. If the total excess supply is at most a $O(1/\ln m_C)$ fraction of the total amount of source supply, we will call the flow problem *successful for that stage* (otherwise *unsuccessful for that stage*) and advance the flow problem into the next stage. We keep working on a flow problem until it is unsuccessful at some stage or we finish successfully through all L stages. We then pick a new source node and work on the flow problem starting with the new node at stage 0, and keep doing this until enough flow problems are successful at any particular stage. The flow problems happen sequentially, i.e. we work on a flow problem through stages $0, 1, \ldots$ until it is unsuccessful at some stage, and then start a new flow problem. Moreover, every flow problem is associated with a distinct (stage-0) source node.

Batches. We will group the sequence of flow problems into batches, where we keep adding flow problems into the current batch until (a) we find enough flow problems successfully finishing at stage L in the current batch or (b) the total number of flow problems in the current batch reaches a limit. In case (a) we start a new batch, and in Case (b) we will show that we can find a relatively balanced low conductance cut.

Capacity sharing. We will run multiple flow problems of the same stage on the same copy of the graph, which means the pre-flows share the edge and sink capacities in the same graph. More specifically,

- All the flow problems (over all batches) in stage 0 will share a single copy of the graph (i.e., the edge and sink capacities). That is, the edge capacities and sink capacities are constraints over the total flow routed by *all* the flow problems in stage 0.

- For any later stage $i \in \{1, \dots, L\}$, all the flow problems *of the same batch in stage i* will run on a single copy of the graph, while flow problems of different batches or of different stages have separate copies of the graph (i.e. , do not share their edge and sink capacities).

Flow problems. We now fill in more details. For a flow problem associated with source node v in batch i, we denote its source supply function at stage j as $\Delta^v_{i,j}(\cdot)$. At stage 0, we have $\Delta^v_{i,0}(v') = 10^7\tilde{\delta}\tau$ if $v' = v$ and 0 otherwise (i.e. start in stage 0 with source supply only at node v). As in the excess scaling algorithm in [HRW20], *all the quantities for a flow problem (i.e. source supply, sink capacity, and edge capacity) in a stage are specified in some unit.* For stage i, the unit is

$$\mu_i = \lceil \frac{2s}{2^i 10^7 \tilde{\delta}\tau} \rceil,$$

and in particular the total number of stages L equals the smallest value of i by which the unit becomes at most 1. Since s is at most $W_C/2$, we know $L \le \ln W_C = O(\ln m_C)$ if edge weights are bounded by polynomials of m. For stage j, the edge capacity (i.e. total net supply routed over the edge by all flow problems sharing the same copy of the graph) of e is $U_j \cdot w_e$ units, where w_e is the edge weight of e. We use $U_0 = 10^8\tau^2$ and $U_j = 100\tau$ for $j \ge 1$. The total sink capacity (shared across all flow problems using the same copy of the graph) of a node v is always its weighted degree $d^W(v)$ units.

We consider all the flow problems sharing the same copy of the graph in aggregate as one flow problem, and denote the (aggregate) source supply function as $\Delta_{i,j}(\cdot)$ for the aggregated problem corresponding to batch i and stage j; for stage 0 since there is one copy of the graph over all batches, the function $\Delta_{i,0}(\cdot)$ is only for convenience in the analysis, and we denote the actual aggregate source supply function as $\Delta_0(\cdot)$. For i, j, the source function $\Delta_{i,j}(\cdot)$ is considered as incremental as we add more flow problems to the aggregate problem, i.e. we add $\Delta^v_{i,j}(\cdot)$ to $\Delta_{i,j}(\cdot)$ when a flow problem in batch i associated with node v reaches stage j, and we solve each aggregate flow problem with one run of the Unit-Flow algorithm with incremental source supply. Because of how Unit-Flow works, if some supply introduced by $\Delta^v_{i,j}(\cdot)$ is absorbed in a sink during the execution of Unit-Flow, it will remain absorbed throughout the execution even with additional source supply introduced later. Similarly, if some supply is left as excess supply after Unit-Flow terminates with some interim $\Delta_{i,j}(\cdot)$, the excess must be at a node already at height h, and it will remain as excess even with additional source supply added later. In particular, this means whenever Unit-Flow finishes with the interim aggregate flow problem after we add some $\Delta^v_{i,j}(\cdot)$ to it, we can tell at that point whether the flow problem of v is successful for stage j or not, and we know which of the supply introduced in $\Delta^v_{i,j}(\cdot)$ will be absorbed or be excess (without even knowing the complete aggregate flow problem in future).

The source supply function $\Delta^v_{i,j+1}(\cdot)$ is defined to be double the amount of absorbed supply on each node, more specifically it is defined as follows: Let $f^v_{i,j}(v')$ be the total amount of supply introduced by $\Delta^v_{i,j}(\cdot)$ that gets routed to any node v' by Unit-Flow, and let $\mathrm{ex}^v_{i,j}(v')$ be the amount of excess (from that flow) routed to v', then formally we set

$$\Delta^v_{i,j+1}(v') = 2\left(f^v_{i,j}(v') - \mathrm{ex}^v_{i,j}(v')\right) \qquad \forall v'.$$

Note that the excess also depends on the earlier flow problems using this copy of the graph since the sink capacities are shared. Recall that each flow problem starts with supply $10^7\tilde{\delta}\tau$ (in unit μ_0) at stage 0, and if all supply has been carried over into the next stage (i.e. no excess) since stage 0, the amount of supply at stage j would increase to $2^j 10^7\tilde{\delta}\tau$ (in unit μ_j since $\mu_0 = 2^j\mu_j$). We call a flow problem *successful at stage j* if at the end of stage j, the amount of supply for the flow problem being absorbed by sinks, i.e. $\sum_{v'} f^v_{i,j}(v')$, is at least $(1 - \frac{j+1}{10(L+1)})2^j 10^7\tilde{\delta}\tau$. Equivalently, a flow problem reaching stage $j \ge 1$ loses at most a $\frac{j}{10(L+1)} = O(j/\ln m_C)$ fraction of the initial source supply as excess over all previous stages. The absorbed supply on nodes after removing excess after stage j would serve as the source supply if we proceed the flow problem to the next stage $j+1$, and the amount would double (i.e. the factor of 2 in the formula above for $\Delta^v_{i,j+1}(\cdot)$) as we switch to the unit $\mu_{j+1} = \mu_j/2$.

In our Unit Flow computation, we will use $h = \Theta(\tau^2 \ln m_G)$ for stage 0, and we put $10^7\tilde{\delta}\tau$ units of source supply on the source node for every flow problem. As the source nodes are distinct for different flow problems and all nodes have weighted degree at least $\tilde{\delta}$, we know the ratio $\Delta_0(v)/d^W(v)$ (i.e. β in Theorem 4.1) is at most $O(\tau)$. For all stages $j \geq 1$, we will use $h = \Theta(\tau \ln m_G)$ and since we only carry absorbed supply into the next stage, we have $\beta \leq 2$ in later stages. Note the 2 comes from the scaling of unit value from stage j to $j+1$.

Termination of the Certification Procedure: Cut or Certification. Whenever we finish the (incremental) Unit-Flow computation of a flow problem in a stage, we perform two checks as described below. If none of the checks passes, we proceed the flow problem to the next stage $j+1$ if it's successful for the current stage $j < L$, or start a new flow problem with a new sourve vertex at stage 0 if the current flow problem is not successful or it finishes stage $j = L$.

Check 1*:* Recall that if a flow problem is successful after L stages, it has lost at most a $\frac{L}{10(L+1)} \leq \frac{1}{10}$ fraction of its initial source supply. Thus, it routes at least $0.9 \cdot 10^7\tilde{\delta}\tau \cdot \mu_0 = 1.8s$ source supply to sinks. When we stop with a particular flow problem successfully at stage L, we check if there are already $\frac{W_C}{10s}$ successful flow problems at stage L in the current batch.

If so, we call the current batch *successful* and start a new batch of flow problems unless we already have 5000τ successful batches. If we succeed in constructing 5000τ successful batches, we can certify that C (or a constant fraction of it) is s'-strong with $s' = s/2$ (see the proof in Section 5.3).

Check 2*:* If Check 1 fails, we check if the result of Unit-Flow for the (aggregate) flow problem of any stage $j \geq 0$ in the current batch has more than $\frac{W_C}{10^6\tau(L+1)}$ excess supply in total. As soon as this happens, we call this batch *unsuccessful* and we will show in Section 5.2 that we can find a *relatively balanced* low conductance cut in component C, i.e., both sides have weighted volume $\Omega(\frac{W_c}{\tau^2 \ln m_G})$. Thus, we stop the certification procedure, divide C along the cut, leave the cut edges as inter-cluster edges, and continue the certification procedure on each of the two components, which are both s-strong by definition since C is s-strong.

We show that the second stopping condition guarantees that the flow problems are well-defined. We start with the following lemma bounding the maximum number of flow problems we may run in a batch and stage (whether the flow problem is successful or not in the stage).

Lemma 5.2. *For any batch i we will work on at most $\frac{W_C}{10^{11}\tilde{\delta}\tau^2}$ flow problems in stage* 0*, and at most $\frac{W_C}{2\cdot 2^j 10^7\tilde{\delta}\tau}$ flow problems in any stage $j \geq 1$.*

Proof. We start with proving the claim for stage 0. Assume by contradiction that we have worked on $\frac{W_C}{10^{11}\tilde{\delta}\tau^2}$ flow problems in any batch i, but neither of the two stopping conditions are satisfied. In particular, this means that for every stage $j \geq 0$ the total excess in the Unit-Flow result of stage j is less than $\frac{W_C}{10^6\tau(L+1)}$. We know that in order for a flow problem to reach stage j but not stage $j+1$, the flow problem must lose at least $\frac{2^j 10^6\tilde{\delta}\tau}{L+1}$ units (in unit μ_j) of supply in stage j alone (i.e. as excess supply). Thus, dividing the upper-bound of the total excess in stage j shows that, in any stage j, the number of unsuccessful flow problems is at most $\frac{W_C}{2^j 10^{12}\tilde{\delta}\tau^2}$. Hence, the total number of flow problems that are unsuccessful over all stages $j \geq 0$ is at most $\frac{2W_C}{10^{12}\tilde{\delta}\tau^2}$. As we have started $\frac{W_C}{10^{11}\tilde{\delta}\tau^2}$ flow problems in stage 0, we must have at least $\frac{W_C}{10^{12}\tilde{\delta}\tau^2} = \frac{W_C}{10s_0} \geq \frac{W_C}{10s}$ successful flow problems in stage L, which contradicts the assumption that we haven't finished this batch.

The claim for any $j \geq 1$ follows the similar argument. In particular, our analysis above says that the number of flow problems that can be unsuccessful over all stages $j' \geq j$ is at most $\frac{2W_C}{2^j 10^{12}\tilde{\delta}\tau^2}$, and if we already have $\frac{W_C}{2\cdot 2^j 10^7\tilde{\delta}\tau}$ flow problems reaching stage j, we must have at least $\frac{W_C}{4\cdot 2^j 10^7\tilde{\delta}\tau}$ of them being successful in stage L. Since $2^j \leq 2^L = \frac{2s}{10^7\tilde{\delta}\tau}$, we have $\frac{W_C}{4\cdot 2^j 10^7\tilde{\delta}\tau} \geq \frac{W_C}{4\cdot 2^L 10^7\tilde{\delta}\tau} = \frac{W_C}{8s} \geq \frac{W_C}{10s}$, which again is a contradiction.
□

With the above result, we can show below that there will be enough distinct nodes in C that we can use as the source node for flow problems in stage 0. Furthermore, we show the aggregate flow problem for any batch and stage has at most $W_C/2$ units of supply. This ensures we must get case (1) or (3) in Theorem 4.1, and if we get case (3) then A (i.e. the side with excess) must be the smaller side of the cut.

Lemma 5.3. *There are enough distinct nodes to serve as source nodes in stage* 0*, and in every stage the total source supply is at most $W_C/2$ in any aggregated flow problem.*

Proof. We show first that there are enough source nodes in stage 0 to support all the (successful or not) flow problems. By Lemma 5.2 we have at most $\frac{W_C}{10^{11}\tilde{\delta}\tau^2}$ flow problems in a batch in stage 0, so we need at most $\frac{5000\tau W_C}{10^{11}\tilde{\delta}\tau^2} = \frac{5W_C}{10^8\tilde{\delta}\tau}$ nodes to serve as the source node in stage 0 over all the 5000τ batches. Recall we have WLOG Assumption 5.1, so we have up to $\frac{W_C}{2\tilde{\delta}} > \frac{5W_C}{10^8\tilde{\delta}\tau}$ nodes to use, which is enough in this case.

Recall that quantities of supply are in the unit μ_j when we discuss flow in stage j. We know the total sink capacity in the graph of stage 0 (which is shared across all batches) is more than twice the total source supply as follows. We calculated above that we have at most $\frac{5W_C}{10^8\tilde{\delta}\tau}$ flow problems in stage 0, and we add $10^7\tilde{\delta}\tau$ source supply in each flow problem, so the total supply is at most $W_C/2$. Note the total sink capacity is W_C in the shared graph.

Now we show the total sink capacity in the graph of stage $j \geq 1$ for any batch i is more than twice of the total supply. Since we work on at most $\frac{W_C}{2\cdot 2^j 10^7\tilde{\delta}\tau}$ flow problems in stage j by Lemma 5.2, and each flow problem can have at most $2^j 10^7\tilde{\delta}\tau$ units (in unit μ_j) of source supply in stage j since we start with $10^7\tilde{\delta}\tau$ units (in unit μ_0) in stage 0, and only the absorbed supply goes into the next stage while the value of one unit being halved. Thus the total supply $|\Delta_{i,j}(\cdot)| \leq W_C/2$ through the procedure for any i, j. ◻

5.2 The Cut Case In this section we show that if in any batch i and stage j, after Unit-Flow finishes with a flow problem, if the second stopping condition (i.e. Check 2) is satisfied, then we can find a relatively balanced low conductance cut. Recall in this case we have at least $\frac{W_C}{10^6\tau(L+1)} = \Omega(\frac{W_C}{\tau \ln m_G})$ excess in the Unit Flow result for the aggregated flow problem of batch i stage j.

LEMMA 5.4. *After (incrementally) running Unit-Flow on a flow problem, if we have at least $\Omega(\frac{W_C}{\tau \ln m_G})$ excess in the flow for some stage $j \geq 0$, then we can find a cut $(A, C \setminus A)$ of conductance $O(\frac{1}{\tau})$ such that*

$$\min\left(\mathbf{vol}^W(A), \mathbf{vol}^W(C \setminus A)\right) \geq \Omega\left(\frac{W_C}{\tau^2 \ln m_G}\right).$$

Proof. We will apply Theorem 4.1 to it to show the result. Note that the flow problem cannot fulfill Case (1) of the theorem since there is excess remaining in the current pre-flow solution, and we also cannot get case (2) as there is not enough supply to saturate all sinks since $|\Delta_{i,j}(\cdot)| \leq W_C/2$ for all i and j by Lemma 5.3. Thus, we must get a cut as in case (3). We look at the parameters of our flow problems show that the conductance and volume properties in the lemma follow from Theorem 4.1(3)(b, c).

In stage 0, the source supply we put on any node v is $10^7\tilde{\delta}\tau$, and since $d^W(v) \geq \tilde{\delta}$ we know the β in Theorem 4.1 is $O(\tau)$, and U is $10^8\tau^2$. Recall that we use $h = \Theta(\tau^2 \ln m_G)$ in stage 0, so Theorem 4.1(3)(b) guarantees the conductance of the cut is $O(\frac{1}{\tau})$. By Lemma 5.3, we have at most $W_C/2$ supply in the aggregate flow problem in each Unit-Flow computation, so the side A in the returned cut must be the smaller side in terms of weighted volume. Since $\text{ex}(T) \geq \Omega(\frac{W_C}{\tau \ln m_G})$, by Theorem 4.1(3)$(c)$ we know $\mathbf{vol}^W(A) \geq \Omega\left(\frac{W_C}{\tau^2 \ln m_G}\right)$ from our choice of U, h, β.

It is straightforward to do the same calculation for stages $j \geq 1$ to show the conductance and volume guarantees, where we have $\beta = 2, U = 100\tau, h = \Theta(\tau \ln m_G)$ in these stages. ◻

Edges on the cut. When we find a relatively balanced low conductance cut as above, we will break the C along the cut into A and $C \setminus A$. The edges crossing the cut are duplicated to exist in both $G\{A\}$ and $G\{C \setminus A\}$ (as self-loops) by our definition of these sub-graphs. Note that if C is s-strong, $G\{A\}$ and $G\{C \setminus A\}$ must also be s-strong, even though we add a copy of each cut edge to both sides. Also since the conductance of the cut is $O(1/\tau)$, the total weight of the inter-cluster edges is at most a $O(1/\tau)$ fraction of the weighted volume of the smaller side, so even with two copies of each cut edge on both side of a cut, we can charge $O(1/\tau)$ to each unit of weight on the smaller side of the cut. Since a unit of weight can be on the smaller side of a cut at most $O(\ln m_G)$ times before we get to a trivial cluster, the total additional weight from adding the cut boundary edges over all recursive calls starting from the whole graph is at most $O(\frac{\mathbf{vol}^W(G)\ln m_G}{\tau}) \leq 0.001\mathbf{vol}^W(G)$ as we have a large enough $\tau = \Omega(\ln m_G)$, so for simplicity we can safely ignore the impact of the additional edges to the total asymptotic running time.

5.3 The Certification Case In the previous section we showed that if the algorithm terminated in the second stopping condition (Check 2), then the algorithm finds a relatively balanced low conductance cut. Next, we look at the case when the algorithm terminated in Check 1. In this section on we will explicitly multiply values in stage j by the unit μ_j so all quantities are in unit 1.

We now consider only the flow problems that successfully finishes all L stages. For such a flow problem associated with source node v, we call v together with the edges on which its supply was transported an *edge-bundle* with *center v*. As the algorithm terminated in Check 1, we find 5000τ batches, each with $\frac{W}{10s}$ edge-bundles. That is, each edge-bundle routes at least $1.8s$ (i.e. $0.9 \cdot 10^7\tilde{\delta}\tau \cdot \mu_0$) supply to sinks in stage L. We start with the following definition.

Definition 5.2. *For an edge-bundle with source node v in a s-strong component C, we call the edge-bundle s-*captured *if in stage 0, the flow problem from v is successful and at most $0.5s$ of its supply is routed to nodes in $C \setminus S$ for any cut S in G with $\mathbf{vol}^W(S \cap C) \le \mathbf{vol}^W(C \setminus S)$ and cut-size $w(S, G \setminus S) \le 1.1\tilde{\delta}$. We call an edge-bundle s-free if it's not s-captured.*

Note in the above definition, since C is s-strong w.r.t the same $\tilde{\delta}$ (Definition 2.1), we know all the cut S being considered must have $\mathbf{vol}^W(S \cap C) \le s$. We start with the following result, which implies that in the case where we successfully get 5000τ bathes each with $\frac{W}{10s}$ edge-bundles, all these edge-bundles are s-free since at least $1.8s > 1.6s$ supply is routed to sinks in stage L for each edge-bundle.

Lemma 5.5. *The flow problem corresponding to any s-captured edge-bundle cannot have a total of more than $1.6s$ supply routed to sinks in stage L.*

Proof. Consider any s-captured edge-bundle and a corresponding cut S. By definition at most $0.5s$ supply is routed to $C \setminus S$ in stage 0. The remaining at least $1.5s$ supply is inside $C \cap S$ after stage 0, and we can look at how much can be routed into $C \setminus S$ in later stages.

By our choice of edge capacity for edge e being $U_j\mu_j w_e = 100\tau\frac{2s}{2^j 10^7\tilde{\delta}\tau}w_e$ in stage $j \ge 1$, we know that over all later stages, the total edge capacity is at most $\frac{2sw_e}{10^5\tilde{\delta}}$ for edge e. If we sum over all edges across the cut $C \cap S$ and $C \setminus S$, the total capacity is at most $\frac{2s\cdot 1.1\tilde{\delta}}{10^5\tilde{\delta}}$, so we can route at most $0.1s$ supply into $C \setminus S$.

In total this means at most $0.6s$ supply can be routed to nodes (and thus absorbed by sinks) in $C \setminus S$ over all L stages. The nodes in $C \cap S$ can absorb at most s supply since $\mathbf{vol}^W(C \cap S) \le s$, so in total at most $1.6s$ supply can be absorbed by sinks. ◻

Lemma 5.6. *There are at most 50τ edge-bundles that are s-free but with centers inside a cut S such that $\mathbf{vol}^W(S \cap C) \le \mathbf{vol}^W(C \setminus S)$ and $w(S \cap C, C \setminus S) \le 1.1\tilde{\delta}$.*

Proof. For an edge-bundle with center inside $C \cap S$ to be s-free, the flow problem must send at least $0.5s$ supply from $v \in C \cap S$ to nodes in $C \setminus S$ in stage 0. The total edge capacity across $C \cap S$ to $C \setminus S$ in stage 0 is at most $U_0\mu_0 \cdot 1.1\tilde{\delta} = 10^8\tau^2\frac{2s}{10^7\tilde{\delta}\tau}1.1\tilde{\delta} \le 22s\tau$, so we can have at most $\frac{22s\tau}{0.5s} \le 50\tau$ such edge-bundles. ◻

We construct a new flow problem Π_F based on all the successful stage L flows we get as follows. There are in total $5000\tau \cdot \frac{W_C}{10s}$ such flow problems, and in the stage-L pre-flow result for each of them, we have at least $1.8s$ supply absorbed in sinks. We aggregate the absorbed supply over all the $5000\tau \cdot \frac{W_C}{10s}$ pre-flows, and scale the amount down by 500τ, and we will use the result amount of supply on each node as the source supply function $\tilde{\Delta}$ for our new flow problem Π_F. The sink capacity of any node v in our new flow problem will again be $d^W(v)$.

Observation 5.1. *The total amount of source supply in Π_F is more than the total sink capacity (i.e. W_C), since $|\tilde{\Delta}(\cdot)| \ge \frac{1.8s}{500\tau}\frac{5000\tau W_C}{10s} = 1.8W_C > W_C$. Moreover, $\tilde{\Delta}(v) \le \frac{5000\tau\cdot d^W(v)}{500\tau} = 10d^W(v)$, since pre-flows in the same batch share the sink capacity $d^W(v)$.*

Note that any source supply in $\tilde{\Delta}$ on some node v is a result of our previous flow problems, that is, the supply on v must be from either the initial source supply we put on v in stage 0, or being routed to v by the pre-flows over the stages $0, \ldots L$. Consider any cut S being considered in the definition of s-captured, we first bound the amount of source flow in $\tilde{\Delta}$ on nodes in $S \cap C$.

Lemma 5.7. *For any cut S such that $w(S, G\setminus S) \le 1.1\tilde{\delta}$ and $\mathbf{vol}^W(S\cap C) \le \mathbf{vol}^W(C\setminus S)$, we have $\tilde{\Delta}(C\cap S) \le s/4$*

Proof. We first bound the amount of supply inside S from the initial source supply we put on nodes in S in stage 0. By Lemma 5.5 and Lemma 5.6, at most 50τ nodes in S are centers of the edge-bundles we include, so the total amount of supply is at most $50\tau \cdot 2s = 100\tau s$.

Now we bound the amount of supply that can be routed in to S by the pre-flows. For stage 0, since all flow problems share a single copy of the graph, the total supply can be routed into S over all of them is at most the total capacity into S, which is

$$U_0 \cdot w(S \cap C, C \setminus S) \cdot \mu_0 \leq 10^8\tau^2 \cdot 1.1\tilde{\delta} \cdot \frac{2s}{10^7\tilde{\delta}\tau} \leq 22\tau s.$$

For stages $1, \ldots, L$, the different batches don't share edge capacities, so the total capacity into S is

$$\begin{aligned} 5000\tau \cdot w(S \cap C, C \setminus S) \cdot \sum_{j=1}^{L} U_j\mu_j \leq & 5000\tau \cdot 1.1\tilde{\delta} \cdot 2U_1\mu_1 \\ = & 5000\tau \cdot 1.1\tilde{\delta} \cdot 100\tau \cdot \frac{2s}{10^7\tilde{\delta}\tau} \leq \tau s. \end{aligned}$$

Since we scale the supply down by 500τ to get $\tilde{\Delta}$, we know

$$\tilde{\Delta}(S \cap C) \leq \frac{100\tau s + 22\tau s + \tau s}{500\tau} \leq \frac{s}{4}.$$

□

For the flow problem Π_F, we will again use $U = 100\tau$ and call Unit-Flow with $h = \Theta(\tau \ln m_G)$, and we have the following result.

LEMMA 5.8. *We can either certify C is $s/2$-strong, or find a cut $A \subsetneq C$ such that $\Phi_C(A) = \frac{w(A, C\setminus A)}{\min(\mathbf{vol}^W(A), \mathbf{vol}^W(C\setminus A))} \leq \tau$, $\mathbf{vol}^W(A) = \Theta(W_C)$ and A is $s/2$-strong.*

Proof. Consider any cut S as in Lemma 5.7, we first show that the set $\tilde{C}$ of nodes whose sinks are saturated in the Unit-Flow result of Π_F must have $\mathbf{vol}^W(\tilde{C} \cap S) \leq s/2$. This is because at most $s/4$ supply starts in S by Lemma 5.7, and in the pre-flow for Π_F, the additional supply that can be routed into S is bounded by the total edge capacity into S, which by our choice of U is at most $100\tau \cdot 1.1\tilde{\delta} \leq s_0/\tau < s/\tau$, and thus the total supply in S when Unit-Flow finishes is at most $s/2$, and thus $\mathbf{vol}^W(\tilde{C} \cap S) \leq s/2$. Note this certified that $\tilde{C}$ (and thus any subset of it) is $s/2$-strong.

By Observation 5.1, we know there must be excess in the pre-flow from Unit-Flow, so we cannot get case (1) in Theorem 4.1. If we get case (2), then we know $C = \tilde{C}$ and by the above argument $\mathbf{vol}^W(C \cap S) \leq s/2$ for all such S, which certifies C is $s/2$-strong. If we get case (3), let $(A, C \setminus A)$ be the returned cut. We know all nodes in A are saturated, i.e. $A \subset \tilde{C}$, so A is $s/2$-strong. The claim on conductance $\Phi(A)$ and volume $\mathbf{vol}^W(A)$ follow (3)(b) and (3)(c) in Theorem 4.1 respectively by our choice of $U = 100\tau, h = \Theta(\tau \ln W_C)$, and note by Observation 5.1 $\beta \leq 10, \mathrm{ex}(T) \geq 0.8W_C$. □

We conclude this section with the proof of the main lemma of s_0-strong partition.

LEMMA 5.1. (s_0-STRONG PARTITION) *Given a weighted graph $G = (V, E, w)$, a parameter $\tilde{\delta}$ such that $\tilde{\delta} \leq \min_{v \in V} d^W(v)$ and a parameter $s_0 = \Theta(\tilde{\delta} \log^c |V|)$ for some $c \geq 2$, there is an algorithm that runs in $\tilde{O}(|E|)$ time and partitions the vertex set V into s_0-strong components for $\tilde{\delta}$ such that the total weight of the edges between distinct components is at most $O(\frac{\sqrt{\tilde{\delta}}\log|V|}{\sqrt{s_0}}) \cdot \mathbf{vol}^W(V)$.*

Proof. Our procedure starts with the graph G, and recursively running the certification procedure until all components are certified to be s_0-strong. Note a component with weighted volume at most s_0 is by definition s_0-strong. Thus, if for any component C we always divide it along a relatively balanced low conductance cut whose smaller side has weighted volume at least $\Omega(\mathbf{vol}^W(C)/\operatorname{polylog}\mathbf{vol}^W(G))$, then the depth of our recursion is at most $O(\operatorname{polylog}\mathbf{vol}^W(G))$ until all components have weighted volume at most s_0.

In our certification procedure, the only case when we don't divide C along a relatively balanced cut is in the case of Section 5.3 where we get a cut $(A, C \setminus A)$ as in Lemma 5.8. Since $\mathbf{vol}^W(A) = \Theta(\mathbf{vol}^W(C))$, if it is not relatively balanced, we know A must be the larger side, and we certify it's $s/2$-strong. Thus, when we cannot find a relatively balanced cut, we make progress by certifying a smaller strength on the larger side of the cut (or possibly the entire component C). This can happen at most $\ln \frac{\mathbf{vol}^W(G)}{s_0}$ times before we are done with the component. Consequently, when we look at both cases (i.e. cut or certify), the depth of the recursion is at most $O(\operatorname{polylog} \mathbf{vol}^W(G))$ since in each level we either decrease the weighted volume by a factor of $1 - \Omega(1/\operatorname{polylog} \mathbf{vol}^W(G))$ or certify a strength of $1/2$ the current certified strength for the larger side of the unbalanced cut.

The running time of our certification procedure for a component C is $\tilde{O}(m_C)$ if C has m_C edges. This is because we run one Unit-Flow computation for the aggregate flow problem in each stage and each batch, so $L \cdot 5000\tau = O(\tau \ln m_C)$ of them in total (plus the last flow problem Π_F if needed). Each of the Unit-Flow computation takes time $\tilde{O}(m_C \cdot h)$ by Theorem 4.1, which is at most $\tilde{O}(m_C \tau^2)$ by our choice of h, i.e. $h = \tau^2 \ln m_G$ for stage 0, and $h = \tau \ln m_G$ for all later stages and Π_F. In total, the running time of all the Unit-Flow calls take $\tilde{O}(m_C \tau^3)$ in the certification procedure of C. By our earlier discussion, the recursion depth of the partition procedure is at most $O(\operatorname{polylog} \mathbf{vol}^W(G))$, the total running time for all the Unit-Flow calls over all components is $\tilde{O}(m_G \tau^3)$.

As to the total weight of edges between distinct components, since we only divide a component along cut of conductance at most $O(1/\tau)$, so we can charge the weight of cut edges to the weighted volume of the smaller side of the cut. Since one unit of volume can be on the smaller side of a cut at most $O(\ln \mathbf{vol}^W(G))$ times before we stop, each unit of weight is charged at most $O(\frac{\ln m_G}{\tau})$, so the total weight of edges between components is $O(\frac{\mathbf{vol}^W(V) \ln |V|}{\tau})$ (assuming polynomially bounded edge weights). Given our choice of $\tau = \Theta(\sqrt{s_0/\tilde{\delta}}) = \Theta(\ln^{c/2} |V|)$, the running time is $\tilde{O}(m_G \tau^3) = \tilde{O}(m_G)$ as in the lemma statement. The weight of edges between components is $O(\frac{\ln |V|}{\tau}) \cdot \mathbf{vol}^W(G)$, which is $O(\frac{\sqrt{\tilde{\delta}} \ln |V|}{\sqrt{s_0}}) \cdot \mathbf{vol}^W(G)$ as claimed in the lemma. $\square$

6 From s_0-strength to efficient uncrossing.

Recall that an s_0-strong decomposition, $\mathcal{D}$, of a graph G is a decomposition of V in which any min-cut, S, has at most volume s_0 in common with any cluster $A \in \mathcal{D}$ in the decomposition, i.e., the volume of $A \cap S$ is at most s_0.

In this section, we convert such a decomposition into a decomposition, $\mathcal{D}'$ such that for any min-cut S and cluster $A \in \mathcal{D}'$ an uncrossing of the cut S with A does not locally increase the cut-size "too much". Recall the definition of boundary-sparsity given in Definition 2.2.

Intuitively, for a cut S that crosses A, if $S \cap A$ is not β-boundary-sparse in A, then uncrossing S with respect to A only increases the cut size by a small amount proportional to $1/\beta$. In other words, we want to find β-boundary-sparse cuts in A since these are the cuts that do not allow for cheap uncrossing. Ideally, in our final decomposition, no cluster would contain any β-boundary-sparse sets. In reality, we will only be able to achieve a slightly relaxed version of this statement; see Lemma 6.1 below.

We will proceed by considering each cluster in the original decomposition and further decomposing it as necessary. In Section 6.1, we give a procedure that works for clusters that have volume $\tilde{O}(s_0)$ and, thus, consist of $\tilde{O}(1)$ vertices. We call such clusters *small* clusters. In Section 6.2 we give a procedure for *large* clusters, i.e. clusters that are not small.

6.1 Efficient uncrossing for small clusters. Recall that a small cluster contains $\tilde{O}(1)$ vertices. Thus a running time that is polynomial in the number of vertices of A is sufficient for our purpose. Throughout this section we assume that $\tilde{\lambda} \leq 1.01\lambda$ is an approximate lower bound of the minimum cut value that is available to our algorithm.

LEMMA 6.1. (SMALL CLUSTER DECOMPOSITION) *Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.01$ be a parameter, and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a disjoint union of clusters $A_1 \cup A_2 \cup \cdots \cup A_k = A$ such that*

1. *For any set $S \subseteq A$ of G with $\partial_{G[A]} S \leq 1.01\tilde{\lambda}$, there is a partition $\mathcal{P}$ of the set $\{A_1, A_2, \ldots, A_k\}$ such that for each part $P = \{A_{i_1}, \ldots, A_{i_\ell}\} \in \mathcal{P}$, the set $S \cap \bigcup_{A' \in P} A'$ is non-$(1-\epsilon)$-boundary-sparse in $\bigcup_{A' \in P} A'$.*

2. *The sum of boundaries $\sum_i \partial A_i$ is at most $O(\epsilon^{-3}(\log |A|)^{O(1)}\partial A)$.*

The running time of the algorithm is $O(|A|^9 \operatorname{polylog}|A|)$.

To prove Lemma 6.1, we will need the following helper lemma, whose proof is standard.

Theorem 6.1. *Given a weighted graph H and a parameter $\alpha \geq 1$, there is a deterministic algorithm that enumerates all α-approximate minimum cuts of H, of which there are $O(n^{\lfloor 2\alpha \rfloor})$. The algorithm runs in time $\tilde{O}(m^2 n^{\lfloor 2\alpha \rfloor}) \leq \tilde{O}(n^{\lfloor 2\alpha \rfloor + 4})$.*

Proof. First, compute a maximum (spanning) tree packing of H, which can be done in $\tilde{O}(mn)$ time [Bar95] to produce a packing of $\tilde{O}(m)$ trees. It was shown in [Kar00]) that for any α-approximate minimum cut S, there is a tree T in a maximum tree packing for which $\partial_T S \leq 2\alpha$, i.e., at most 2α many edges of T cross the cut S. The algorithm iterates over all $\tilde{O}(m)$ trees in the packing, and for each tree T, considers all ways to remove at most 2α edges, group the components together into two sides of a cut, and determine the size of the cut. For constant α, this takes $O(mn^{\lfloor 2\alpha \rfloor})$ time per tree, for a total of $\tilde{O}(m^2 n^{\lfloor 2\alpha \rfloor})$ time. The algorithm then computes the minimum cut λ over this collection, which is the minimum cut of G, and outputs the subcollection of cuts of value at most $\alpha\lambda$. The bound of $O(n^{\lfloor 2\alpha \rfloor})$ for the number of such cuts was shown in [Kar00]. ◻

We now prove Lemma 6.1. The algorithm for Lemma 6.1 has two stages. The first stage, described in the next lemma, decomposes A into clusters such that (a) each has boundary size $O(\tilde{\lambda})$ and minimum cut in the induced graph of at least $0.49\tilde{\lambda}$ and (b) the size of the sum of their boundaries is small, namely $O(\epsilon^{-1}\partial A)$. These clusters are further broken down in the next stage given in Lemma 6.3. To shorten the notation given a cluster A and a set $U \subseteq A$, we use ∂U to denote $w(U, \bar{U})$ and $\partial_{G[A]} U$ to denote $w(U, A \setminus U)$.

Lemma 6.2. *Let $A \subseteq V$ be a cluster. Suppose an algorithm is given a mincut estimate $\tilde{\lambda} \in [\lambda, 1.01\lambda]$. Then, the algorithm can decompose A into a disjoint union of clusters $A_1 \cup A_2 \cup \cdots \cup A_k = A$ such that*

1. *For each cluster A_i, either $\partial A_i \leq 3\tilde{\lambda}$ or there is no $(1-\epsilon)$-boundary-sparse set U in A_i with $\partial_{G[A_i]} U \leq 1.01\tilde{\lambda}$.*
2. *The sum of boundaries $\sum_i \partial A_i$ is $O(\epsilon^{-1}\partial A)$.*
3. *For each cluster A_i, the induced graph $G[A_i]$ has minimum cut at least $0.49\tilde{\lambda}$.*

The algorithm takes time $\tilde{O}(|A|^9)$.

Proof. We first describe an algorithm that fulfills the first two properties. At the end, we augment it to satisfy property (3).

Property 1. Given a cluster A the algorithm maintains a current decomposition $\mathcal{D}$, initialized to the singleton set $\{A\}$. The algorithm iteratively considers sets in $\mathcal{D}$ that have not been processed yet until all have been processed, starting with A. For a given set A', the algorithm does the following.

First, if $\partial A' \leq 3\tilde{\lambda}$, then the algorithm keeps A' as is, declares A' as processed, and proceeds to the next set in $\mathcal{D}$. Otherwise, it applies the algorithm of Theorem 6.1 to the induced graph $G[A']$ for $\alpha = 2.1$, which takes $\tilde{O}(|A'|^8)$ time, and let $\mathcal{C}$ be the set of returned cuts. Compute the cut $U^* \in \mathcal{C}$ such that $\partial_{G[A']} U^*$ is the smallest cut over all cuts in $\mathcal{C}$. By the choice of $\mathcal{C}$, U^* is the minimum cut in $G[A']$. We now have a few cases.

(a) If $\partial_{G[A']} U^* \leq 0.49\tilde{\lambda}$, then the algorithm decomposes A' into U^* and $A' \setminus U^*$, i.e., it removes A' from $\mathcal{D}$ and adds the unprocessed sets U^* and $A' \setminus U^*$ to $\mathcal{D}$. (The sets U^* and $A' \setminus U^*$ are processed at a later iteration.)

(b) If $\partial_{G[A']} U^* > 0.49\tilde{\lambda}$, then it follows from our choice of $\alpha = 2.1$ that the collection $\mathcal{C}$ contains all sets $U \subseteq A'$ with $\partial_{G[A']} U \leq 1.01\tilde{\lambda}$. The algorithm iterates over all sets in $\mathcal{C}$ and looks for one that is $(1-\epsilon)$-boundary-sparse and has boundary size $\leq 1.01\tilde{\lambda}$. If a $(1-\epsilon)$-boundary-sparse set with boundary size $\leq 1.01\tilde{\lambda}$ does not exist, then the algorithm keeps A' as is, declares A' as processed, and proceeds to the next set in $\mathcal{D}$. Otherwise, let U be an arbitrary $(1-\epsilon)$-boundary-sparse set with boundary size $\leq 1.01\tilde{\lambda}$. The algorithm decomposes A' into U and $A' \setminus U$, i.e., removes A' from $\mathcal{D}$ and adds the unprocessed sets U and $A' \setminus U$ into $\mathcal{D}$. (The sets U and $A' \setminus U$ are processed at a later iteration.)

All sets in the final decomposition satisfy property (1) by construction. As each iteration of the algorithm processes one set, each set is processed only once, and the collection of processed sets forms a laminar family, there are at most $|A|$ many iterations. As each iteration takes time $\tilde{O}(|A|^8)$, the total running time is $\tilde{O}(|A|^9)$.

Property 2. Let $\mathcal{D}_1$ be the decomposition of A at the end of the algorithm. If $\partial A \leq 3\tilde{\lambda}$ then the algorithm does nothing and property (2) is trivially true. Otherwise, assume that the algorithm decomposes A. Throughout the algorithm, we maintain a potential function

$$\Phi(\mathcal{D}) = \sum_{A' \in \mathcal{D}} \max\{0, \partial A' - 2.1\tilde{\lambda}\}.$$

The initial value of $\Phi(\mathcal{D}) = \Phi(\{A\}) = O(\partial A)$. We will show (1) that $\Phi(\mathcal{D})$ does not increase throughout the decomposition, which implies that $\Phi(\mathcal{D}_1) \leq \Phi(\{A\})$. Moreover, we show that (2) each time a set A' with $\partial A' \leq 2.2\tilde{\lambda}$ is added to $\mathcal{D}$, the potential $\Phi(\mathcal{D})$ drops by at least $\Omega(\epsilon\tilde{\lambda})$. These two claims together imply property (2) because

$$\begin{aligned}\sum_{A' \in \mathcal{D}_1} \partial A' &= \sum_{A' \in \mathcal{D}_1 : \partial A' \leq 2.2\tilde{\lambda}} \partial A' + \sum_{A' \in \mathcal{D}_1 : \partial A' > 2.2\tilde{\lambda}} \partial A' \\ &\leq \frac{\Phi(\{A\})}{\Omega(\epsilon\tilde{\lambda})} \cdot 0.1\tilde{\lambda} + O(\Phi(\mathcal{D}_1)) \leq O(\Phi(\{A\})/\epsilon) \leq O(\partial A/\epsilon).\end{aligned}$$

To show the two claims, suppose a set A' is decomposed into U and $A' \setminus U$ for some $U \subseteq A'$. By construction, we always have $\partial_{G[A']} U \leq 1.01\tilde{\lambda}$. If ∂U and $\partial(A' \setminus U)$ are both at least $2.1\tilde{\lambda}$, then the net increase in $\Phi(\mathcal{D})$ is

$$\begin{aligned}&\max\{0, \partial U - 2.1\tilde{\lambda}\} + \max\{0, \partial(A' \setminus U) - 2.1\tilde{\lambda}\} - \max\{0, \partial A' - 2.1\tilde{\lambda}\} \\ &= (\partial U - 2.1\tilde{\lambda}) + (\partial(A' \setminus U) - 2.1\tilde{\lambda}) - (\partial A' - 2.1\tilde{\lambda}) \\ &= 2w(U, A' \setminus U) - 2.1\tilde{\lambda} \leq 2 \cdot 1.01\tilde{\lambda} - 2.1\tilde{\lambda} < -0.08\tilde{\lambda}.\end{aligned}$$

If ∂U and $\partial(A' \setminus U)$ are both at most $2.1\tilde{\lambda}$, then the net increase in $\Phi(\mathcal{D})$ is

$$\begin{aligned}&\max\{0, \partial U - 2.1\tilde{\lambda}\} + \max\{0, \partial(A' \setminus U) - 2.1\tilde{\lambda}\} - \max\{0, \partial A' - 2.1\tilde{\lambda}\} \\ &\leq 0 + 0 - (\partial A' - 2.1\tilde{\lambda}) \leq 0 + 0 - (3\tilde{\lambda} - 2.1\tilde{\lambda}) \leq -0.9\tilde{\lambda}.\end{aligned}$$

Else, suppose without loss of generality that $\partial U \leq 2.1\tilde{\lambda} \leq \partial(A' \setminus U)$. The net increase in $\Phi(\mathcal{D})$ is

$$\begin{aligned}&\max\{0, \partial U - 2.1\tilde{\lambda}\} + \max\{0, \partial(A' \setminus U) - 2.1\tilde{\lambda}\} - \max\{0, \partial A' - 2.1\tilde{\lambda}\} \\ &= 0 + (\partial(A' \setminus U) - 2.1\tilde{\lambda}) - (\partial A' - 2.1\tilde{\lambda}) \\ &= \partial(A' \setminus U) - \partial A' \\ &= w(U, A' \setminus U) - w(U, \overline{A'}).\end{aligned}$$

If $\partial_{G[A']} U \leq 0.49\tilde{\lambda}$, i.e., the algorithm went through case (a), then

$$w(U, \overline{A'}) = \partial U - w(U, A' \setminus U) \geq \lambda - 0.49\tilde{\lambda} \geq 0.99\tilde{\lambda} - 0.49\tilde{\lambda} = 0.5\tilde{\lambda},$$

and the net increase in $\Phi(\mathcal{D})$ is

$$w(U, A' \setminus U) - w(U, \overline{A'}) \leq 0.49\tilde{\lambda} - 0.5\tilde{\lambda} < -0.01\tilde{\lambda}.$$

If $\partial_{G[A']} U > 0.49\tilde{\lambda}$, i.e., case (b), then U is $(1-\epsilon)$-boundary-sparse in A' by construction, so the net increase in $\Phi(\mathcal{D})$ is

$$w(U, A' \setminus U) - w(U, \overline{A'}) \leq w(U, A' \setminus U) - \frac{w(U, A' \setminus U)}{1 - \epsilon} \leq -\Omega(\epsilon w(U, A' \setminus U)) \leq -\Omega(\epsilon\tilde{\lambda}).$$

This concludes the proof that the algorithm that fulfills the properties (1) and (2).

Property 3. To satisfy property (3), we take the decomposition $\mathcal{D}_1$ after termination of the algorithm above and, while there exists a cluster A' in the current decomposition whose induced graph $G[A']$ has minimum cut $0.49\tilde{\lambda}$ or less, we find the minimum cut $U^* \subseteq A'$ and decompose A' into U^* and $A' \setminus U^*$. Computing the minimum cut can be done by applying Theorem 6.1 for $\alpha = 1$, which takes $\tilde{O}(|A'|^6)$ time.

We claim that this additional step increases $\sum_{A' \in \mathcal{D}_1} \partial A'$ by only a constant factor. Let $\mathcal{D}$ be the decomposition during this process and let $\mathcal{D}_2$ be the decomposition at the end of this process. Consider the potential function

$$\Phi'(\mathcal{D}) = \sum_{A' \in \mathcal{D}} (\partial A' - 0.98\tilde{\lambda}).$$

Note that $\Phi'(\mathcal{D}_1) \le \sum_{A' \in \mathcal{D}_1} \partial A'$. Since $\partial A' \ge \lambda \ge \frac{1}{1.01}\tilde{\lambda} \ge 0.99\tilde{\lambda}$, we have $\partial A' \le 99(\partial A' - 0.98\tilde{\lambda})$ and $\sum_{A' \in \mathcal{D}} \partial A' \le O(\Phi'(\mathcal{D}))$. In particular, $\sum_{A' \in \mathcal{D}_2} \partial A' \le O(\Phi'(\mathcal{D}_2))$. We will show that $\Phi'(\mathcal{D})$ does not increase over time. This shows the claim as it implies that $\sum_{A' \in \mathcal{D}_2} \partial A' \le O(\Phi'(\mathcal{D}_2)) \le O(\Phi'(\mathcal{D}_1)) = O(\sum_{A' \in \mathcal{D}_1} \partial A')$. To show that Φ' does not increase upon decomposing some $A' \in \mathcal{D}$ into U and $A' \setminus U$, the net increase of $\Phi'(\mathcal{D})$ is

$$(\partial U - 0.98\tilde{\lambda}) + (\partial(A' \setminus U) - 0.98\tilde{\lambda}) - (\partial A' - 0.98\tilde{\lambda}) = 2w(U, A' \setminus U) - 0.98\tilde{\lambda} \le 2 \cdot 0.49\tilde{\lambda} - 0.98\tilde{\lambda} \le 0,$$

proving the claim. This concludes the proof. ◻

Using Lemma 6.2 as a first step, we now apply the following lemma to each cluster with boundary at most $3\tilde{\lambda}$ output by Lemma 6.2.

Lemma 6.3. *Let $A \subseteq V$ be a cluster satisfying the following two guarantees:*

- *$\partial A \le 3\tilde{\lambda}$, and*
- *the minimum cut of $G[A]$ is at least $0.49\tilde{\lambda}$.*

Let $0 < \epsilon \le 0.01$ be a parameter. Suppose an algorithm is given a mincut estimate $\tilde{\lambda} \in [\lambda, 1.01\lambda]$. Then, the algorithm can decompose A into a disjoint union of clusters $A_1 \cup A_2 \cup \cdots \cup A_k = A$ such that

1. *For any set $S \subseteq A$ with $\partial_{G[A]} S \le 1.01\tilde{\lambda}$, there is a partition $\mathcal{P}$ of $\{A_1, A_2, \ldots, A_k\}$ such that for each part $P \in \mathcal{P}$, the set $S \cap \bigcup_{A' \in P} A'$ is non-$(1-\epsilon)$-boundary-sparse in $\bigcup_{A' \in P} A'$.*
2. *There are at most $O(\epsilon^{-2}(\log|A|)^{O(1)})$ many clusters, and each cluster A_i satisfies $\partial A_i \le \partial A$.*

The algorithm runs in time $\tilde{O}(|A|^8)$.

Proof. Apply Theorem 6.1 for $\alpha = 2.1$ to obtain, in time $\tilde{O}(|A|^8)$, a collection $\mathcal{C}_0$ of $\tilde{O}(|A|^4)$ sets that contains all $\emptyset \subsetneq U \subsetneq A$ with $\partial_{G[A]} U \le 1.01\tilde{\lambda}$. Take the subcollection $\mathcal{C} \subseteq \mathcal{C}_0$ of all $U \in \mathcal{C}$ with $\partial_{G[A]} U \le 1.01\tilde{\lambda}$. The rest of the algorithm recursively decomposes an input cluster A' (starting with $A' = A$) while keeping a set $\mathcal{C}' \subseteq \mathcal{C}$ that starts as $\mathcal{C}' = \mathcal{C}$ and may lose elements over time.

Initially, the algorithm removes from $\mathcal{C}'$ all cuts $C \in \mathcal{C}'$ such that $C \cap A'$ is not $(1-\epsilon)$-boundary sparse in A'. We call this the initial *pruning* step. The recursive algorithm then considers the following cases in order:

(a) If there is $C \in \mathcal{C}'$ that crosses A' such that $\partial_{G[A']}(C \cap A') \le 0.4\tilde{\lambda}$, decompose A' into $C \cap A'$ and $A' \setminus C$ and recursively call $C \cap A'$ and $A' \setminus C$ with the updated $\mathcal{C}'$.

(b) Otherwise, if there exists $C \in \mathcal{C}'$ that crosses A' , take a set $C \in \mathcal{C}'$ with minimum $|C \cap A'|$ crossing A'. Decompose A' into $C \cap A'$ and $A' \setminus C$ and recursively call $C \cap A'$ and $A' \setminus C$ with the updated $\mathcal{C}'$.

(c) Otherwise, return the trivial decomposition A'.

Running time analysis. In the initial pruning step, the algorithm can check each of the $\tilde{O}(|A|^4)$ sets in $\mathcal{C}'$ in $O(|A|^2)$ time. As the set of processed sets forms a laminar family, there are $O(|A|)$ many recursive instances, so the recursive algorithm takes time $\tilde{O}(|A|^7)$. Together with the initial $\tilde{O}(|A|^8)$ time to compute $\mathcal{C}'$, this establishes the running time.

Correctness: Property (1). We first show that the resulting decomposition satisfies property (1) of the lemma. Consider a set $S \subseteq A$ with $\partial_{G[A]} S \le 1.01\tilde{\lambda}$. If $S = \emptyset$ or $S = A$, then property (1) is trivial because $\emptyset$ and A are always non-$(1-\epsilon)$-boundary sparse in A.

Thus we are left with proving property (1) if S is a cut in A. Since $\partial_{G[A]} S \le 1.01\tilde{\lambda}$, we have $S \cap A \in \mathcal{C}$, so initially $S \cap A \in \mathcal{C}'$ on the root instance A. We now prove property (1) by induction on the recursion tree of the algorithm. As base cases, we consider instances A' where S is removed from $\mathcal{C}'$ in the initial pruning step. Since S is removed from $\mathcal{C}'$, it is non-$(1-\epsilon)$-boundary sparse in A', i.e., property (1) holds for the trivial partition with the entire decomposition of A' as a single part. For instances A' where $S \in \mathcal{C}'$ survives the initial pruning step, we apply induction on the two recursive instances $C \cap A'$ and $A' \setminus C$ and take the union of the two partitions, which is a partition of the decomposition returned by instance A'. It follows that property (1) holds.

Correctness: Property (2). To show property (2), we first make two observations.

OBSERVATION 6.1. *If the algorithm decomposes by case (a) into $U = C \cap A'$ and $A' \setminus C = A' \setminus U$, then $\partial U \le \partial A' - 0.19\tilde{\lambda}$ and $\partial(A' \setminus U) \le \partial A' - 0.19\tilde{\lambda}$.*

Proof. Suppose the algorithm decomposes into $U = C \cap A'$ and $A' \setminus C = A' \setminus U$. First observe that if the algorithm decomposes by case (a), then

$$\begin{aligned} w(U, \overline{A'}) &= \partial U - w(U, A' \setminus U) \ge \lambda - 0.4\tilde{\lambda} \ge 0.99\tilde{\lambda} - 0.4\tilde{\lambda} = 0.59\tilde{\lambda} \\ \implies \partial(A' \setminus U) &= \partial A' - w(U, \overline{A'}) + w(U, A' \setminus U) \le \partial A' - 0.59\tilde{\lambda} + 0.4\tilde{\lambda} = \partial A' - 0.19\tilde{\lambda} \end{aligned}$$

and by symmetry, we also obtain $\partial U \le \partial A' - 0.19\tilde{\lambda}$. In other words, boundary size decreases by at least 0.19λ on both recursive instances. ◻

OBSERVATION 6.2. *If the algorithm decomposes by case (b) into $U = C \cap A'$ and $A' \setminus C = A' \setminus U$, then $\partial U \le \partial A' - 0.4\epsilon\tilde{\lambda}$ and $\partial(A' \setminus U) \le \partial A' - 0.4\epsilon\tilde{\lambda}$.*

Proof. If the algorithm decomposes by case (b), then since U is $(1-\epsilon)$-boundary-sparse in A',

$$\begin{aligned} \partial(A' \setminus U) &= \partial A' - w(U, \overline{A'}) + w(U, A' \setminus U) \\ &\le \partial A' - \frac{1}{1-\epsilon} w(U, A' \setminus U) + w(U, A' \setminus U) \le \partial A' - \epsilon w(U, A' \setminus U) \le \partial A' - 0.4\epsilon\tilde{\lambda} \end{aligned}$$

and by symmetry, $\partial U \le \partial A' - 0.4\epsilon\tilde{\lambda}$. ◻

The two observations together show that boundary size decreases by at least $\min\{0.19\tilde{\lambda}, 0.4\epsilon\tilde{\lambda}\}$ on both recursive instances.

We now upper bound the number of clusters by tracking the following three parameters of a cluster A':

1. The boundary size $\partial A'$, and

2. The maximum cut size within A' of a cut in $\mathcal{C}'$ after the pruning step, defined by

$$c(A') = \max_{\substack{C \in \mathcal{C}' \\ \text{after} \\ \text{pruning}}} \partial_{G[A']}(C \cap A'),$$

   which we define as 0 if $\mathcal{C}' = \emptyset$, and

3. The cluster size $|A'|$.

We claim that the first two parameters are at most $3\tilde{\lambda}$ and $1.01\tilde{\lambda}$, respectively. The first parameter is always at most $3\tilde{\lambda}$ since originally $\partial A \le 3\tilde{\lambda}$ by assumption, and we have shown that boundary size never increases upon recursion. The second parameter is at most $1.01\tilde{\lambda}$ originally, and it never increases upon recursion since the value $\partial_{G[A']}(C \cap A')$ decreases as A' gets smaller upon recursion and $\mathcal{C}'$ can only lose elements over time.

Let $f(b,c,d)$ be the maximum number of clusters in the final decomposition starting from a cluster A' with $\partial A' \le b$, $c(A') \le c$, and $|A'| \le d$. Our goal is to show that

$$f(b,c,d) \le O\left((\frac{b}{\epsilon\tilde{\lambda}})^2 (\log d)^{O(b/\tilde{\lambda})} \right).$$

We recursively bound $f(b,c,d)$ by stepping through each case of the algorithm. To do so we first make two observations.

Observation 6.3. *If the algorithm decomposes by case (a) into $U = C \cap A'$ and $A' \setminus C = A' \setminus U$, then the maximum number of clusters in the final decomposition of each recursive instance $U, A' \setminus U$ is upper bounded by $f(b - 0.19\tilde{\lambda}, c, d)$, i.e., we have $f(b,c,d) = 2f(b - 0.19\tilde{\lambda}, c, d)$.*

Proof. This observation follows directly from Observation 6.1 as it shows that $\partial U, \partial(A' \setminus U) \le \partial A' - 0.19\tilde{\lambda} \le b - 0.19\tilde{\lambda}$. ∎

Observation 6.4. *If the algorithm decomposes by case (b), we must have $|C \cap A'| \le |A'|/2$. Moreover, all sets $C' \in \mathcal{C}'$ that cross $C \cap A'$ must also cross $A' \setminus C$.*

Proof. For the first statement, if $C \cap A'$ is $(1-\epsilon)$-boundary-sparse, then so is $A' \setminus C$, and since the algorithm minimizes for $|C \cap A'|$, we must have $|C \cap A'| \le |A' \setminus C|$, which means that $|C \cap A'| \le |A'|/2$.

For the second statement, suppose for contradiction that there exists $C' \in \mathcal{C}'$ that crosses $C \cap A'$ but not $A' \setminus C$. Without loss of generality, by replacing C' with $\overline{C'}$ if needed, we can assume that C' is disjoint from $A' \setminus C$. But then $|C' \cap A'| < |C \cap A'|$, and by minimality, C' should have been selected over C, a contradiction. ∎

Observation 6.5. *If the algorithm decomposes by case (b) we obtain the recursive bound*

$$f(b,c,d) \le \max\{1,\ 2f(b - 0.19\tilde{\lambda}, c, d) + f(b, c, d/2),\ f(b, 0.6c, d) + f(b - 0.4\epsilon\tilde{\lambda}, c, d)\}$$

Proof. Suppose that the algorithm decomposes by case (b) into $U = C \cap A'$ and $A' \setminus C = A' \setminus U$. By Observation 6.4, we have $|U| \le |A'|/2$ and moreover, all sets $C' \in \mathcal{C}'$ that cross U must also cross $A' \setminus U$. We case on whether $c(U) \ge 0.6c(A')$, i.e., whether there exists $C' \in \mathcal{C}'$ that crosses U with $\partial_{G[U]} C' \ge 0.6c(A')$.

1. If such a set C' exists, then from $\partial_{G[U]} C' + \partial_{G[A' \setminus U]} C' \le \partial_{G[A']} C'$ we obtain $\partial_{G[A' \setminus U]} C' \le \partial_{G[A']} C' - \partial_{G[U]} C' \le c(A') - 0.6c(A') = 0.4c(A')$. Since C' also crosses $A' \setminus U$, the recursive instance $A' \setminus U$ must go through case (a); let A_1, A_2 be the decomposition of $A' \setminus U$. Then by Observation 6.1, $\partial A_1, \partial A_2 \le \partial(A' \setminus U) - 0.19\tilde{\lambda} \le \partial A' - 0.19\tilde{\lambda}$. By induction on clusters U, A_1, A_2, the total number of clusters at the end is at most
$$2f(b - 0.19\tilde{\lambda}, c, d) + f(b, c, d/2).$$
2. Otherwise, we have $c(U) < 0.6c(A')$. By Observation 6.2, we have $\partial(A' \setminus U) \le \partial A' - 0.4\epsilon\tilde{\lambda}$. By induction, the total number of clusters at the end is at most
$$f(b, 0.6c, d) + f(b - 0.4\epsilon\tilde{\lambda}, c, d).$$

Overall, if the algorithm decomposes by case (b) we obtain the recursive bound

$$\begin{aligned} f(b,c,d) \le \max\{1,& \\ & 2f(b - 0.19\tilde{\lambda}, c, d) + f(b, c, d/2), \\ & f(b, 0.6c, d) + f(b - 0.4\epsilon\tilde{\lambda}, c, d)\}. \end{aligned}$$

∎

Next we state the base cases for our induction that bounds $f(b,c,d)$.

1. Since we always have $\partial A' \geq \lambda > 0.99\tilde{\lambda}$, we have the vacuous bound $f(b,c,d) = 0$ for $b \leq 0.99\tilde{\lambda}$.

2. If $\mathcal{C}' = \emptyset$, no further recursion is necessary. Thus, $f(b,0,d) = 1$.

3. If $|A'| = 1$, the cluster cannot be cut any further. Thus, $f(b,c,1) = 1$.

We now bound $f(b,c,d)$ using a case analysis depending on the value of the second parameter. As shown by Observation 6.3 and 6.5, the value of the second parameter never increases. Furthermore, if it drops to at most $0.4\tilde{\lambda}$ on a set A' then the algorithm always executes case (a) for all subsequent recursive calls on subsets of A'. Thus, we bound this setting first.

Case 1: If $c(A') \leq 0.4\tilde{\lambda}$, then the algorithm either takes case (a) or does nothing (in the case $c(A') = 0$), so

$$f(b,c,d) \leq \max\{1, 2f(b - 0.19\tilde{\lambda}, c, d)\} \text{ for } c \leq 0.4\tilde{\lambda}.$$

Every recursive case still has $c(A') \leq 0.4\tilde{\lambda}$, so the algorithm only makes recursive calls through case (a). The number of recursive branches doubles for at most $\lceil \frac{b-0.99\tilde{\lambda}}{0.19\tilde{\lambda}} \rceil \leq \frac{b-0.99\tilde{\lambda}}{0.19\tilde{\lambda}} + 1 \leq \frac{b}{0.19\tilde{\lambda}} \leq \frac{6b}{\tilde{\lambda}}$ levels of recursion and $2^x \geq 1$ for all positive x, so it holds that $f(b,c,d) \leq 2^{6b/\tilde{\lambda}}$.

Case 2: For $c \in (0.4\tilde{\lambda}, 0.61\tilde{\lambda}]$, the algorithm either takes case (b) or case (c). We claim the bound

$$f(b,c,d) \leq \frac{b}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6b/\tilde{\lambda}}.$$

It is trivially true for $b \leq 0.99\tilde{\lambda}$ and $d = 1$. Note that $0.6c \leq 0.6 \cdot 0.61\tilde{\lambda} \leq 0.4\tilde{\lambda}$ so that a call on a set with second parameter $0.6c$ guarantees that case (a) is executed on this call and we can apply the bound from Case 1. Now by induction on $b \geq 0.99\tilde{\lambda}$ and $d > 1$ it holds that

$$\begin{aligned}
f(b,c,d) \leq \max\{1,&\\
&2f(b - 0.19\tilde{\lambda}, c, d) + f(b, c, d/2),\\
&f(b, 0.6c, d) + f(b - 0.4\epsilon\tilde{\lambda}, c, d)\}\\
\leq \max\{1,&\\
&2 \cdot \frac{b}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6(b-0.19\tilde{\lambda})/\tilde{\lambda}} + \frac{b}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2(d/2))^{6b/\tilde{\lambda}},\\
&2^{6b/\tilde{\lambda}} + \frac{b - 0.4\epsilon\tilde{\lambda}}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6b/\tilde{\lambda}}\}\\
\leq \max\{1,&\\
&2 \cdot \frac{b}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6b/\tilde{\lambda}-1} + \frac{b}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6b/\tilde{\lambda}-1} \cdot (2 + 2\log_2 d - 2),\\
&(2 + 2\log_2 d)^{6b/\tilde{\lambda}} + \frac{b - 0.4\epsilon\tilde{\lambda}}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6b/\tilde{\lambda}}\}\\
= \quad &\frac{b}{0.4\epsilon\tilde{\lambda}}(2 + 2\log_2 d)^{6b/\tilde{\lambda}}.
\end{aligned}$$

Case 3: For $c \in [0.61\tilde{\lambda}, 1.01\tilde{\lambda}]$, we claim the bound

$$f(b,c,d) \leq \left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)^2 (2 + 2\log_2 d)^{6b/\tilde{\lambda}}.$$

It is trivially true for $b \leq 0.99\tilde{\lambda}$ and $d = 1$. Note that $0.6c \leq 0.6 \cdot 1.01\tilde{\lambda} \leq 0.61\tilde{\lambda}$, which implies that a call on a set with second parameter $0.6c$ guarantees that the bound from Case 2 can be applied to this call. Now by induction on $b \geq 0.99\tilde{\lambda}$ and $d > 1$, it holds that

$$\begin{aligned}
f(b,c,d) &\leq \max\{1, \\
&\qquad 2f(b-0.19\tilde{\lambda},c,d) + f(b,c,d/2), \\
&\qquad f(b,0.6c,d) + f(b-0.4\epsilon\tilde{\lambda},c,d)\} \\
&\leq \max\{1, \\
&\qquad 2\cdot\left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)^2 (2+2\log_2 d)^{6(b-0.19\tilde{\lambda})/\tilde{\lambda}} + \left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)^2 (2+2\log_2(d/2))^{6b/\tilde{\lambda}}, \\
&\qquad \frac{b}{0.4\epsilon\tilde{\lambda}}(2+2\log_2 d)^{6b/\tilde{\lambda}} + \left(\frac{b-0.4\epsilon\tilde{\lambda}}{0.4\epsilon\tilde{\lambda}}\right)^2 (2+2\log_2 d)^{6b/\tilde{\lambda}}\} \\
&\leq \max\{1, \\
&\qquad 2\cdot\left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)^2 (2+2\log_2 d)^{6b/\tilde{\lambda}-1} + \left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)^2 (2+2\log_2 d)^{6b/\tilde{\lambda}-1}\cdot(2+2\log_2 d-2), \\
&\qquad \frac{b}{0.4\epsilon\tilde{\lambda}}(2+2\log_2 d)^{6b/\tilde{\lambda}} + \left(\frac{b-0.4\epsilon\tilde{\lambda}}{0.4\epsilon\tilde{\lambda}}\right)\left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)(2+2\log_2 d)^{6b/\tilde{\lambda}}\} \\
&= \qquad \left(\frac{b}{0.4\epsilon\tilde{\lambda}}\right)^2 (2+2\log_2 d)^{6b/\tilde{\lambda}}.
\end{aligned}$$

Since $b \leq 3\tilde{\lambda}$, $c \leq 1.01\tilde{\lambda}$, and $d \leq |A|$, we obtain the desired bound $O(\epsilon^{-2}(\log|A|)^{O(1)})$. ∎

Finally, we claim that Lemmas 6.2 and 6.3 together imply Lemma 6.1. Applying Lemma 6.2 to the initial cluster A produces clusters A' whose total boundary size is $O(\epsilon^{-1}\partial A)$. Each cluster A' with $\partial A' > 3\tilde{\lambda}$ has no $(1-\epsilon)$-boundary-sparse set $U \subseteq A$ with $\partial_{G[A]}A \leq 1.01\tilde{\lambda}$, so in particular, for every 1.01-approximate minimum cut $S \subseteq V$ of G, the set $S \cap A'$ is non-$(1-\epsilon)$-boundary-sparse in A'.

For each cluster A' with $\partial A' \leq 3\tilde{\lambda}$, Lemma 6.3 produces $O(\epsilon^{-2}(\log|A'|)^{O(1)})$ many clusters, each of boundary size at most $\partial A'$, such that for every 1.01-approximate minimum cut $S \subseteq V$ of G, there is a partition $\mathcal{P}$ of the decomposition such that for each part $P \in \mathcal{P}$, the set $S \cap \bigcup_{A' \in P} A'$ is non-$(1-\epsilon)$-boundary-sparse in $\bigcup_{A' \in P} A'$. Recall that the total boundary size of the output of Lemma 6.2 is $O(\epsilon^{-1}\partial A)$. It follows that the total boundary size of the outputs of Lemma 6.3 over all A' is $O(\epsilon^{-1}\partial A)\cdot O(\epsilon^{-2}(\log|A|)^{O(1)}) = O(\epsilon^{-3}(\log|A|)^{O(1)}\partial A)$.

Given a 1.01-approximate minimum cut $S \subseteq V$ of G, we define the partition $\mathcal{P}$ so that every cluster A' with $\partial A' > 3\tilde{\lambda}$ produced by Lemma 6.2 is a singleton part in $\mathcal{P}$, and for each cluster A' that is further decomposed by Lemma 6.3, we add to $\mathcal{P}$ the partition guaranteed property (1) of Lemma 6.3.

By construction, for each part P in the final partition $\mathcal{P}$, the set $S \cap \bigcup_{A' \in P} A'$ is non-$(1-\epsilon)$-boundary sparse in $\bigcup_{A' \in P} A'$. For the running time, the algorithm calls Lemma 6.2 once and Lemma 6.3 up to $|A|$ times, for a total running time of $\tilde{O}(|A|^9)$. This concludes the proof of Lemma 6.1.

6.2 Large clusters

In this section, we present a fast algorithm to break down a large cluster A, i.e., a cluster with $\Omega(s_0)$ total volume, into a (potentially large) cluster A_0 and a (potentially empty) collection of small clusters $A_1,\ldots,A_r$. The key property of cluster A_0 is that any near-minimum cut U with small volume can be transformed, or "uncrossed", into a small enough cut U^* with $U^* \cap A_0 = \emptyset$. More formally, the main result of this section is the following.

Lemma 6.4. (Large cluster decomposition) *Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.1$ be a parameter and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a (potentially "large") set A_0 and "small" sets $A_1,\ldots,A_r$ such that*

1. *For each $i \in \{1,2,\ldots,r\}$, we have $\mathbf{vol}^W_G(A_i) \leq (\frac{10s_0}{\epsilon\tilde{\lambda}})\cdot s_0$.*
2. *The total weight $w(A_0,A_1,\ldots,A_r)$ of inter-cluster edges is at most $O(\epsilon^{-1})\cdot\partial A$.*
3. *For any set $U \subseteq A$ with $\partial_{G[A]}U \leq 1.01\tilde{\lambda}$ and $\mathbf{vol}^W_G(U) \leq s_0$, there exists $U^* \subseteq U$ with $U^* \cap A_0 = \emptyset$ and $w(U \setminus U^*, V \setminus A) + \partial_{G[A]}U^* \leq (1+\epsilon)\partial_{G[A]}U$.*

The algorithm runs in time $\tilde{O}((\frac{s_0}{\epsilon\tilde{\lambda}})^{O(1)}|E(G[A])|)$.

We begin with a high-level description of the algorithm. In the rest of the section let A be a large cluster.

Setting up the sources. To find cuts that are not uncrossable with A, the algorithm sets up a collection of flow problems. Let $U \subset A$ is a set with $\mathbf{vol}_G(U) \le s_0$ that is not uncrossable with A. More precisely, suppose that $w(U, V \setminus A) \ge (1+\epsilon)w(U, A \setminus U)$; we will use this definition of "uncrossable" for intuition in this section. Consider a flow problem in $G[A]$ where each vertex $v \in U$ has source $w(v, V \setminus A)$, each edge has flow capacity equal to its weight, and each vertex has sink capacity equal to $\eta d_G^W(v)$ for small enough $\eta > 0$. Within U, there is $w(U, V \setminus A)$ source originating from U, at most $w(U, A \setminus U)$ flow leaves through the edges $\partial_{G[A]}U$, and at most $\eta \mathbf{vol}_G^W(U) \le \eta s_0$ flow that is absorbed in U. Since $w(U, V \setminus A) \ge (1+\epsilon)w(U, A \setminus U)$, the flow cannot be feasible as long as $\eta > 0$ is small enough. It turns out that to make this argument work, we must set $\eta > 0$ small enough that there is not enough sink to absorb all the source. Therefore, we split the source up into many sub-sources, each of them small enough to be fully absorbed. More formally, we construct a collection of subsets $S \subseteq A$ that are small—$O(s_0)$ volume each—such that for each U with (i) $\partial_{G[A]}U \lesssim \tilde{\lambda}$ and (ii) $\mathbf{vol}^W(U) \le s_0$ that is (iii) not uncrossable, there exists a set S containing U. For each set S, we construct a flow where only vertices $v \in S$ receive their $w(v, V \setminus A)$ of source (with edge capacities and sinks unchanged). This is not quite achievable for technical purposes, but we show that it is possible *if the induced graph* $G[U]$ *has minimum cut* $\Omega(\lambda)$. In this case, if we run a variant of tree packing on $G[A]$, then there is a lot of "room" in $G[U]$ to pack trees, and the vertices in U will be contained in the union of $O(1)$ many trees, even after a post-processing step where each tree is broken down into pieces of volume $O(s_0)$. We then stitch the trees together by connecting pieces together in a bounded fashion so that one of the resulting sets S contains U.

Executing the flows. Consider a set U with $\partial_{G[A]}U \lesssim \tilde{\lambda}$ and $\mathbf{vol}^W(U) \le s_0$ that is not uncrossable. In the previous step we have constructed a "source set" $S \subseteq A$ with $\mathbf{vol}^W(S) \le O(s_0)$ and $U \subseteq S$. We cannot afford to run a separate flow where each vertex of S is a source, for each "source set" S, since each flow computation may take linear time. (Note that if the graph $G[A]$ were unweighted, then we can resort to local flow algorithms, but these do not have local guarantees in weighted graphs.)

Instead, we set up instances of *approximate isolating cuts* which allow us to execute the flows of many sources at once. We partition the collection of sources into polylog(n) subcollections that are vertex-disjoint (which is a requirement for approximate isolating cuts) and solve an instance of the approximate isolating cuts problem for each subcollection, obtaining a cut $C_i \subseteq A$ for each source S_i such that if $U \subseteq S_i$ then either $U \subseteq C_i$ or $U \setminus C_i$ is now uncrossable in $G[A \setminus C_i]$. We then remove all cuts C_i from A, and continue to the instance of the next subcollection where we restrict to the remaining sources $S \cap A$ on the remaining induced graph $G[A]$.

Removing the connected assumption. In general, $G[U]$ may not be well-connected. (It may even be disconnected, or very loosely connected.) The algorithm simply handles this issue by cycling through the approximate isolating cut problem instances some polylog(n) times. The analysis is a lot more complicated and technical, and we have to break down a general set U into well-connected sets and analyze the progress the algorithm makes in each cycle.

6.2.1 Setting up the sources

Let $W = \tilde{\lambda}^3/(50s_0^2)$. Let $A \subseteq V$. The algorithm first constructs the following unweighted multi-graph H supported on the same vertices and edges as $G[A]$: for each edge e in $G[A]$, let $w_H(e) = \lfloor w_G(e)/W \rfloor$, i.e., there are $\lfloor w_G(e)/W \rfloor$ parallel edges between the endpoints of e in H.

Well-connected case. We first treat the "well-connected case" established by the following lemma, whose proof is the main focus of this subsection. By well-connected, we refer to the condition in property (3) below that $G[U']$ has minimum cut at least $0.1\tilde{\lambda}$.

LEMMA 6.5. *Let* $A \subseteq V$ *be a cluster. There is an algorithm that runs in* $\tilde{O}((s_0/\tilde{\lambda})^{O(1)}|E(G[A])|)$ *time and computes a collection* $\mathcal{X}$ *of subsets of* A *such that*

1. $\mathbf{vol}^W(X) \le 3s_0$ *for each* $X \in \mathcal{X}$.

2. *Each vertex is in* $O((s_0/\tilde{\lambda})^5 \ln m)$ *many sets in* $\mathcal{X}$. *In particular,* $|\mathcal{X}| \le O((s_0/\tilde{\lambda})^5 \ln m)|A|$.

3. *Consider a set* $U \subseteq A$ *such that* $\partial_{G[A]}U \le 1.01\tilde{\lambda}$ *and* $\mathbf{vol}^W(U) \le s_0$. *Let* $U' \subseteq U$ *and suppose that the minimum cut in* $G[U']$ *is at least* $0.1\tilde{\lambda}$. *There exist a family of at most* 2525 *subsets in* $\mathcal{X}$ *whose union contains* U'.

We prove Lemma 6.5 with the help of the following lemmata.

Lemma 6.6. *Consider a set $U \subseteq A$ such that $\mathbf{vol}^W(U) \le s_0$ and the minimum cut in $G[U]$ is at least $0.1\tilde{\lambda}$. Then, the minimum cut in $H[U]$ is at least $\tilde{\lambda}/(25W) = s_0^2/(5\tilde{\lambda}^2)$.*

Proof. Let $S \subseteq U$ be the minimum cut in $H[U]$. Since $|U| \le \mathbf{vol}^W(U)/\delta \le \mathbf{vol}^W(U)/\lambda \le 1.01\mathbf{vol}^W(U)/\tilde{\lambda}$, there are at most $(|U|/2)^2 \le (1.01\mathbf{vol}^W(U)/(2\tilde{\lambda}))^2$ many edges between S and $U \setminus S$ in $H[U]$. For each such edge e, we have $w_G(e) \le w_H(e) \cdot W + W$. Summing over all edges e between S and $U \setminus S$ gives

$$\begin{aligned} w_G(S, U \setminus S) &\le w_H(S, U \setminus S) \cdot W + \left(\frac{1.01\mathbf{vol}^W(U)}{2\tilde{\lambda}}\right)^2 \cdot W \\ &\le w_H(S, U \setminus S) \cdot W + \left(\frac{1.01 s_0}{2\tilde{\lambda}}\right)^2 \cdot \frac{\tilde{\lambda}^3}{5 s_0^2} \\ &\le w_H(S, U \setminus S) \cdot W + 0.06\tilde{\lambda}. \end{aligned}$$

Rearranging and using that $w_G(S, U \setminus S) \ge 0.1\tilde{\lambda}$ gives $w_H(S, U \setminus S) \ge \tilde{\lambda}/(25W) = s_0^2/(5\tilde{\lambda}^2)$, as promised. ∎

We will need the following two lemmas, whose technical proofs are deferred to Section 6.2.3. First, we run a variant of tree packing on $G[A]$ where we pack forests instead, since $G[A]$ may not be well-connected enough to admit a large tree packing.

Lemma 6.7. *Let $\kappa \ge 1$ be a parameter. There is an algorithm that, given κ, runs in $O(\kappa |E(H)| \log^2 m)$ time and computes $\lceil \kappa \ln m \rceil$ forests in H such that*

1. *Each edge e in H participates in at most $100 \ln m$ forests.*
2. *For any induced subgraph $H[U]$ with minimum cut at least κ, all vertices belong to a single connected component for each of the forests.*

Then, the lemma below decomposes each forest in our packing into a collection of subtrees whose vertex sets have small volume.

Lemma 6.8. *Consider a tree T supported on a subset of vertices in A. We can compute subsets $V_1, \dots, V_\ell \subseteq V(T)$ satisfying the following.*

1. *$\mathbf{vol}^W(V_i) \le 3s_0$ for all i.*
2. *Each vertex $v \in A$ is in at most $d_T(v)$ many subsets V_i.*
3. *Consider any (not necessarily connected) subset $U \subseteq V(T)$ with $\mathbf{vol}^W(U) \le s_0$ and let $k = \partial_T U$ be the number of edges in T with exactly one endpoint in U. Then, there are at most k subsets V_i intersecting U, and their union contains U.*

The algorithm runs in $\tilde{O}(|V(T)|)$ time.

The algorithm applies Lemma 6.7 to $\kappa = \tilde{\lambda}/(25W) = s_0^2/(5\tilde{\lambda}^2)$, producing a collection $\mathcal{F}$ of $\lceil \tilde{\lambda}/(25W) \cdot \ln m \rceil$ forests. For each forest $F \in \mathcal{F}$, apply Lemma 6.8 to each connected component (tree) of F. Take the union of the output $\{V_1, \dots, V_\ell\}$ over all connected components, and let this collection of vertex subsets be $\mathcal{V}(F)$.

Our proof of Lemma 6.5 proceeds as follows: We set $\mathcal{X}$ as the union of $\mathcal{V}(F)$ over all $F \in \mathcal{F}$.

- Property (1) follows from property (1) of Lemma 6.8.
- For property (2), observe that there are $\lceil \tilde{\lambda}/(25W) \cdot \ln m \rceil = \lceil s_0^2/(5\tilde{\lambda}^2) \cdot \ln m \rceil$ forests $F \in \mathcal{F}$. For each $F \in \mathcal{F}$, by property (2) of Lemma 6.8, each vertex $v \in A$ is in at most $d_F(v)$ many subsets in $\mathcal{V}(F)$, and by property (1), each vertex $v \in A$ that joins some subset in $\mathcal{V}(F)$ must have $d_G^W(v) \le 3s_0$. Since F is a spanning forest of H, we have $d_F(v) \le_H (v) \le d_G^W(v)/W \le 3s_0/W \le O((s_0/\tilde{\lambda})^3)$. So in total, each vertex $v \in A$ is in at most $O((s_0/\tilde{\lambda})^3)$ subsets in $\mathcal{V}(F)$ for each of $O((s_0/\tilde{\lambda})^2 \ln m)$ many forests $F \in \mathcal{F}$, concluding property (2).

So all that we are left to show is property (3). For that we use the following lemma, which concludes the proof of Lemma 6.5.

LEMMA 6.9. *Consider a set $U \subseteq A$ such that $\partial_{G[A]}U \le 1.01\tilde{\lambda}$ and $\mathbf{vol}^W(U) \le s_0$. Let $U' \subseteq U$ and suppose that the minimum cut in $G[U']$ is at least $0.1\tilde{\lambda}$. There exists a forest F such that there are at most 2525 many subsets $V' \in \mathcal{V}(F)$ that intersect U', and their union contains U'.*

Proof. By construction of H, we have $\partial_H U \le \partial_{G[A]}U/W \le 1.01\tilde{\lambda}/W$. By property (1) of Lemma 6.7, each of the at most $1.01\tilde{\lambda}/W$ edges in $\partial_H U$ participates in at most $100 \ln m$ forests. Since there are $\lceil \tilde{\lambda}/(25W) \cdot \ln m \rceil$ many forests, there is a forest $F \in \mathcal{F}$ with

$$\partial_F U \le \frac{1.01\tilde{\lambda}/W \cdot 100 \ln m}{\tilde{\lambda}/(25W) \cdot \ln m} = 2525.$$

By Lemma 6.6, the minimum cut in $H[U']$ is at least $\kappa = \tilde{\lambda}/(25W)$, so property (2) of Lemma 6.7 guarantees that all vertices of U' belong to a single tree T of F. Then, property (3) of Lemma 6.8 applied to subset $U \cap V(T)$ guarantees at most $\partial_T U \le \partial_F U \le 2525$ subsets $V_i \in \mathcal{V}(F)$ intersecting $U \cap V(T)$, and their union contains $U \cap V(T)$ which contains U'. ◻

To finish the proof of Lemma 6.5, it remains to bound the running time. Since $\kappa = O(s_0/\tilde{\lambda})^{O(1)}$ and $|E(H)| \le |E(G[A])|$, Lemma 6.7 runs in $\tilde{O}((s_0/\tilde{\lambda})^{O(1)}|E(G[A])|)$ time. There are $O(s_0/\tilde{\lambda})^{O(1)} \log m$ many forests, and Lemma 6.8 runs in $\tilde{O}(|V(F)|)$ time total over all the trees of each forest $F \in \mathcal{F}$. It follows that the overall running time is $\tilde{O}((s_0/\tilde{\lambda})^{O(1)}|E(G[A])|)$.

General case. We show the following lemmata which relate the general case to the well-connected case. We remark that the partition specified below is not computed by the algorithm; we only need its existence to prove correctness.

LEMMA 6.10. *Consider a set $U \subseteq A$ such that $\partial_{G[A]}U \le 1.01\tilde{\lambda}$ and $\mathbf{vol}^W(U) \le s_0$. There exists an integer k and a partition $U_1, \ldots, U_k$ of U such that*

1. *For each $i \in [k]$, the minimum cut of $G[U_i]$ is at least $0.1\tilde{\lambda}$.*
2. *For at least $k/2 - 5$ many indices $i \in [k]$, the boundary $\partial_{G[A]}U_i$ is at most $0.4\tilde{\lambda}$.*
3. *The total weight $w(U_1, \ldots, U_k)$ of inter-cluster edges is at most $0.1(k-1)\tilde{\lambda}$.*

More generally, for any $A' \subseteq A$, if we consider the subset $I \subseteq [k]$ of indices $i \in [k]$ with $U_i \cap A' \ne \emptyset$, then

2'. *For at least $|I|/2 - 5$ many indices $i \in I$, the boundary $\partial_{G[A']}(U_i \cap A')$ is at most $0.4\tilde{\lambda}$.*

Proof. Iteratively partition the vertex set U as follows: start with the collection $\mathcal{P} = \{U\}$, and while there exists a vertex set $U' \in \mathcal{P}$ such that $G[U']$ has a cut of weight less than $0.1\tilde{\lambda}$, replace U' in $\mathcal{P}$ with the two sides $U'_1, U'_2 \subseteq U'$ of the cut. Let $\mathcal{P} = \{U_1, \ldots, U_k\}$ be the final partition. Property (1) holds by construction. The algorithm makes $k-1$ many cuts of weight at most $0.1\tilde{\lambda}$ each, so the total weight of inter-cluster edges is at most $(k-1) \cdot 0.1\tilde{\lambda}$, proving property (3).

We now show property (2). Suppose for contradiction that fewer than $k/2 - 5$ many indices $i \in [k]$ have $\partial_{G[A]}U_i \le 0.4\tilde{\lambda}$. Equivalently, more than $k/2+5$ many indices $i \in [k]$ have $\partial_{G[A]}U_i > 0.4\tilde{\lambda}$. Consider the partition $U_1, \ldots, U_k, A \setminus U$ of A; the total weight $w(U_1, \ldots, U_k, A \setminus U) = \frac{1}{2}(\sum_i \partial_{G[A]}U_i + \partial_{G[A]}U)$ of inter-cluster edges is at least $\frac{1}{2} \cdot (k/2+5) \cdot 0.4\tilde{\lambda}$. On the other hand, $w(U_1, \ldots, U_k, A \setminus U) = w(U_1, \ldots, U_k) + \partial_{G[A]}U \le 0.1(k-1)\tilde{\lambda} + 1.01\tilde{\lambda}$ by property (3) and the assumption $\partial_{G[A]}U \le 1.01\tilde{\lambda}$. Altogether,

$$w(U_1, \ldots, U_k, A \setminus U) \le 0.1(k-1)\tilde{\lambda} + 1.01\tilde{\lambda} < \frac{1}{2} \cdot (k/2+5) \cdot 0.4\tilde{\lambda} \le w(U_1, \ldots, U_k, A \setminus U),$$

a contradiction.

Finally, to show property (2'), consider any $A' \subseteq A$. Consider the sequence of cuts of weight less than $0.1\tilde{\lambda}$ that were made to construct $\mathcal{P} = \{U_1, \ldots, U_k\}$. Repeat this sequence of cuts on $G[A']$ as follows: for the next cut $(C, \bar{C})$ of $G[U']$ for some $U' \subseteq A$, if $C \cap A' = \emptyset$ or $\bar{C} \cap A' = \emptyset$ then do nothing; otherwise make the cut $(C \cap A', \bar{C} \cap A')$ and split U' into $C \cap A'$ and $\bar{C} \cap A'$. It is clear that the new output $\mathcal{P}'$ is $\{U_i \cap A' : i \in I\}$. In total, there are $|I| - 1$ cuts that split a set into two, so the total weight in $G[A']$ of inter-cluster edges of $\mathcal{P}'$ is at most $(|I| - 1) \cdot 0.1\tilde{\lambda}$. The rest of property (2') follows similarly from the proof of property (2): suppose for contradiction that more than $|I|/2 + 5$ many indices in I have $\partial_{G[A']} U_i > 0.4\tilde{\lambda}$; then writing $\mathcal{P}' = U_{i_1} \cap A', \ldots, U_{i_{|I|}} \cap A'$, we obtain

$$\begin{aligned} w_{G[A']}(U_{i_1} \cap A', \ldots, U_{i_{|I|}} \cap A', A' \setminus U) &= w_{G[A']}(U_{i_1} \cap A', \ldots, U_{i_{|I|}} \cap A') + \partial_{G[A']}(U \cap A') \\ &\le w_{G[A']}(U_{i_1} \cap A', \ldots, U_{i_{|I|}} \cap A') + \partial_{G[A]} U \\ &\le 0.1|I|\tilde{\lambda} + 1.01\tilde{\lambda} \\ &< \frac{1}{2} \cdot (|I|/2 + 5) \cdot 0.4\tilde{\lambda} + \frac{1}{2} \cdot 1.01\tilde{\lambda} \\ &\le w_{G[A']}(U_{i_1} \cap A', \ldots, U_{i_{|I|}} \cap A', A' \setminus U), \end{aligned}$$

a contradiction. ◻

Stitching pieces together. Let U' be defined as in Lemma 6.5. Property (3) of Lemma 6.5 guarantees that there are at most 2525 sets in $\mathcal{X}$ whose union contains a set U' satisfying the conditions. In the lemma below, we stitch together sets in $\mathcal{X}$ so that there is now a single set containing U', as long as the induced subgraph $G[U']$ is moderately connected: instead of a minimum cut of $\Omega(\tilde{\lambda})$, we can allow $\Omega(\gamma\tilde{\lambda})$ for any $\gamma = 1/\text{polylog}(n)$.

LEMMA 6.11. *Let $A \subseteq V$ be a cluster and let γ, K be parameters. There is an algorithm that runs in $\tilde{O}((\frac{s_0 \ln m}{\tilde{\lambda}\gamma})^{O(K)}|E(G[A])|)$ time and computes a collection $\mathcal{S}$ of subsets of A such that*

1. $\mathbf{vol}^W(S) \le O(Ks_0)$ *for each $S \in \mathcal{S}$.*
2. *Each vertex is in $O(\frac{s_0 \ln m}{\tilde{\lambda}\gamma})^{O(K)}$ many sets in $\mathcal{S}$. In particular, $|\mathcal{S}| \le O(\frac{s_0 \ln m}{\tilde{\lambda}\gamma})^{O(K)}|A|$.*
3. *Consider a set $U \subseteq A$ such that $\partial_{G[A]} U \le 1.01\tilde{\lambda}$ and $\mathbf{vol}^W(U) \le s_0$, and consider a subset $U' \subseteq U$. Suppose that the minimum cut in $G[U']$ is at least $\gamma\tilde{\lambda}$, and suppose that there exist (not necessarily disjoint) $U_1, \ldots, U_k \subseteq U$ with $k \le K$ such that $U_1 \cup \cdots \cup U_k \supseteq U'$ and each $G[U_i]$ has minimum cut at least $0.1\tilde{\lambda}$. Then, there exists a set $S \in \mathcal{S}$ containing U'.*

Proof. The algorithm begins with the set $\mathcal{X}$ from Lemma 6.5. Construct an auxiliary graph H whose vertices are the sets $X \in \mathcal{X}$. Add an edge between $X, X' \in \mathcal{X}$ if $X \cap X' \neq \emptyset$ or there exists an edge between X and X' of weight at least $3\gamma\tilde{\lambda}^3/s_0^2$. This completes the construction of H.

We show that the degree of H is $O((s_0/\tilde{\lambda})^8(\ln m)/\gamma)$. For a given $X \in \mathcal{X}$, since $\mathbf{vol}^W(X) \le 3s_0$ by property (1) of Lemma 6.5, and since each vertex $v \in X$ has $d^W(v) \ge \lambda$, we have $|X| \le 3s_0/\lambda \le O(s_0/\tilde{\lambda})$. By property (2), each vertex $v \in X$ is in $O((s_0/\tilde{\lambda})^5 \ln m)$ many sets $X \in \mathcal{X}$, so the total number of edges (X, X') added incident to X because $X \cap X' \neq \emptyset$ is $O((s_0/\tilde{\lambda})^6 \ln m)$. Next, we bound the number of edges (X, X') added because of an edge between X and X' of weight at least $3\gamma\tilde{\lambda}^3/s_0^2$. Note that from $\mathbf{vol}^W(X) \le 3s_0$ we obtain that there are at most $3s_0/(3\gamma\tilde{\lambda}^3/s_0^2) = (s_0/\tilde{\lambda})^3/\gamma$ many edges of weight at least $3\gamma\tilde{\lambda}^3/s_0^2$ incident to X. For each of these edges, there are $O((s_0/\tilde{\lambda})^5 \ln m)$ many sets $X' \in \mathcal{X}$ containing the other endpoint, for a total of $O((s_0/\tilde{\lambda})^8(\ln m)/\gamma)$ many edges (X, X').

The algorithm considers all connected subgraphs of at most $2525K$ vertices in H. For each connected subgraph with vertices $X_1, \ldots, X_\ell$, add $X_1 \cup \cdots \cup X_\ell$ to $\mathcal{S}$. Since $\mathbf{vol}^W(X_i) \le 3s_0$ for each $i \in [\ell]$, property (1) holds by construction. By the degree bound on H, each vertex X in H is in $O((s_0/\tilde{\lambda})^8(\ln m)/\gamma)^{O(K)}$ many connected subgraphs (since we can identify each one by a spanning tree of size $O(K)$). Each vertex is in at most $O((s_0/\tilde{\lambda})^5 \ln m)$ many sets $X \in \mathcal{X}$, each of which shows up $O((s_0/\tilde{\lambda})^8(\ln m)/\gamma)^{O(K)}$ times in the construction of $\mathcal{S}$, fulfilling property (2).

For the rest of the proof, we prove property (3). For each $i \in [k]$, since $U_i \subseteq U$ and $G[U_i]$ has minimum cut at least $0.1\tilde{\lambda}$, property (3) of Lemma 6.5 guarantees at most 2525 sets $X \in \mathcal{X}$ whose union contains U_i. In total, over all $i \in [k]$, there are at most $2525K$ many subsets $X \in \mathcal{X}$ whose union contains $U_1 \cup \cdots \cup U_k \supseteq U'$. It remains to show that these sets, when viewed as vertices in H, form a connected component in H.

Let Y be this set of vertices in H. Suppose for contradiction that there is a bipartition $\{X_1, \ldots, X_\ell\}$, $\{X'_1, \ldots, X'_{\ell'}\}$ of Y such that there is no edge in H between any pair of vertices X_i, X'_j. Then, $Z = X_1 \cup \cdots \cup X_\ell$ and $Z' = X'_1 \cup \cdots \cup X'_{\ell'}$ are disjoint vertex sets in A since a common vertex would imply an edge between the bipartition in H. In particular, since $Z \cup Z'$ contains U', we have that $Z \cap U'$ and $Z' \cap U'$ partition U'. Since $\mathbf{vol}^W(U') \leq \mathbf{vol}^W(U) \leq s_0$, we have $|U'| \leq s_0/\lambda$, so there are at most $(|U'|/2)^2 \leq s_0^2/(4\lambda^2)$ many edges between $Z \cap U'$ and $Z' \cap U'$. Since the minimum cut in $G[U']$ is at least $\gamma\tilde{\lambda}$, the total weight of edges between $Z \cap U'$ and $Z' \cap U'$ in G is at least $\gamma\tilde{\lambda}$. It follows that at least one edge between $Z \cap U'$ and $Z' \cap U'$ has weight at least $\gamma\tilde{\lambda}/(s_0^2/(4\lambda^2)) = 4\gamma\tilde{\lambda}\lambda^2/s_0^2 \geq 3\gamma\tilde{\lambda}^3/s_0^2$. By construction of the edges of H, there must be an edge between some X_i, X'_j, a contradiction.

Finally, we bound the running time. By property (2) of Lemma 6.5, each vertex appears at most $O((s_0/\tilde{\lambda})^5 \ln m)$ times in $\mathcal{X}$, so in the construction of H, it is visited $O((s_0/\tilde{\lambda})^5 \ln m)$ times in total. Also, $|\mathcal{X}| \leq O((s_0/\tilde{\lambda})^5 \ln m) \cdot |A|$. Therefore, constructing H takes time $O((s_0/\tilde{\lambda})^5 \ln m)|E(G[A])|$. For each vertex in H, there are $O((s_0/\tilde{\lambda})^8 (\ln m)/\gamma)^{O(K)}$ many connected components with at most $2525K$ vertices, and these can be enumerated in the same asymptotic running time. For each connected subgraph with vertices $X_1, \ldots, X_\ell$, we have $|X_1 \cup \cdots \cup X_\ell| \leq \sum_i |X_i| \leq \sum_i \mathbf{vol}(X_i)/\lambda \leq \sum_i 3s_0/\lambda \leq \sum_i O(s_0/\tilde{\lambda})$, so it takes $O(Ks_0/\tilde{\lambda})$ time to add the set $X_1 \cup \cdots \cup X_\ell$ to $\mathcal{S}$. Overall, the running time is bounded by $\tilde{O}((\frac{s_0 \ln m}{\tilde{\lambda}\gamma})^{O(K)}|E(G[A])|)$. $\square$

6.2.2 Executing the flows With the sources established by Lemma 6.11, we now set up and run the flow instances. We need the following algorithm which is based on the *approximate isolating cuts* algorithm of [LNPS23]. We defer the details to Section 6.2.3. Given a large cluster A, the algorithm repeatedly removes subsets that form small clusters from A. In this subsection we use A' to denote the current version of A, which potentially has some small clusters already removed.

Given a cluster A' and a family of vertex-disjoints sets of $\mathcal{S}$ (where $\mathcal{S}$ was constructed for A'), the algorithm in the next lemma constructs a family of subsets C_i of A' such that three properties hold, which are crucial to prove Lemma 6.4. Intuitively, Property (3) below says that the cut $U^\dagger$ can be uncrossed to U^* that is contained within C_i, but we are also penalized for boundary edges in the difference $U^\dagger \setminus U^*$; see Figure 1. Note that we do not guarantee that U^* equals $U^\dagger \cap C_i$ as standard uncrossing may suggest; we only guarantee that is it a subset.

Lemma 6.12. *Let $A' \subseteq V$ be a cluster, let $\epsilon \leq 0.1$ be a parameter, and let $S_1, \ldots, S_k$ be disjoint vertex subsets of A'. There is an algorithm that computes, for each $i \in [k]$, a set $C_i \subseteq A'$ such that*

1. $\mathbf{vol}^W(C_i) \leq (\frac{10s_0}{\epsilon\tilde{\lambda}})\,\mathbf{vol}^W(S_i)$.
2. $\partial_{G[A']}C_i \leq (1-\epsilon)w(C_i, V \setminus A')$.
3. *For any set $U^\dagger \subseteq S_i$ with $\mathbf{vol}^W(U^\dagger) \leq s_0$, there exists $U^* \subseteq U^\dagger \cap C_i$ such that $w(V \setminus A', U^\dagger \setminus U^*) + \partial_{G[A']}U^* \leq (1+\epsilon)\partial_{G[A']}U^\dagger$. Moreover, if $U^* \neq U^\dagger$ then $\partial_{G[A']}U^\dagger \geq 0.2\lambda$.*

The sets C_i are vertex-disjoint, and the algorithm runs in time $\tilde{O}((\frac{s_0}{\epsilon\tilde{\lambda}})^{O(1)}|E(G[A'])|)$ (over all $i \in [k]$).

We now prove the main lemma of this section, restated below. In the lemma, we decompose a cluster A into sets A_i (that result from the C_i of the lemma above) and the "leftover" piece, called A_0. The first property of the lemma below guarantees that each set A_i with $i \geq 1$ is small and, thus, the algorithm for small clusters can be used to uncross all approximate minimum cuts in it. The second property guarantees that the boundary $\partial_{G[A']}C_i$ is a factor of $(1-\epsilon)$ smaller than the piece of the boundary $\partial A'$ that is cut out when C_i is removed from A'. This property can be used to bound the number of new inter-cluster edges that are created when C_i is removed from A'. Think of U in the third property as being an approximately minimum cut, which is then "uncrossed" with A_0: the new U^* no longer crosses the leftover piece A_0 and its cut size $\partial_{G[A]}U^*$ is only a factor $(1+\epsilon)$ larger. For technical reasons, we also have to account for the total weight $w(U \setminus U^*, V \setminus A)$ of "external edges" lost moving from U to U^*.

Lemma 6.4. (Large cluster decomposition) *Let $A \subseteq V$ be a cluster, let $0 < \epsilon \leq 0.1$ be a parameter and let $\tilde{\lambda} \leq 1.01\lambda$ be an approximate lower bound to the minimum cut known to the algorithm. There is an algorithm that partitions A into a (potentially "large") set A_0 and "small" sets $A_1, \ldots, A_r$ such that*

1. *For each $i \in \{1, 2, \ldots, r\}$, we have $\mathbf{vol}_G^W(A_i) \leq (\frac{10s_0}{\epsilon\tilde{\lambda}}) \cdot s_0$.*

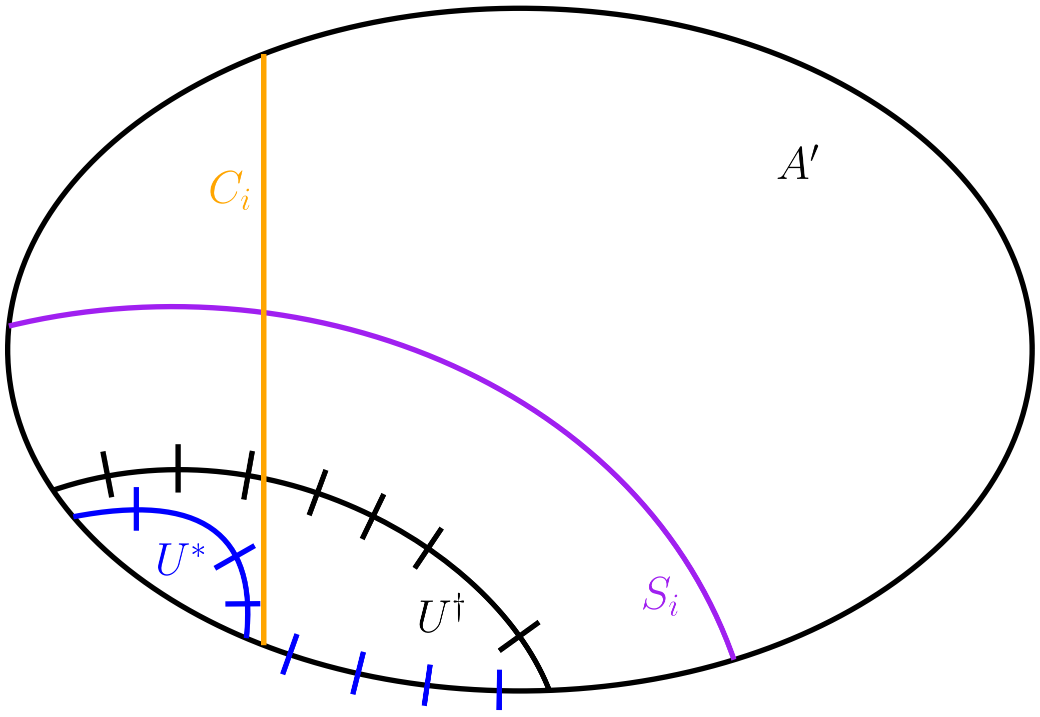

Figure 1: The set $U^\dagger$ is uncrossed to $U^* \subseteq U^\dagger \cap C_i$ according to property (3) of Lemma 6.12. The blue edges have weight at most $(1+\epsilon)$ times the black edges.

2. *The total weight $w(A_0, A_1, \ldots, A_r)$ of inter-cluster edges is at most $O(\epsilon^{-1}) \cdot \partial A$.*
3. *For any set $U \subseteq A$ with $\partial_{G[A]} U \leq 1.01\tilde{\lambda}$ and $\mathbf{vol}_G^W(U) \leq s_0$, there exists $U^* \subseteq U$ with $U^* \cap A_0 = \emptyset$ and $w(U \setminus U^*, V \setminus A) + \partial_{G[A]} U^* \leq (1+\epsilon)\partial_{G[A]} U$.*

The algorithm runs in time $\tilde{O}((\frac{s_0}{\epsilon\tilde{\lambda}})^{O(1)}|E(G[A])|)$.

For the rest of the section, we describe and analyze the algorithm for Lemma 6.4, called $\mathcal{A}$.
The algorithm $\mathcal{A}$ consists of the following steps:

1. Apply Lemma 6.11 with $\gamma = \frac{\epsilon\tilde{\lambda}}{100 s_0}$ and $K = 10$ to obtain the set $\mathcal{S}$.
2. Partition $\mathcal{S}$ into subsets $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$, where $\ell \leq (\frac{s_0 \ln m}{\epsilon\tilde{\lambda}})^{O(1)}$ and the sets in each $\mathcal{S}_i$ are vertex-disjoint. This can be done by partitioning the sets in $\mathcal{S}$ into independent sets of the graph H in the proof of Lemma 6.11, which has degree $O((s_0/\tilde{\lambda})^8(\ln m)/\gamma) \leq (\frac{s_0 \ln m}{\epsilon\tilde{\lambda}})^{O(1)}$.
3. Set $A' \leftarrow A$ and $\mathcal{C} \leftarrow \emptyset$.
4. Repeat $O(\log|A|)$ times:
   (a) For every set $\mathcal{S}_j$ in $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$ execute the following steps:
      i. For each $S_i \in \mathcal{S}_j$ run the algorithm in Lemma 6.12 with parameter $\epsilon/2$, the current cluster A' and the sets $S_i \cap A'$ (discarding any sets that are empty), which computes for every set $S_i \cap A'$ with $S \in \mathcal{S}_j$ a set C_i
      ii. Add all sets C_i as a new elements to $\mathcal{C}$
      iii. Set $A' \leftarrow A' \setminus \cup_i C_i$.
   (b) Set A_0 as the final A', and rename the sets in $\mathcal{C}$ to be $\{A_1, \ldots, A_r\}$.

This concludes the description of the algorithm, which runs in time $\tilde{O}((\frac{s_0}{\epsilon\tilde{\lambda}})^{O(1)}|E(G[A])|)$ because Lemma 6.11 and Lemma 6.12 run in the same asymptotic time and the latter is called $\ell \log |A| \leq (\frac{s_0 \ln m}{\epsilon\tilde{\lambda}})^{O(1)}$ times.

We first prove Properties (1) and (2). Property (1) follows from Property (1) of Lemma 6.12 and property (1) of Lemma 6.11. For Property (2), we track the value of $\partial A'$ throughout the algorithm. Every time a collection of sets C_i are removed from A', the value $\partial A'$ loses $\sum_i w(C_i, V \setminus A')$ and gains at most $\sum_i \partial_{G[A']} C_i$. For each i, we have $\partial_{G[A']} C_i \leq (1-\epsilon/2) w(C_i, V \setminus A')$ by property (2) of Lemma 6.12, so the net loss of $\partial A'$ is at least $(\epsilon/2)\sum_i w(C_i, V \setminus A') \geq (\epsilon/2)\sum_i \partial_{G[A']} C_i$. We charge the new edges cut, which have total weight at most $\sum_i \partial_{G[A']} C_i$, to the decrease in $\partial A'$. It follows that the total weight of inter-cluster edges cannot exceed $O(\epsilon^{-1})$ times the original value of $\partial A'$, which is ∂A.

For the remainder of the proof, we focus on property (3). Fix a set $U \subseteq A$ with $\partial_{G[A]} U \leq 1.001\tilde{\lambda}$ and $\mathbf{vol}^W(U) \leq s_0$. Let $U_1, \ldots, U_k$ be a partition the fulfills the conditions of Lemma 6.10. Note that each U_i is a subset U' of A that satisfies the conditions in property (3) of Lemma 6.11 with $\gamma = 0.1$ and $k = 1$. Thus, Lemma 6.11 implies that each U_i is contained in some set that belongs to the collection $\mathcal{S}$ computed by the above algorithm in the first step.

Let I be the subset of indices $i \in [k]$ with $U_i \cap A' \neq \emptyset$ (for the current set A'), which may lose elements over time as A' shrinks.

Lemma 6.13. *On each cycle through $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$, the set I loses at least $|I|/2 - 5$ elements.*

Proof. By property (2') of Lemma 6.10, there are at least $|I|/2-5$ indices $i \in I$ such that $\partial_{G[A']}(U_i \cap A') \leq 0.4\tilde{\lambda}$. We show that after another cycle through $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$, each of these indices is removed from I. Let $i \in I$ be an index with $\partial_{G[A']}(U_i \cap A') \leq 0.4\tilde{\lambda}$. Recall that U_i is contained in some set S_i, which belongs to some $\mathcal{S}_j$. As the algorithm goes through $\mathcal{S}_1, \ldots, \mathcal{S}_{j-1}$, the set A' can only shrink, so we still have $U_i \cap A' \neq \emptyset$ and $\partial_{G[A']}(U_i \cap A') \leq 0.4\tilde{\lambda}$ right before processing $\mathcal{S}_j$.

To avoid clutter, define $U_i' = U_i \cap A'$ and $S_i' = S_i \cap A'$. Recall that on iteration $\mathcal{S}_j$, the algorithm runs Lemma 6.12 on the sets $S \cap A'$ for $S \in \mathcal{S}_j$. In particular, we have $U_i' = U_i \cap A' \subseteq S_i \cap A' = S_i'$ and $\mathbf{vol}^W(U_i') \leq s_0$, so U_i' satisfies the conditions of property (3) of Lemma 6.12 (for $U^\dagger \leftarrow U_i'$ and $S_i \leftarrow S_i'$). Let $C_i \subseteq A'$ be the corresponding set for $S_i \leftarrow S_i'$, and let U_i^* be the set U^* guaranteed by property (3) for $U^\dagger \leftarrow U_i'$, so that $U_i^* \subseteq U_i' \cap C_i$. If $U_i^* = U_i'$, then $U_i \cap A' = U_i^* \subseteq C_i$ and C_i is removed from A' on iteration $\mathcal{S}_j$, which means index i is removed from I, a contradiction. Therefore, $U_i^* \subsetneq U_i'$.

Consider the quantity $\partial_G(U_i' \setminus U_i^*)$. All edges involved are either in $\partial_{G[A']}(U_i' \setminus U_i^*)$ or $w(U_i' \setminus U_i^*, V \setminus A')$. The former expression can be further divided: the edges in $\partial_{G[A']}(U_i' \setminus U_i^*)$ are either in $\partial_{G[A']} U_i'$ or $\partial_{G[A']} U_i^*$ (or both). Applying the conditions of property (3), we obtain

$$\lambda \leq \partial_G(U_i' \setminus U_i^*) \leq \partial_{G[A']} U_i' + \partial_{G[A']} U_i^* + w(U_i' \setminus U_i^*, V \setminus A') \leq (2+\epsilon)\partial_{G[A']} U_i'.$$

It follows that $\partial_{G[A']} U_i' > 0.4\tilde{\lambda}$, a contradiction. It follows that index i leaves set I after processing $\mathcal{S}_j$. ◻

As long as $|I| \geq 11$, we have $|I|/2 - 5 > 0$, so the set I loses a constant fraction of its vertices each cycle. It follows that after $O(\log |A|)$ cycles, we have $|I| \leq 10$.

At this point, let $A^\dagger$ be the current value of A', and consider $U \cap A^\dagger$ in $G[A^\dagger]$, which also satisfies $\partial_{G[A^\dagger]}(U \cap A^\dagger) \leq \partial_{G[A]} U \leq 1.001\tilde{\lambda}$ and $\mathbf{vol}(U \cap A^\dagger) \leq s_0$. We define a refinement of this partition as follows (these are not steps executed by our algorithm, as U is unknown, but are just used for refining $\mathcal{P}$): Iteratively partition the vertex set $U \cap A^\dagger$ as follows: start with the collection $\mathcal{P} = \{U \cap A^\dagger\}$, and while there exists a vertex set $U' \in \mathcal{P}$ such that $G[U']$ has a cut of weight less than $\gamma\tilde{\lambda} = \frac{\epsilon\tilde{\lambda}}{100 s_0}\tilde{\lambda}$, replace U' in $\mathcal{P}$ with the two sides $U_1', U_2' \subseteq U'$ of the cut. Let $\mathcal{P}$ be the final partition of $U \cap A^\dagger$.

Lemma 6.14. *After one more cycle through $\mathcal{S}_1, \ldots, \mathcal{S}_\ell$ (and setting $A_0 \leftarrow A'$ at the end), for each $U' \in \mathcal{P}$, there exist $A^* \supseteq A_0$ and $U^* \subseteq U' \cap A^* \setminus A_0$ such that $w(V \setminus A^*, (U' \cap A^*) \setminus U^*) + \partial_{G[A^*]} U^* \leq (1+\epsilon/2)\partial_{G[A^*]}(U' \cap A^*)$. Moreover, if $U^* \neq U' \cap A^*$ then $\partial_{G[A^*]}(U' \cap A^*) \geq 0.2\lambda$.*

Proof. In the construction of $\mathcal{P}$, we make at most $|U \cap A^\dagger| \leq |U| \leq s_0/\lambda \leq 2s_0/\tilde{\lambda}$ many cuts of weight at most $\gamma\tilde{\lambda} = \frac{\epsilon\tilde{\lambda}}{100 s_0}\tilde{\lambda}$, so each $U' \in \mathcal{P}$ at the end has boundary

$$\partial_{G[A^\dagger]} U' \leq \partial_{G[A^\dagger]}(U \cap A^\dagger) + 2s_0/\tilde{\lambda} \cdot \gamma\tilde{\lambda} \leq 1.001\tilde{\lambda} + 0.02\epsilon\tilde{\lambda} \leq 1.01\tilde{\lambda}.$$

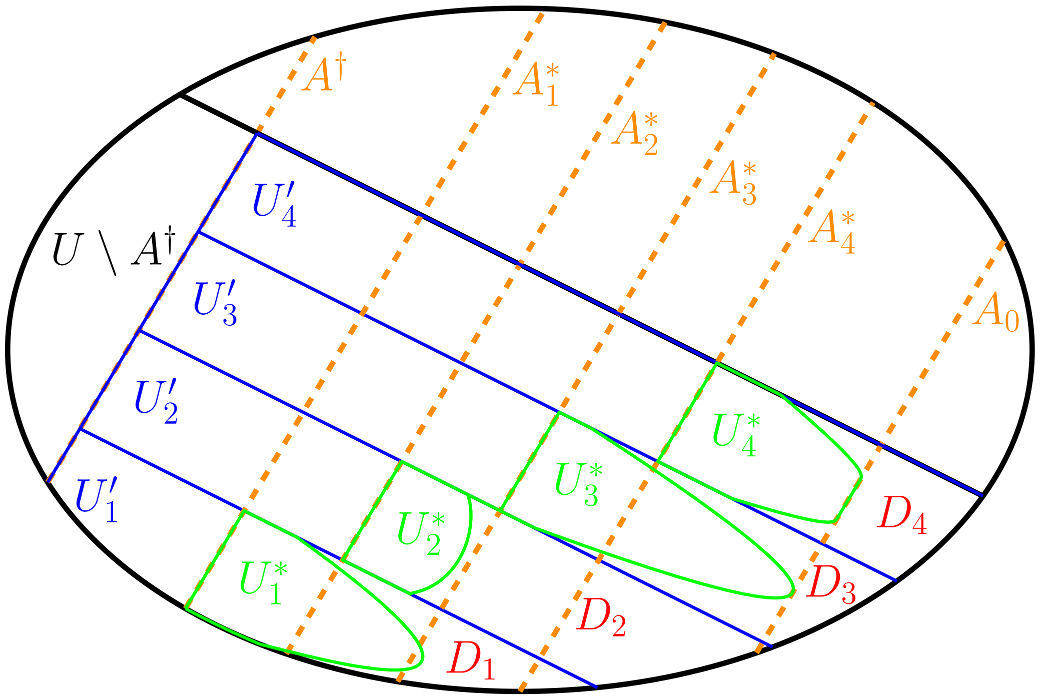

Figure 2: The variables defined for the remainder of the proof. The black ellipse represents the cluster A and the bottom half (below the black diagonal) is the set U.

By construction, $G[U']$ has minimum cut at least $\gamma\tilde{\lambda}$, so applying property (3) of Lemma 6.11 on $U' \subseteq U$ and the (at most 10) sets U_i for $i \in I$ guarantees a set $S' \in \mathcal{S}$ containing U'. Let $\mathcal{S}_j$ be the set containing S'.

As the algorithm $\mathcal{A}$ goes through $\mathcal{S}_1, \ldots, \mathcal{S}_{j-1}$, the set A' can only shrink, so we still have $\partial_{G[A']}(U' \cap A') \leq 1.01\tilde{\lambda}$ right before processing $\mathcal{S}_j$. Recall that on iteration $\mathcal{S}_j$, the algorithm runs Lemma 6.12 on the sets $S \cap A'$ for $S \in \mathcal{S}_j$. In particular, we have $U' \cap A' \subseteq S' \cap A'$, so the conditions of property (3) of Lemma 6.12 are satisfied (for $U^\dagger \leftarrow U' \cap A'$ and $S_i \leftarrow S' \cap A'$). We obtain a set $U^* \subseteq U' \cap A' \cap C_i$ with $w(V \setminus A', (U' \cap A') \setminus U^*) + \partial_{G[A']}U^* \leq (1 + \epsilon/2)\partial_{G[A']}(U' \cap A')$. We set $A^* = A'$ for the current set A' (which might be a superset of the final set A') which implies that the inequality in Lemma 6.14 holds.

If $U^* \neq U' \cap A^*$ then $\partial_{G[A^*]}(U' \cap A^*) \geq 0.2\lambda$ by the additional guarantee of property (3) of Lemma 6.12. At the end of the cycle, we have $A^* \supseteq A' = A_0$ because A' can only lose elements over time, and $U^* \subseteq U' \cap A^* \setminus A_0$ because $U^* \subseteq U' \cap A^* \cap C_i$ and $A_0 \cap C_i = \emptyset$ (since C_i is removed from A' on iteration $\mathcal{S}_j$). □

For the remainder of the proof, we encourage the reader to use Figure 2 as a reference. Suppose the algorithm $\mathcal{A}$ terminates after the cycle analyzed by Lemma 6.14. Let us denote the sets in $\mathcal{P}$ by $U_1', \ldots, U_k'$. Lemma 6.14 applied to each U_i' shows the existence of a set $A_i^* \supseteq A_0$ and a set $U_i^* \subseteq U_i' \cap A_i^* \setminus A_0$. To avoid clutter, define $D_i = (U_i' \cap A_i^*) \setminus U_i^*$. We finally set $U^* = U \setminus \cup_i D_i$.

Recall that A is a cluster and we need to show a claim about a set $U \subseteq A$. Observe that U can be decomposed as $U = (U \setminus A^\dagger) \cup (\cup_i U_i')$, and U^* can be decomposed as $U^* = (U \setminus A^\dagger) \cup (\cup_i (U_i' \setminus D_i))$. We write

$$\begin{aligned}
w(D_i, U^*) &= w(D_i, (U \setminus A^\dagger) \cup (\cup_j (U_j' \setminus D_j))) \\
&= w(D_i, U \setminus A^\dagger) + \sum_{j \in [k]} w(D_i, U_j' \setminus D_j) \\
&= w(D_i, U \setminus A^\dagger) + w(D_i, U_i' \setminus D_i) + \sum_{j \neq i} w(D_i, U_j' \setminus D_j) \\
&= w(D_i, (U \setminus A^\dagger) \cup (U_i' \setminus D_i)) + \sum_{j \neq i} w(D_i, U_j' \setminus D_j) \\
&\leq w(D_i, (A \setminus A_i^*) \cup (U_i' \cap A_i^* \setminus D_i)) + \sum_{j \neq i} w(U_i', U_j') \\
&= w(D_i, A \setminus A_i^*) + w(D_i, U_i' \cap A_i^* \setminus D_i) + \sum_{j \neq i} w(U_i', U_j') \\
&= w(D_i, A \setminus A_i^*) + w(D_i, U_i^*) + \sum_{j \neq i} w(U_i', U_j'), \qquad (6.1)
\end{aligned}$$

where the last equation holds as $U_i' \cap A_i^* \setminus D_i = U_i^*$. and

$$\begin{aligned}
w(D_i, A \setminus U) &\geq w(D_i, A_i^* \setminus U) \qquad (6.2) \\
&= w(D_i, A_i^* \setminus \cup_j U_j') \qquad \text{since } A_i^* \subseteq A^\dagger \\
&= w(D_i, (A_i^* \setminus U_i') \setminus \cup_{j \neq i} U_j') \\
&\geq w(D_i, A_i^* \setminus U_i') - \sum_{j \neq i} w(D_i, U_j') \\
&\geq w(D_i, A_i^* \setminus U_i') - \sum_{j \neq i} w(U_i', U_j'). \qquad (6.3)
\end{aligned}$$

To avoid clutter, define $D = \cup_i D_i$. Note that $U^* = U \setminus D$. By examining the difference in edges, observe that $\partial_{G[A]}(U^*) - \partial_{G[A]} U = w(D, U^*) - w(D, A \setminus U)$. Using Equations (6.1) and (6.3), we obtain

$$\begin{aligned}
\partial_{G[A]} U^* - \partial_{G[A]} U &= \partial_{G[A]}(U^*) - \partial_{G[A]} U \\
&= w(D, U^*) - w(D, A \setminus U) \\
&= \sum_{i \in [k]} \big(w((D_i, U^*) - w((D_i, A \setminus U) \big) \\
&\overset{(6.1),(6.3)}{\leq} \sum_{i \in [k]} \Bigg(\big(w(D_i, A \setminus A_i^*) + w(D_i, U_i^*) + \sum_{j \neq i} w(U_i', U_j') \big) \\
&\qquad - \big(w(D_i, A_i^* \setminus U_i') - \sum_{j \neq i} w(U_i', U_j') \big) \Bigg) \\
&= \sum_{i \in [k]} \big(w(D_i, A \setminus A_i^*) + w(D_i, U_i^*) - w(D_i, A_i^* \setminus U_i') \big) \qquad (6.4) \\
&\quad + 2 \sum_{\substack{i, j \in [k] \\ i \neq j}} w(U_i', U_j').
\end{aligned}$$

In the following note that $w(A_i^* \setminus U_i^*, U_i^*) = \partial_{G[A_i^*]} U_i^*$ and $w(U_i' \cap A_i^*, A_i^* \setminus U_i') = \partial_{G[A_i^*]}(U_i' \cap A_i^*)$. For the expression $w(D_i, A \setminus A_i^*) + w(D_i, U_i^*) - w(D_i, A_i^* \setminus U_i')$, we add $w(U_i^*, A_i^* \setminus U_i')$ to the second and third terms so that they become

$$\begin{aligned}
&w(D_i, A \setminus A_i^*) + w(D_i \cup (A_i^* \setminus U_i'), U_i^*) - w(D_i \cup U_i^*, A_i^* \setminus U_i')\\
&= w(D_i, A \setminus A_i^*) + w(A_i^* \setminus U_i^*, U_i^*) - w(U_i' \cap A_i^*, A_i^* \setminus U_i')\\
&= w(D_i, A \setminus A_i^*) + w(A_i^* \setminus U_i^*, U_i^*) - w(U_i' \cap A_i^*, A_i^* \setminus U_i')\\
&= w(D_i, A \setminus A_i^*) + \partial_{G[A_i^*]} U_i^* - \partial_{G[A_i^*]}(U_i' \cap A_i^*)\\
&= w(D_i, V \setminus A_i^*) - w(D_i, V \setminus A) + \partial_{G[A_i^*]} U_i^* - \partial_{G[A_i^*]}(U_i' \cap A_i^*)\\
&\le (1+\epsilon/2)\partial_{G[A_i^*]}(U_i' \cap A_i^*) - \partial_{G[A_i^*]} U_i^* - w(D_i, V \setminus A) + \partial_{G[A_i^*]} U_i^* - \partial_{G[A_i^*]}(U_i' \cap A_i^*)\\
&= \epsilon/2 \partial_{G[A_i^*]}(U_i' \cap A_i^*) - w(D_i, V \setminus A),
\end{aligned}$$

where the last inequality follow from Lemma 6.14.

Continuing from Equation (6.4), we obtain the following bound. Notat that the second inequality below comes from the fact that U_j' can be larger than $A_i^* \cap U_j'$ in general.

$$\begin{aligned}
\partial_{G[A]} U^* - \partial_{G[A]} U &\le \sum_{i \in [k]} \big(\epsilon/2 \cdot \partial_{G[A_i^*]}(U_i' \cap A_i^*) - w(D_i, V \setminus A)\big) + 2 \sum_{i,j : i \ne j} w(U_i', U_j')\\
&= \big(\sum_{i \in [k]} \epsilon/2 \cdot \partial_{G[A_i^*]}(U_i' \cap A_i^*)\big) - w(D, V \setminus A) + 2 \sum_{i,j : i \ne j} w(U_i', U_j')\\
&= \sum_{i \in [k]} \epsilon/2 \cdot w(U_i' \cap A_i^*, A_i^* \setminus U_i') - w(D, V \setminus A) + 2 \sum_{i,j : i \ne j} w(U_i', U_j')\\
&\le \sum_{i \in [k]} \epsilon/2 \Big(w(U_i' \cap A_i^*, A_i^* \setminus U) + \sum_{j \ne i} w(U_i' \cap A_i^*, U_j') \Big) - w(D, V \setminus A)\\
&\quad + 2 \sum_{i,j : i \ne j} w(U_i', U_j')\\
&\le \sum_{i \in [k]} \epsilon/2 \cdot w(U_i' \cap A_i^*, A_i^* \setminus U) - w(D, V \setminus A) + (2 + \epsilon/2) \sum_{i,j : i \ne j} w(U_i', U_j'). \qquad (6.5)
\end{aligned}$$

We bound $\sum_{i \in [k]} \epsilon/2 \cdot w(U_i' \cap A_i^*, A_i^* \setminus U)$ as

$$\begin{aligned}
\sum_{i \in [k]} \epsilon/2 \cdot w(U_i' \cap A_i^*, A_i^* \setminus U) \le \sum_{i \in [k]} \epsilon/2 \cdot w(U_i' \cap A^\dagger, A^\dagger \setminus U) &= \epsilon/2 \cdot w(\cup_i (U_i' \cap A^\dagger), A^\dagger \setminus U)\\
&= \epsilon/2 \cdot w(U \cap A^\dagger, A^\dagger \setminus U)\\
&= \epsilon/2 \cdot \partial_{G[A^\dagger]}(U \cap A^\dagger)\\
&\le \epsilon/2 \cdot \partial_{G[A]} U. \qquad (6.6)
\end{aligned}$$

To bound $\sum_{i,j : i \ne j} w(U_i', U_j')$, recall that the construction of $\mathcal{P} = \{U_1', \ldots, U_k'\}$ was formed by at most $|U \cap A^\dagger| \le |U| \le s_0/\lambda \le 2s_0/\tilde{\lambda}$ many cuts, each of weight less than $\gamma\tilde{\lambda} = \frac{\epsilon\tilde{\lambda}}{100 s_0}\tilde{\lambda}$, so $\sum_{i,j : i \ne j} w(U_i', U_j') = w(U_1', \ldots, U_k') \le 2s_0/\tilde{\lambda} \cdot \gamma\tilde{\lambda} \le 0.02\epsilon\tilde{\lambda}$. Continuing from Equation (6.5), we obtain

$$\partial_{G[A]} U^* - \partial_{G[A]} U \le \epsilon/2 \cdot \partial_{G[A]} U - w(D, V \setminus A) + 0.05\epsilon\tilde{\lambda}.$$

Note that we need to show that

$$\partial_{G[A]} U^* - \partial_{G[A]} U \le \epsilon \partial_{G[A]} U - w(D, V \setminus A).$$

If $\partial_{G[A]} U \ge 0.1\lambda$ then property (3) follows, since $D = U \setminus U^*$ and $\epsilon/2 \cdot \partial_{G[A]} U + 0.05\epsilon\tilde{\lambda} \le \epsilon \partial_{G[A]} U$.

If $\partial_{G[A]} U < 0.1\lambda$ then we actually claim that $U^* = U$. If not, then there exists $i \in [k]$ such that $U_i^* \ne U_i' \cap A_i^*$. By Lemma 6.14, we must have $\partial_{G[A_i^*]}(U_i' \cap A_i^*) \ge 0.2\lambda$. Repeating the chain of inequalities from Equation (6.6) for the last inequality below,

$$0.2\lambda \le \partial_{G[A_i^*]}(U_i' \cap A_i^*) = w(U_i' \cap A_i^*, A_i^* \setminus U_i') \le w(U_i' \cap A_i^*, A_i^* \setminus U) + \sum_{j \ne i} w(U_i' \cap A_i^*, U_j')$$

$$\le w(U_i' \cap A_i^*, A_i^* \setminus U) + 0.02\epsilon\tilde{\lambda} \overset{(6.6)}{\le} \partial_{G[A]}U + 0.02\epsilon\tilde{\lambda},$$

contradicting the assumption that $\partial_{G[A]}U < 0.1\lambda$. It follows that $U^* = U$. Note that $U^* \cap A_0 = \emptyset$ follows by construction and $\partial_{G[A]}U^* - \partial_{G[A]}U \le \epsilon\partial_{G[A]}U - w(D, V \setminus A)$ hold trivially. Thus property (3) is fulfilled.

Therefore, we have ensured property (3) follows. This concludes the proof of Lemma 6.4.

6.2.3 Deferred Proofs

Proof. [Proof of Lemma 6.7] The algorithm is a greedy minimum spanning tree packing algorithm. Initialize each edge length ℓ_e in $H[U]$ to be 1. For iteration i from 1 to $\lceil \kappa \ln m \rceil$, compute a minimum spanning tree T_i with respect to lengths ℓ_e, and then remove all edges in T_i of length more than e^5m^6. Add the resulting forest F_i to the collection and then increase the length ℓ_e of each edge e in F_i by factor 1.1.

We first prove property (1). For each edge e in H, its length ℓ_e increases by factor 1.1 every time e joins a forest F_i, so the edge participates in at most $\log_{1.1} e^5m^6 \le 100 \ln m$ forests for m large enough, as promised.

To prove property (2), fix U such that $H[U]$ has minimum cut at least κ. It suffices to show that over $\lceil \kappa \ln m \rceil$ iterations of the algorithm, no edge in $H[U]$ has length ℓ_e more than e^5m^6. In that case, each minimum spanning tree T_i must connect the vertices in U with edges of length $\ell_e \le e^5m^6$, and these edges are not deleted in F_i, i.e., all nodes of u are connected in each forest F_i.

To show this claim, we interpret the algorithm in the fractional packing setting of Young [You02] and follow the proof of Lemma 6.2 in [You02]. Consider an induced subgraph $H[U]$ with minimum cut at least κ. For each i from 1 to $\lceil \kappa \ln m \rceil$, consider the forest $T_i \cap H[U]$. It is not hard to see, by properties of the minimum spanning tree, that $T_i \cap H[U]$ is a subset of some minimum spanning tree T_i' of $H[U]$ with respect to the current lengths ℓ. In other words, on each iteration, within the scope of $H[U]$, the algorithm increases by factor 1.1 the lengths of a subset of the edges of a minimum spanning tree with the current lengths ℓ. We show that over $\lceil \kappa \ln m \rceil$ iterations, no edge in $H[U]$ can have length more than e^5m^6.

Define $P \subseteq \mathbb{R}^{E(H[U])}$ as the spanning tree polytope on $H[U]$, i.e., the convex hull of the indicator vectors of the spanning trees of $H[U]$. Define $\Delta \subseteq \mathbb{R}^{E(H[U])}$ as the simplex on $E(H[U])$, i.e., the convex hull of the unit vectors. By duality, we have

$$\min_{x \in P} \max_{y \in \Delta} \langle x, y \rangle = \max_{y \in \Delta} \min_{x \in P} \langle x, y \rangle = \max_{\substack{y \ge \mathbf{0} \\ y \ne \mathbf{0}}} \frac{\min_{x \in P} \sum_e x_e y_e}{\sum_e y_e}.$$

By a theorem of Nash-Williams [NW61], there is an (integral) spanning tree packing of any graph of value at least half the minimum cut of the graph. By scaling down the packing and the fact that $H[U]$ has minimum cut at least κ, we obtain

$$\min_{x \in P} \max_{y \in \Delta} \langle x, y \rangle \le \frac{1}{\kappa/2} = \frac{2}{\kappa}.$$

For each iteration i, by construction, the quantity $\sum_{e \in H[U]} \ell_e$ increases by factor $\sum_{e \in T_i \cap H[U]} 0.1\ell_e$. Since $T_i \cap H[U] \subseteq T_i'$, this is at most $\sum_{e \in T_i'} 0.1\ell_e$. Since the spanning tree polytope P is integral, we have $\sum_{e \in T_i'} \ell_e = \min_{x \in P} \sum_e x_e \ell_e$. Putting everything together, the increase in $\sum_{e \in H[U]} \ell_e$ is

$$\sum_{e \in T_i \cap H[U]} 0.1\ell_e \le \sum_{e \in T_i'} 0.1\ell_e = 0.1 \min_{x \in P} \sum_{e \in H[U]} x_e \ell_e \le 0.1 \cdot (\min_{x \in P} \max_{y \in \Delta} \langle x, y \rangle) \sum_{e \in H[U]} \ell_e \le 0.1 \cdot \frac{2}{\kappa} \sum_{e \in H[U]} \ell_e.$$

After $\lceil \kappa \ln m \rceil$ iterations, the quantity $\sum_{e \in H[U]} \ell_e$ is at most

$$m \cdot \left(1 + \frac{5}{\kappa}\right)^{\lceil \kappa \ln m \rceil} \le m \cdot e^{(5/\kappa)(\kappa \ln m + 1)} \le m \cdot e^{5 \ln m + 5} = e^5m^6.$$

It follows that $\ell_e \le \sum_{e \in H[U]} \ell_e \le e^5m^6$ for each edge $e \in H[U]$, as claimed.

□

Proof. [Proof of Lemma 6.8] First, remove from T all vertices $v \in V(H) = A$ with $d_G^W(v) > s_0$. Then, consider each connected component (tree) T' in the remaining forest. If $\mathbf{vol}_G^W(V(T')) \le 3s_0$, we add $V(T')$ to the output and proceed to the next connected component. If $\mathbf{vol}_G^W(V(T')) > 3s_0$, we will partition the edges of T' into edge-disjoint trees $T_1, \dots, T_\ell$ such that $\mathbf{vol}^W(V(T_i)) \in (s_0, 3s_0]$, and then add the sets $V(T_i)$ to the output.

We now describe the partitioning process. Root the tree T' at an arbitrary vertex and while the remaining tree has (vertex) volume more than $3s_0$, do the following. Find the lowest (i.e., farthest from the root) node v whose subtree rooted at v has volume more than s_0. By construction, the subtrees rooted at the children of v have volume at most s_0. We now have two cases.

1. If the subtrees rooted at the children of v have total volume at most $2s_0$, then since $d_G^W(v) \le s_0$, the total volume of the subtree rooted at v is at most $3s_0$, and we declare the subtree rooted at v as one of the trees T_i.

2. Otherwise, take a subset of these subtrees whose total volume is in $(s_0, 2s_0]$. Connect v to these subtrees, declare the result as one of the output trees T_i.

In both cases, we have $\mathbf{vol}^W(V(T_i)) \in (s_0, 3s_0]$ by construction. We then remove T_i from T' (as well as the newly isolated vertices) and proceed to the next iteration. Each iteration decreases $\mathbf{vol}^W(V(T'))$ by at most $2s_0$ until $\mathbf{vol}^W(V(T_i)) \le 3s_0$, which means $\mathbf{vol}^W(V(T_i)) > s_0$ and we finish by taking T' as the final T_i.

Property (1) holds by construction. For property (2), since the trees T_i are edge-disjoint, each vertex $v \in A$ can belong to at most $d_{T'}(v) \le_T (v)$ many sets $V(T_i)$. For the rest of the proof, we focus on property (3). Since $\mathbf{vol}^W(U) \le s_0$, we cannot remove any vertices from U on the very first step, so U is contained in a connected component T'. If $\mathbf{vol}_G^W(V(T')) \le 3s_0$, then the entire $V(T')$ is added and property (3) holds. Otherwise, the edges of T' are partitioned into edge-disjoint trees $T_1, \dots, T_\ell$. Since $\mathbf{vol}^W(V(T_i)) > s_0$ for all i, we cannot have $V(T_i) \subseteq U$ for any i. So any $V(T_i)$ that intersects U must contain an edge with exactly one endpoint in U. Since there are k such edges, there are at most k many $V(T_i)$ that intersect U, concluding property (3).

This algorithm can be easily implemented in $\tilde{O}(|V(T)|)$ time using dynamic tree data structures that update the volume of subtrees after each vertex deletion. ∎

Proof. [Proof of Lemma 6.12.] We use the following *approximate isolating cuts* algorithm of [LNPS23]. The theorem below resembles Theorem 6.2 of [LNPS23] with an additional property that follows directly from the algorithm itself. We also relabel a few of the parameters to avoid overloading variables.

THEOREM 6.2. *Fix any $\epsilon' < 1$. Given an undirected graph $H = (V, E)$ and a set of terminal vertices $T \subseteq V$, there is an algorithm that outputs, for each $t \in T$, a $(1+\epsilon')$-approximate minimum cut V_t separating t from $T \setminus t$, i.e., $V_t \cap T = \{t\}$. Moreover, the sets V_t are disjoint, and for each $t \in T$ and a set U separating t from $T \setminus t$, there exists $U' \subseteq U \cap V_t$ such that $\partial U' \le (1+\epsilon')\partial U$. Finally, for each $t \in T$, there exists a feasible flow such that*

1. *The flow originates at t and is not absorbed by any nodes in V_t. More precisely, vertex t has positive source and zero sink, and each vertex $v \in V_t \setminus t$ has zero source and zero sink.*

2. *Each edge $e \in E(V_t, V \setminus V_t)$ sends at least $(1-\epsilon')w(e)$ flow in the direction from V_t to $V \setminus V_t$.*

Note that in $V \setminus V_t$, the flow can do anything as long as it remains feasible.

Let ϕ be the conductance *of H, defined as* $\min_{\emptyset \subsetneq S \subsetneq V} \partial S / \min\{\mathbf{vol}^W(S), \mathbf{vol}^W(\bar{S})\}$. *Then, the algorithm takes deterministic time $O(m/(\epsilon\phi)^{O(1)})$.*

Proof. [Proof (Sketch)] We refer to Algorithm 1 in Section 6 of [LNPS23]. The algorithm begins with a collection of $O(\log |T|)$ bipartitions of T such that every pair of terminals in T is separated by at least one bipartition. For each bipartition X_i, Y_i of T, the algorithm computes a $(1 + \Theta(\epsilon/\log |T|))$-*fair cut* S_i between X_i and Y_i. The definition of fair cuts from [LNPS23] is rather technical so we skip it for simplicity. Instead, we focus on its key *approximate uncrossing* property formalized by Lemma 1.2 of [LNPS23]. Let V_t^* be the minimum cut separating t from $T \setminus t$, fix an index $i \le O(\log |T|)$, and suppose without loss of generality that S_i is the side of the (X_i, Y_i)-cut containing t. We can "uncross" V_t^* with the cut S_i so that the new cut $V_t^* \cap S_i$ only increases by factor $(1+\Theta(\epsilon/\log |T|))$ compared to V_t^*. We uncross V_t^* iteratively with each S_i so that the final cut $V_t^* \cap (\cap_i S_i)$ increases by a factor $(1 + \Theta(\epsilon/\log |T|))^{O(\log |T|)} \le 1 + O(\epsilon)$ compared to V_t^*.

The rest of the algorithm looks for $V_t^* \cap (\cap_i S_i)$ instead of V_t^*. After computing the $O(\log|T|)$ cuts, the algorithm removes all cut edges from the graph, isolating the terminals from each other. For each terminal $t \in T$, the algorithm considers the connected component containing t, contracts the graph outside this component, and makes one final $(1+\Theta(\epsilon/\log|T|))$-fair cut computation to obtain V_t, which approximates $V_t^* \cap (\cap_i S_i)$. The existence of a feasible flow satisfying properties (1) and (2) follows from the definition of fair cuts; we omit the details.

For the running time, we note that the fair cuts algorithm in [LNPS23] runs in $\tilde{O}(m/\epsilon^{O(1)})$ time regardless of ϕ, but their algorithm is randomized. Fortunately, as remarked right before Section 1.2 of [LNPS23], the only randomized component is building a *congestion approximator* [She13] of the graph. If G has conductance ϕ, then there is a trivial congestion approximator (the singleton vertices) with "quality" $1/\phi$, so there is a fast, deterministic construction. The final running time picks up a $1/\phi^{O(1)}$ factor due to the quality. We omit the details, which are standard in the literature. $\square$

We construct $H = (V_H, E_H)$ as follows. Start with $G[A']$ and add a vertex t^* connected to each vertex $v \in A'$ by an edge of weight $\frac{\epsilon\tilde{\lambda}}{10s_0} d_G^W(v)$. Next, for each $i \in [k]$, add a vertex t_i connected to each vertex $v \in S_i$ by an edge of weight $(1-2\epsilon)w_G(V \setminus A', v)$. Call the resulting graph H, and let $T = \{t_1, \ldots, t_k, t^*\}$.

We first bound the conductance of the graph. Consider a subset $S \subseteq V_H$, and assume without loss of generality that $t^* \notin S$. Then, $\partial_H S \geq w_H(S, t^*) = \frac{\epsilon\tilde{\lambda}}{10s_0}\mathbf{vol}_G^W(S \setminus T)$. Since $t^* \notin S$, we have $\mathbf{vol}_H^W(S) = \Theta(\mathbf{vol}_G^W(S \setminus T))$ by construction. Therefore, $\partial_H S \geq \Omega(\frac{\epsilon\tilde{\lambda}}{s_0}\mathbf{vol}_H^W(S)) \geq \Omega(\frac{\epsilon\tilde{\lambda}}{s_0}) \cdot \min\{\mathbf{vol}_H^W(S), \mathbf{vol}_H^W(\bar{S})\}$, and H has conductance $\Omega(\frac{\epsilon\tilde{\lambda}}{s_0})$.

We run Theorem 6.2 on H and T with parameter $\epsilon' \leftarrow \min\{\epsilon, c \cdot \tilde{\lambda}/s_0\}$ for sufficiently small constant $c > 0$, and let $V_{t_1}, \ldots, V_{t_k}, V_{t^*}$ be the returned set. By the conductance bound, the algorithm takes time $\tilde{O}((\frac{\epsilon\tilde{\lambda}}{s_0})^{O(1)}|E(G[A'])|)$. We define $C_i = V_{t_i} \setminus \{t_i\}$ for each $i \in [k]$. (We ignore the set V_{t^*}.)

We now show the properties of Lemma 6.12 in order.

1. The singleton cut $\{t_i\}$ separates t_i from $T\setminus\{t_i\}$, and since C_i is a $(1+\epsilon)$-approximate minimum cut separating t_i from $T \setminus \{t_i\}$, we have $\partial_H C_i \leq (1+\epsilon)\partial_H\{t_i\} = (1+\epsilon)(1-2\epsilon)w_G(V \setminus A', S_i) \leq (1-\epsilon)w_G(V \setminus A', S_i)$. Also, since $t^* \in T \setminus \{t_i\}$, we have $t^* \notin C_i$, so $\partial_H C_i \geq w_H(C_i, t^*) = \frac{\epsilon\tilde{\lambda}}{10s_0}\mathbf{vol}_G^W(C_i)$. Combining these two inequalities, we obtain $\frac{\epsilon\tilde{\lambda}}{10s_0}\mathbf{vol}_G^W(C_i) \leq \partial_H C_i \leq (1-\epsilon)w_G(V \setminus A', S_i) \leq (1-\epsilon)\mathbf{vol}_G^W(S_i)$, which proves property (1).

2. By property (2) of Theorem 6.2, there is at least $(1-\epsilon')w_H(V_{t_i}, V_H \setminus V_{t_i})$ flow going out of V_{t_i}. By property (1), this flow can only originate from t_i. Since there is at most $w_H(t_i, V_{t_i} \setminus \{t_i\})$ total capacity from t_i into $V_{t_i} \setminus \{t_i\}$, we must have $w_H(t_i, V_{t_i} \setminus \{t_i\}) \geq (1-\epsilon')w_H(V_{t_i}, V_H \setminus V_{t_i})$. We now relate these quantities to those involving graph G. By construction, $w_H(t_i, V_{t_i} \setminus \{t_i\}) = w_H(t_i, C_i) = (1-2\epsilon)w_G(V \setminus A', C_i \cap S_i)$. Also, $\partial_{G[A']}C_i = w_H(V_{t_i} \setminus \{t_i\}, V_H \setminus (V_{t_i} \cup T))$ since no additional edges in H are added between $V_{t_i} \setminus \{t_i\}$ and $V_H \setminus (V_{t_i} \cup T)$. Therefore,
$$\begin{aligned}\partial_{G[A']}C_i &= w_H(V_{t_i} \setminus \{t_i\}, V_H \setminus (V_{t_i} \cup T)) \\ &\leq w_H(V_{t_i}, V_H \setminus V_{t_i}) \\ &\leq \frac{1}{1-\epsilon'}w_H(t_i, V_{t_i} \setminus \{t_i\}) \\ &= \frac{1}{1-\epsilon'} \cdot (1-2\epsilon)w_G(V \setminus A', C_i \cap S_i) \\ &\leq \frac{1}{1-\epsilon'} \cdot (1-2\epsilon)w_G(V \setminus A', C_i) \\ &\leq (1-\epsilon)w_G(V \setminus A', C_i),\end{aligned}$$
where the last inequality uses $\epsilon' \leq \epsilon$. This concludes property (2).

3. Consider a set $U^\dagger \subseteq S_i$ with $\partial_{G[A']}U^\dagger \leq 1.01\tilde{\lambda}$ and $\mathbf{vol}_G^W(U^\dagger) \leq s_0$. Let $U = U^\dagger \cup \{t_i\}$. Since $U \subseteq S_i$ and

S_i is disjoint from S_j for any $j \neq i$, there are no edges from U to t_j, so

$$\partial_H U = E_H(U, A' \setminus U) + E_H(t_i, S_i \setminus U) + E_H(U, t^*) = \partial_{G[A']} U^\dagger + (1-2\epsilon) w_G(V \setminus A', S_i \setminus U^\dagger) + \frac{\epsilon \tilde{\lambda}}{10 s_0} \mathbf{vol}_G^W(U^\dagger).$$

Theorem 6.2 guarantees a set $U' \subseteq U \cap V_t$ such that $\partial_H U' \leq (1+\epsilon')\partial_H U$. Let $U^* = U' \setminus \{t_i\}$, so that we similarly have $\partial_H U' = \partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', S_i \setminus U^*) + \frac{\epsilon \tilde{\lambda}}{10 s_0} \mathbf{vol}_G^W(U^*)$. It follows that

$$\begin{aligned}
(6.7) \qquad & \partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', S_i \setminus U^*) \\
&\leq \partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', S_i \setminus U^*) + \frac{\epsilon \tilde{\lambda}}{10 s_0} \mathbf{vol}_G^W(U^*) \\
&= \partial_H U' \\
&\leq (1+\epsilon')\partial_H U \\
&= (1+\epsilon')\big(\partial_{G[A']} U^\dagger + (1-2\epsilon) w_G(V \setminus A', S_i \setminus U^\dagger) + \frac{\epsilon \tilde{\lambda}}{10 s_0} \mathbf{vol}_G^W(U^\dagger)\big).
\end{aligned}$$

From $U^* \subseteq U^\dagger \subseteq S_i$ we obtain that

$$w_G(V \setminus A', S_i \setminus U^*) - w_G(V \setminus A', S_i \setminus U^\dagger) = w_G(V \setminus A', U^\dagger \setminus U^*),$$

so subtracting $(1-2\epsilon) w_G(V \setminus A', S_i \setminus U^\dagger)$ from both sides gives

$$\begin{aligned}
(6.8) \qquad & \partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', U^\dagger \setminus U^*) \\
&\leq (1+\epsilon')\partial_{G[A']} U^\dagger + \epsilon'(1-2\epsilon) w_G(V \setminus A', S_i \setminus U^\dagger) + \frac{(1+\epsilon')\epsilon \tilde{\lambda}}{10 s_0} \mathbf{vol}_G^W(U^\dagger) \\
&\leq (1+\epsilon')\partial_{G[A']} U^\dagger + \epsilon'(1-2\epsilon) \mathbf{vol}_G^W(S_i \setminus U^\dagger) + \frac{(1+\epsilon')\epsilon \tilde{\lambda}}{10 s_0} \mathbf{vol}_G^W(U^\dagger) \\
&\leq (1+\epsilon')\partial_{G[A']} U^\dagger + \epsilon' \cdot O(s_0) + \frac{(1+\epsilon')\epsilon \tilde{\lambda}}{10 s_0} \cdot s_0.
\end{aligned}$$

As long as $\epsilon' \leq c \cdot \tilde{\lambda}/s_0$ for sufficiently small constant $c > 0$, we obtain

$$(6.9) \qquad \partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', U^\dagger \setminus U^*) \leq (1+\epsilon')\partial_{G[A']} U^\dagger + \epsilon \tilde{\lambda}/2.$$

If $\partial_{G[A']} U^\dagger \geq 0.2\lambda$, then we obtain $\partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', U^\dagger \setminus U^*) \leq (1+O(\epsilon))\partial_{G[A']} U^\dagger$, and dividing by $(1-2\epsilon)$ fulfills property (3) with ϵ replaced by $O(\epsilon)$. Otherwise, if $\partial_{G[A']} U^\dagger < 0.2\lambda$, then we need to show that $U^* = U^\dagger$. Suppose for contradiction that $U^* \subsetneq U^\dagger$. From Equation (6.9) and recalling that $\epsilon' \leq \epsilon \leq 0.1$ and $\tilde{\lambda} \leq 1.01\lambda$, we obtain

$$(6.10) \qquad \partial_{G[A']} U^* + (1-2\epsilon) w_G(V \setminus A', U^\dagger \setminus U^*) = (1+\epsilon')\partial_{G[A']} U^\dagger + \epsilon \tilde{\lambda}/2 < 0.3\lambda.$$

Since $U^\dagger \setminus U^*$ is non-empty by assumption,

$$\begin{aligned}
\lambda \leq \partial_G(U^\dagger \setminus U^*) &= \partial_{G[A']}(U^\dagger \setminus U^*) + w_G(V \setminus A', U^\dagger \setminus U^*) \\
&\leq \partial_{G[A']} U^\dagger + \partial_{G[A']} U^* + w_G(V \setminus A', U^\dagger \setminus U^*) \\
&\overset{(6.10)}{\leq} \partial_{G[A']} U^\dagger + 0.3\lambda + w_G(V \setminus A', U^\dagger \setminus U^*) \\
&\leq 0.5\lambda + w_G(V \setminus A', U^\dagger \setminus U^*).
\end{aligned}$$

It follows that $w_G(V \setminus A', U^\dagger \setminus U^*) \geq 0.5\lambda$, which contradicts Equation (6.10) for $\epsilon \leq 0.1$. Finally, to satisfy property (3) with factor $(1+\epsilon)$ instead of $(1+O(\epsilon))$, we can simply reset ϵ to be a constant factor smaller.

With all three properties established, this concludes the proof. ∎

7 Sparsifier construction

In this section, we outline the construction of the skeleton graph using the iterative graph decomposition from the previous section. Similarly to [Li21], we separate the construction into an "unbalanced" case and a "balanced" case.

At a high level, the algorithm derandomizes the sampling procedure where edges are sampled independently with probability proportional to their weights. The challenge is in establishing an efficient "union bound" over all cuts, which are exponential in number. To cope with this issue, we exploit the structure of graph cuts guaranteed by Lemma 3.2. Still, the derandomization is only sufficient to preserve "unbalanced" cuts, which includes (near-) minimum cuts. To address the remaining "balanced" cuts, we construct a separate graph that forces these cuts to be large enough.

7.1 Unbalanced sparsifier In this section, we present our construction of the unbalanced sparsifier H, which is an unweighted graph supported on a subset of edges in G. To establish the properties we want, we require that a certain list of quantities are approximately preserved. To formally describe these properties, we introduce the graph Laplacian for an algebraic representation of the cuts of a graph.

For a given set $S \subseteq V$, let $\mathbb{1}_S \in \{0,1\}^V$ be the vector with value 1 at vertex v if $v \in S$, and value 0 otherwise. Define the *Laplacian* L_G of a weighted graph G as the $|V| \times |V|$ matrix indexed by $V \times V$, where each diagonal entry (v,v) is $d^W(v)$ and each off-diagonal entry (u,v) is $-w(u,v)$ if $(u,v) \in E$ and 0 otherwise. It is well-known that the quadratic form $\mathbb{1}_S^T L_G \mathbb{1}_S$ is equal to $\partial_G S$.

Let L be the total number of iterations of Step 2b until G_L is a single vertex. For each iteration j and each vertex $v \in V_j$, let $\overline{v} \subseteq V$ denote its "pre-image under the contraction map", defined as all original vertices in V that get contracted into v in graph G_j. For a set $S \subseteq V_j$, let $\overline{S} = \bigcup_{v \in D} \overline{v}$ be the pre-image of S.

The unbalanced sparsifier H is an unweighted graph supported on a subset of the edges of G. Let $\nu > 0$ and $\eta > 0$ be parameters. Our goal is to compute H and an associated weight W so that for all iterations j,k and vertices $u \in V_j, v \in V_k$ with $d^W_{V_j}(u) \le \nu$ and $d^W_{V_k}(v) \le \nu$, the quantity $\mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}}$ is preserved to an additive error of $\eta\lambda$: $|W \cdot \mathbb{1}_{\overline{u}}^T L_H \mathbb{1}_{\overline{v}} - \mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}}| \le \eta\lambda$.

We first observe that if $d^W_{V_j}(u) \le \nu$ and $d^W_{V_k}(v) \le \nu$, then the total weight of edges that affect the expression $\mathbb{1}_{\overline{u}}^T L_H \mathbb{1}_{\overline{v}}$ is at most ν. Assume without loss of generality that $j \le k$; then, either $\overline{u} \cap \overline{v} = \emptyset$ or $\overline{v} \subseteq \overline{u}$. If the former is true, then $\mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}} = -w(\overline{u},\overline{v})$ only depends on edges with exactly one endpoint in $\overline{u}$, which is at most $\partial_G(\overline{u}) = d^W_{V_j}(u)$ total weight of edges. If the latter, then $\mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}} = \mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{u}} - \mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v} \setminus \overline{u}} = \partial_G(\overline{u}) - w(\overline{u}, \overline{v} \setminus \overline{u})$ also only depends on edges with exactly one endpoint in $\overline{u}$.

Therefore, a simple random sampling procedure gives sufficient concentration. Namely, sampling each edge $e \in E$ with probability proportional to $w(e)$ and assigning the same weight W to every sampled edge preserves the $\eta\lambda$ additive error with high probability by Chernoff bounds. Also, note that $\mathbb{1}_{\overline{u}}^T L_H \mathbb{1}_{\overline{v}}$ is only nonzero when there is an edge in G between $\overline{u}$ and $\overline{v}$. In fact, for each j,k, there are $O(1)$ choices of $u \in V_j$ and $v \in V_k$ for which the edge participates in $\mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}}$: the endpoints of the edge must be contained in $\overline{u}$ and $\overline{v}$. So the total "representation size" of the required constraints is $O(L^2|E|)$, excluding the constraints for which the quantities are zero, which are vacuously satisfied. Since the total representation size of the constraints is only $O(L^2|E|)$, we can derandomize the above sampling procedure with the method of pessimistic estimators, following the approach from Section 3.2.1 of [Li21]. We omit the technical details since the setup is nearly identical and is standard in the literature. The final result can be stated as follows.

Lemma 7.1. (Lemma 3.8 of [Li21]) *For any parameters $\nu, \eta > 0$, there is an algorithm with running time $\tilde{O}(L^2 m)$ that computes an unweighted graph H and a weight $W \le O(\frac{\eta\lambda^2}{\nu \ln(Lm)})$ such that for all iterations j,k and vertices $u \in V_j, v \in V_k$ with $d^W_{V_j}(u) \le \nu$ and $d^W_{V_k}(v) \le \nu$, we have $|W \cdot \mathbb{1}_{\overline{u}}^T L_H \mathbb{1}_{\overline{v}} - \mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}}| \le \eta\lambda$.*

We now prove the key property of the unbalanced sparsifier H and associated weight W. Intuitively, the lemma below says that if a set $S \subseteq V$ can be represented efficiently by the set difference of sets of the form $\overline{v}$ with small total degree $\sum_v d^W(v)$, then the cut ∂S is approximately preserved by H. Here, the degree $d^W(v)$ of vertex $v \in \cup_j V_j$ equals $d^W_{V_j}(v)$ for the iteration j with $v \in V_j$.

Lemma 7.2. *For any cut $S \subseteq V$ in G, suppose that there exists a set $D \subseteq \cup_j V_j$ such that $S = \triangle_{v \in D} \overline{v}$ and $d = \sum_{v \in D} d^W(v) \le \nu$. Then, $|W \cdot \partial_H S - \partial_G S| \le (d/\lambda)^2 \eta\lambda$.*

Proof. Since the sets $\overline{v}$ for $v \in \cup_j V_j$ form a laminar family, we can translate the set difference into algebraic form $\mathbb{1}_S = \sum_{v\in D} \pm \mathbb{1}_{\overline{v}}$ by traversing through the sets V_j in decreasing order of j and adding or subtracting vectors $\mathbb{1}_{\overline{v}}$ as necessary. We now write

$$\begin{aligned} W \cdot \partial_H S - \partial_G S &= W \cdot \mathbb{1}_S^T L_H \mathbb{1}_S - \mathbb{1}_S^T L_G \mathbb{1}_S \\ &= \mathbb{1}_S^T (W \cdot L_H - L_G) \mathbb{1}_S \\ &= \bigg(\sum_{v\in D} \pm \mathbb{1}_{\overline{v}} \bigg)^T (W \cdot L_H - L_G) \bigg(\sum_{v\in D} \pm \mathbb{1}_{\overline{v}} \bigg). \end{aligned}$$

Expanding the summation gives $|D|^2$ terms of the form $\mathbb{1}_{\overline{u}}^T (W \cdot L_H - L_G) \mathbb{1}_{\overline{v}}$, each of which is approximated to additive error $\eta\lambda$ by Lemma 7.1 since $d \le \nu$. It remains to bound the number of terms. Each $v \in D$ has $d^W(v) = \partial_G \overline{v} \ge \lambda$ unless $\overline{v} = V$, in which case $\mathbb{1}_{\overline{u}}^T (W \cdot L_H - L_G) \mathbb{1}_{\overline{v}} = \mathbb{1}_{\overline{u}}^T (W \cdot L_H - L_G) \mathbb{1}_V = 0$ for any u. So there are at most d/λ many vertices, and the additive error of $W \cdot \mathbb{1}_S^T L_H \mathbb{1}_S - \mathbb{1}_S^T L_G \mathbb{1}_S$ is at most $(d/\lambda)^2 \eta\lambda$, as promised. ∎

We now show that in particular, the minimum cut can be efficiently represented this way.

Lemma 7.3. *For any minimum cut $S \subseteq V$ in G, we can represent S as the symmetric difference $\triangle_{v\in D} \overline{v}$ for a set $D \subseteq \cup_j V_j$ with $\sum_{v\in D} d^W(v) \le O(Ls_0^2/(\epsilon\lambda))$.*

Proof. Since S is a minimum cut, we have $\partial_G S = \lambda$. Construct a sequence $S = S_0, S_1, \ldots, S_\ell$ as follows. For each j in increasing order, we ensure by induction that $S \subseteq V_j$ is a cut with value $\partial_{G_j} S \le (1 + 3\epsilon)^j \lambda$. Since $\epsilon = 1/\log^{1.1}|V|$ and $j \le \log|V|$, we have $\partial_{G_j} S \le (1 + 3\epsilon)^j \lambda \le 1.01\lambda$. So $\partial_{G_j} S \le 1.01\lambda$, and we can apply Lemma 3.2 to obtain a set S', which we index as S'_j. If $S'_j = \emptyset$ or $S'_j = V_j$, then we stop. Otherwise, let S_{j+1} be the set S'_j after the contraction from V_j to V_{j+1}, which is a cut with value

$$\partial_{G_{j+1}} S_{j+1} = \partial_{G_j} S'_j \le (1 + 3\epsilon) \partial_{G_j} S_j \le (1 + 3\epsilon)^{j+1} \lambda,$$

preserving the inductive statement.

We define $D' = \cup_{j=0}^{\ell-1} (S_j \triangle S'_j)$, which satisfies either $\triangle_{v\in D'} \overline{v} = S$ or $\triangle_{v\in D'} \overline{v} = V \setminus S$ depending on whether $S'_j = \emptyset$ or $S'_j = V_j$ on the last iteration. If the latter, then set $D = D' \cup \{v\}$ for the single vertex $v \in V_L$; otherwise, set $D = D'$. By construction, $S = \triangle_{v\in D} \overline{v}$. Also, by property (3) of Lemma 3.2, we have $\mathbf{vol}^W(S_j \triangle S'_j) \le O(s_0^2/(\epsilon\lambda))$ for each j, so the total degree $\sum_{v\in D} d^W(v)$ is at most L times that, as promised. ∎

7.2 Balanced sparsifier

In [Li21], the balanced sparsifier is simply a deterministic $n^{o(1)}$-approximate sparsifier, which is itself constructed by overlaying expanders at each level of an expander decomposition hierarchy. For technical reasons, we are unable to extend this approach to our setting. Instead, we will introduce *Steiner* vertices in the balanced sparsifier, which are eventually *split off*.

For each iteration $j \in \{1, 2, \ldots, L\}$ and each vertex $v \in V_j$, we introduce a Steiner vertex x_v^j. For each original vertex $v \in V$, let x_v^0 indicate that vertex. For each iteration $j \in \{0, 1, \ldots, L-1\}$ and each vertex $u \in V_j$, let $v \in V_{j+1}$ be the vertex that u contracts to; add an edge (x_u^j, x_v^{j+1}) of weight $d_{V_j}^W(u)$. We do not add any edges between original vertices. This concludes the construction of the balanced sparsifier, which we name $X = (V_X, E_X)$.

For each cut in G, we can extend it to a cut in X by adding each Steiner vertex to one side of the cut. The key lemma for the balanced sparsifier is the following. Intuitively, it says that cuts S in G that can be extended to small cuts in X (by siding the Steiner vertices optimally) are precisely the sets S that can be represented by the set difference of sets of the form $\overline{v}$ with small total degree $\sum_v d^W(v)$.

Lemma 7.4. *For any cut $S \subseteq V$ in G, the following two minima are equal:*

1. *The minimum value $\sum_{v\in D} d^W(v)$ over all sets $D \subseteq \cup_j V_j$ such that the symmetric difference $\triangle_{v\in D} \overline{v}$ equals S. Here, $d^W(v) = d_{G_j}^W(v)$ for the iteration j with $v \in V_j$.*
2. *The minimum value of $\partial_X S^*$ over all cuts $S^* \subseteq V_X$ with $S^* \cap V = S$.*

Proof. To show that the minimum in (1) is at most the minimum in (2), consider a set $D \subseteq \cup_j V_j$ such that the symmetric difference $\triangle_{v\in D}\overline{v}$ equals S. We construct a cut S^* in X with $S^* \cap V = S$ and total weight $\sum_{v\in D} d^W(v)$ as follows. Initialize $S_0 \leftarrow S$, and for each iteration j in increasing order, let $S'_j = S_j \triangle (D \cap V_j)$ and let $S_{j+1} \subseteq V_{j+1}$ be the contraction of the set S'_j into V_{j+1}, which is well-defined. The cut S^* is the union of $S \subseteq V$ and $\{x^j_v \mid v, j : v \in S_j\}$. For any edge (x^j_u, x^{j+1}_v) cut by S^* in X, where $u \in V_j$ contracts to $v \in V_{j+1}$, we must have either $u \in S_j$ or $v \in S_{j+1}$ but not both. It follows that $u \in D \cap V_j$. Conversely, for any $u \in D \cap V_j$ that contracts to $v \in V_{j+1}$, we have either $x^j_u \in S^*$ or $x^{j+1}_v \in S^*$ but not both, so edge (x^j_u, x^{j+1}_v) cut by S^* is cut by S^*. So there is a one-to-one mapping between vertices in D and edges cut by S^* with the same degree and edge weight, respectively. It follows that the minimum in (1) is at most the minimum in (2),

To show the other direction, consider a cut S^* in X with $S^* \cap V = S$. Denote by x^L the vertex x^L_v for the single vertex $v \in L$. Without loss of generality, assume that $x^L \notin S^*$. Let D consist of all vertices u for which the edge (x^j_u, x^{j+1}_v) is cut by S^* in X. By construction, $\sum_{v\in D} d^W(v)$ is the value of the cut S^* in X. It remains to show that $S = \triangle_{v\in D}\overline{v}$. For any $v \in S$ and iteration j, there is exactly one vertex $u \in V_j$ with $v \in \overline{u}$, and the corresponding vertex x^j_u is on the path from x^0_v to x^L. Since $x^L \notin S^*$, for any $v \in S$ there is an odd number of edges cut by S^* on the path from x^0_v to x^L, which corresponds to an odd number of vertices $u \in D$ for which $v \in \overline{u}$. It follows that $v \in \triangle_{u\in D}\overline{u}$. For any $v \notin S$, replacing "odd" with "even" in the argument above, we have $v \notin \triangle_{u\in D}\overline{u}$. Thus $S = \triangle_{v\in D}\overline{v}$, concluding the proof. ∎

7.2.1 Combining the two sparsifiers Let G' be the union of the following two graphs: the unbalanced sparsifier H with all edges weighted by W, and the graph X with all edge weights multiplied by $Z = (\epsilon\lambda)^2/Ls_0^2$. So G' is supported on vertex set V_X. For a cut $S \subseteq V$, we call a cut $S^* \subseteq V_X$ an *extension* of S if $S^* \cap V = S$. We now claim that the minimum cut can be extended to a near-minimum cut in G'.

Claim 7.1. *Assume that $\nu \ge \Omega(Ls_0^2/(\epsilon\lambda))$. For any minimum cut $S \subseteq V$ in G, there exists an extension $S^* \subseteq V_X$ with $\partial_{G'} S^* \le \big(1 + O(\epsilon) + O((\frac{Ls_0^2}{\epsilon\lambda^2})^2\eta)\big)\lambda$.*

Proof. By Lemma 7.3, we can write $S = \triangle_{v\in D}\overline{v}$ for a set $D \subseteq \cup_j V_j$ with $\sum_{v\in D} d^W(v) \le O(Ls_0^2/(\epsilon\lambda))$. By Lemma 7.2 with $d = O(Ls_0^2/(\epsilon\lambda)) \le \nu$, we obtain

$$W \cdot \partial_H S \le \partial_G S + O\left(\frac{Ls_0^2}{\epsilon\lambda^2}\right)^2 \eta\lambda = \left(1 + O\left(\frac{Ls_0^2}{\epsilon\lambda^2}\right)^2 \eta\right)\lambda.$$

So for any extension $S^* \subseteq S$, the total weight in $\partial_{G'} S^*$ resulting from the unbalanced sparsifier H is at most $\big(1 + O((\frac{Ls_0^2}{\epsilon\lambda^2})^2\eta)\big)\lambda$.

By Lemma 7.4, there exists a set $S^* \subseteq V_X$ with $\partial_X S^* = \sum_{v\in D} d^W(v) = O(Ls_0^2/(\epsilon\lambda))$. After dividing every weight by $Z = (\epsilon\lambda)^2/Ls_0^2$, the total weight in $\partial_{G'} S^*$ resulting from the balanced sparsifier X is at most $O(\epsilon\lambda)$. Together with the weight from the unbalanced sparsifier H, we obtain the desired bound. ∎

At the same time, we do not want any extensions of cuts to go below the minimum cut λ. The claim below ensures this does not happen.

Claim 7.2. *Assume that $\nu \ge Ls_0^2/(\epsilon^2\lambda)$. Then, for any cut $S \subseteq V$ in G and any extension $S^* \subseteq V_X$, we have $\partial_{G'} S^* \ge (1 - (\epsilon\lambda)^4/(L^2 s_0^4)\eta)\lambda$.*

Proof. Consider the minimum value $\sum_{v\in D} d^W(v)$ over all sets $D \subseteq \cup_j V_j$ such that the symmetric difference $\triangle_{v\in D}\overline{v}$ equals S. If $\sum_{v\in D} d^W(v) \ge \lambda/Z$, then by Lemma 7.4, any extension $S^* \subseteq V$ must have $\partial_X S^* \ge \lambda/Z$. After scaling by Z, the total weight in $\partial_{G'} S^*$ resulting from the balanced sparsifier X is at least λ, as promised.

Otherwise, we have $\sum_{v\in D} d^W(v) \le \lambda/Z = Ls_0^2/(\epsilon^2\lambda) \le \nu$, so Lemma 7.2 guarantees that

$$W \cdot \mathbb{1}_{\overline{u}}^T L_H \mathbb{1}_{\overline{v}} \ge \mathbb{1}_{\overline{u}}^T L_G \mathbb{1}_{\overline{v}} - (1/Z)^2 \eta\lambda = \lambda - (\epsilon\lambda)^4/(Ls_0^2)^2 \eta\lambda.$$

So the total weight in $\partial_{G'} S^*$ resulting from the unbalanced sparsifier H is at least $(1 - (\epsilon\lambda)^4/(Ls_0^2)^2\eta)\lambda$, as promised. ∎

7.2.2 The final construction

We start by finalizing our parameters for the algorithm:

1. We set $\eta = \text{poly}(\epsilon, 1/L, \lambda/s_0)$ small enough so that the right-hand side of the guarantees of Claim 7.1 and Claim 7.2 are $(1 + O(\epsilon))\lambda$ and $(1 - O(\epsilon))\lambda$, respectively.

2. We set $\nu = c \cdot Ls_0^2/(\epsilon^2\lambda)$ for some constant $c \geq 1$ so that Claim 7.1 and Claim 7.2 hold.

3. Given η and ν, we set $W \leq O(\frac{\eta\lambda^2}{\nu \ln(Lm)}) = \text{poly}(\epsilon, 1/L, \lambda/s_0, 1/\log n) \cdot \lambda$ so that Lemma 7.1 holds.

4. Recall that we set $Z = (\epsilon\lambda)^2/Ls_0^2 = \text{poly}(\epsilon, 1/L, \lambda/s_0)$.

We thus obtain a graph G' satisfying Claims 7.1 and 7.2. We want an unweighted sparsifier at the end, so we now “discretize” the edges of G'. By construction, the balanced sparsifier X consists of edges (x_u^j, x_v^{j+1}) of weight $d_{V_j}^W(u) \geq \lambda$, so all edges in G' originating from X have weight at least λ/Z. By construction, all edges originating from the unbalanced sparsifier H have weight W. For each edge e in G', replace it with $\lfloor w(e)/\min\{\epsilon W, \epsilon\lambda/Z\}\rfloor$ many parallel edges of weight $W' = \min\{\epsilon W, \epsilon\lambda/Z\}$ each, which preserves all cuts up to factor $(1 + O(\epsilon))$. We now bound

$$W' = \text{poly}(\epsilon, 1/L, \lambda/s_0, 1/\log n) \cdot \lambda = (\epsilon/\log n)^{O(1)}\lambda$$

since $L = O(\log n)$ and $s_0 = \lambda \,\text{polylog}(n)$. Let H' be the unweighted graph consisting of these edges of weight W'.

Next, we want to split off the Steiner vertices originating from X. Given a graph and a subset of vertices called the *terminals*, the Steiner minimum cut of a graph is the minimum cut separating at least two terminals. Let $V \subseteq V_X$ be the terminals in our setting. We now call the *splitting-off* algorithm of Bhalgat et al. [BHKP08] on H', which guarantees the following.

THEOREM 7.1. ([BHKP08]) *Given an unweighted graph $G = (V_X, E)$ with terminal set $T \subseteq V_X$ and Steiner minimum cut c, there is a deterministic, $\tilde{O}(m + nc^2)$ time algorithm that computes a graph G' satisfying the following:*

1. *Cuts cannot increase in value: for each cut $S \subseteq V_X$, we have $\partial_{G'}S \leq \partial_G S$,*

2. *The Steiner minimum cut of G equals the Steiner minimum cut of G', and*

3. *Every non-terminal vertex $v \in V_X \setminus T$ is isolated: $d_{G'}(v) = 0$.*

Let H'' be the output of applying Theorem 7.1 to H'. By construction of H', any minimum cut $S \subseteq V$ in G extends to a cut $S^* \subseteq V_X$ in H' with $\partial_{H'}S^* \leq (1+O(\epsilon))\lambda/W'$, and property (1) guarantees that $\partial_{H''}S^* \leq (1+O(\epsilon))\lambda/W'$ as well. In particular, the Steiner minimum cut of H' is at most $(1 + O(\epsilon))\lambda/W'$, and Theorem 7.1 takes $\tilde{O}(m + n(\lambda/W')^2) = \tilde{O}(m/\epsilon^{O(1)})$ time. On the other hand, any extension $S^* \subseteq V_X$ of a cut $S \subseteq V$ in G satisfies $\partial_{H'}S^* \geq (1 - O(\epsilon))\lambda/W'$, so the Steiner minimum cut of H' is at least $(1 - O(\epsilon))\lambda/W'$, and property (2) ensures that the Steiner minimum cut of H'' is also at least $(1 - O(\epsilon))\lambda/W'$. Finally, let $\widetilde{H}$ be the restriction of H'' to the terminal set V, which does not affect cut extensions by property (3). We have thus constructed a minimum cut sparsifier, also known as a *skeleton graph*, formalized below (after resetting $\epsilon > 0$ to be a constant factor smaller).

THEOREM 7.2. (SKELETON GRAPH, SEE THEOREM 1.5 OF [LI21]) *For any $0 < \epsilon \leq 1$, we can compute, in deterministic $\tilde{O}(m/\epsilon^{O(1)})$ time, an unweighted graph $\widetilde{H}$ and some weight $W' = (\epsilon/\log n)^{O(1)}\lambda$ such that*

1. *For any mincut ∂S^* of G, we have $W' \cdot |\partial_{\widetilde{H}}S^*| \leq (1 + \epsilon)\lambda$, and*

2. *For any cut $\emptyset \subsetneq S \subsetneq V$ of G, we have $W' \cdot |\partial_{\widetilde{H}}S| \geq (1 - \epsilon)\lambda$.*

Section 1.2.1 of [Li21] outlines Karger’s framework for deterministic minimum cut given the skeleton graph of Theorem 7.2. The final running time is $\tilde{O}(m)$.

Acknowledgements. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and the Austrian Science Fund (FWF) project Z 422-N, project "Static and Dynamic Hierarchical Graph Decompositions", I 5982-N, and project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the netidee SCIENCE Stiftung, 2020–2024.

References

[Bar95] Francisco Barahona. Packing spanning trees. *Mathematics of Operations Research*, 20(1):104–115, 1995.

[BHKP08] Anand Bhalgat, Ramesh Hariharan, Telikepalli Kavitha, and Debmalya Panigrahi. Fast edge splitting and edmonds' arborescence construction for unweighted graphs. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 455–464, 2008.

[BK96] András A Benczúr and David R Karger. Approximating st minimum cuts in õ (n 2) time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55, 1996.

[BK15] András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

[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.

[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. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022.

[Gab95] Harold Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of computer and system sciences*, 50:259–273, 1995.

[GMW20] Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in $o(mlog^2n)$ time. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, oct 1988.

[HRW20] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM J. Comput.*, 49(1):1–36, 2020.

[KAR93] David R. KARGER. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proc. the 4th ACM-SIAM Symposium on Discrete Algorithms, 1993*, 1993.

[Kar96] David R. Karger. Minimum cuts in near-linear time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 56–63, New York, NY, USA, 1996. Association for Computing Machinery.

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

[KS96] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

[KT19] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019.

[Li21] Jason Li. Deterministic mincut in almost-linear time. 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 384–395. ACM, 2021.

[LNPS23] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Near-linear time approximations for cut problems via fair cuts. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 240–275. SIAM, 2023.

[LS21] Jason Li and Thatchaphol Saranurak. Deterministic weighted expander decomposition in almost-linear time. *arXiv preprint arXiv:2106.01567*, 2021.

[Mat93] David W. Matula. A linear time 2+epsilon approximation algorithm for edge connectivity. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 500–504. ACM/SIAM, 1993.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

[NW61] C St JA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

[PST95] Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

[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.

[ST81] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. STOC '81, page 114–122, New York, NY, USA, 1981. Association for Computing Machinery.

[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 thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2616–2635. SIAM, 2019.

[You02] Neal E Young. Randomized rounding without solving the linear program. *arXiv preprint cs/0205036*, 2002.

A Missing details and analysis of Section 4

In this section we provide the implementation details and the analysis of the Unit-Flow algorithm from Section 4. The algorithm and its analysis largely follows the standard Push-relabel algorithm [GT88] and the original version of Unit-Flow for unweighted graphs in [HRW20] with fairly mechanical adaptation, and we include material directly from aforementioned work here for completeness. For readers familiar with Push-relabel and unweighted Unit-Flow, the main difference from the unweighted version of Unit-Flow is that we need to use the dynamic-tree data structure [ST81] for weighted graphs (as in the standard Push-relabel algorithm) since we want the running time to depend on the number of edges rather than the weighted volume. Similar to the unweighted case, by limiting the vertex labels to be at most h in Unit-Flow, we can bound the total number of relabels per node to be $O(h)$. This is a crucial factor in the running time of Push-relabel, and in particular bounds the number of times we cut or link each edge in the dynamic-tree data structure at $O(h)$. This is the reason that our Unit-Flow runs in time $\tilde{O}(m \cdot h)$ instead of the $\tilde{O}(mn)$ running time of standard Push-relabel where labels can go as high as $O(n)$.

A.1 Unit-Flow Implementation *Unit-Flow* (Algorithm A.1) takes as input an undirected weighted graph $G = (V, E, w)$ (possibly with self-loops), source function Δ and integer $\beta \geq 2$ such that $\forall v : 0 \leq \Delta(v) \leq \beta d^W(v)$, as well as a capacity parameter $U > 0$ on all edges. Each vertex v is a sink of capacity $d^W(v)$. Furthermore, the procedure takes as input an integer $h \geq \ln(\mathbf{vol}^W(G))$ to customize the Push-relabel algorithm, which we describe next.

In the Push-relabel algorithm [GT88], each vertex v has a non-negative integer label $l(v)$ which is initially zero. The label of a vertex only increases during the execution of the algorithm and in our modification of the standard Push-relabel the label cannot become larger than h. The bound of h on the labels makes the running time of *Unit-Flow* linear in h, but it may prevent our algorithm from routing all the supply to sinks even when there exists a feasible routing for the flow problem. However, when our algorithm cannot route a feasible flow, allowing labels of value up to h is sufficient to find a cut with low conductance (i.e., of value inversely proportional to h), which is our primary concern.

The algorithm maintains a pre-flow and the standard residual network, where each undirected edge $\{v, u\}$ in G corresponds to two directed arcs (v, u) and (u, v) in the residual network, with flow values such that $f(v, u) = -f(u, v)$, and $|f(v, u)|, |f(u, v)| \leq U \cdot w_e$. The residual capacity of an arc (v, u) is $r_f(v, u) = U \cdot w_e - f(v, u)$. We also maintain $f(v) = \Delta(v) + \sum_u f(u, v)$, which will be non-negative for all nodes v during the execution, and the excess $\mathrm{ex}(v) = \max\left(0, f(v) - d^W(v)\right)$.

As in the generic Push-relabel framework, an arc (v, u) is *eligible* if $r_f(v, u) > 0$ and $l(v) = l(u) + 1$. A vertex v is *active* if $l(v) < h$ and $\mathrm{ex}(v) > 0$. The algorithm maintains the property that for any arc (v, u) with positive residual capacity, $l(v) \leq l(u) + 1$. The algorithm maintains a (FIFO) queue Q of active vertices and repeatedly picks an active vertex v from Q to either pushes along an eligible arc incident to v if there is one, or to raise the label of v by 1 if v is active but there is no eligible arc out of v. In addition, we follow the efficient implementation of Push-relabel using the dynamic-tree data structure, which supports these operations below. The total time for a sequence of t tree operations starting with a collection of single-vertex trees is $O(t \ln n)$ when we have n vertices in total.

The dynamic-tree data structure maintains a set of vertex-disjoint rooted trees, where each vertex v has an associated real value $g(v)$, possibly $\pm\infty$. Tree edges are directed toward the root, and every vertex v is considered as both an ancestor and a descendant of itself. In our context, the value $g(v)$ of a vertex v in its dynamic tree is $r_f(v, w)$ if v has a parent w and $g(v) = \infty$ if v is a tree root. Each vertex starts in a single-vertex tree and has $g(v) = \infty$. As we don't aim to optimize the $\ln |C|$ term in the running time, we allow any tree to grow as large as possible (in contrast to the standard Push-relabel implementation). The tree edges in the dynamic trees form a subset of the eligible arcs intersecting the set of current incident edge being considered for each vertex (denoted as current(v)), and the tree operations allow us to push supply along an entire path in a tree in $O(\ln |C|)$ time, causing either a saturating push (i.e. use up the residual capacity of an arc on the path) or moving excess supply from some vertex in the tree all the way to the tree root (See the *Send* operation in Algorithm A.1). The algorithm keeps the pre-flow f in two different ways; If (v, w) is an edge not in any dynamic tree, $f(v, w)$ is stored explicitly with the edge; Otherwise, if $e = (v, w)$ is a dynamic tree edge, with w being the parent of v, then $g(v) = U \cdot w_e - f(v, w)$ is stored implicitly in the dynamic-tree data structure, and will be computed when the tree edge (v, w) is cut. However, note in our Unit-Flow implementation with incremental source supply, we only need to know the $f(v)$'s and $\text{ex}(v)$'s (i.e. supply and excess on nodes) each time Unit-Flow finishes with an interim source supply (to check if we move to the next stage or if there is enough total excess to make a cut). The supply ending on vertices can always be maintained explicitly during the execution of the algorithm, since only the two endpoints' supply will change when we send supply along a tree path. In particular, this means when $\Delta(\cdot)$ increases, we can just continue from the dynamic-tree data structure's state at the time when Unit-Flow finishes with the previous $\Delta(\cdot)$, and we don't need to do anything about the data structure (e.g. explicitly compute the supply routed on each edge) before finishing with the entire aggregate flow problem. To avoid including more redundant material, we refer readers to [GT88] for further details.

Dynamic-Tree Operations

- find-root(v): Find and return the root of the tree containing vertex v.
- find-size(v): Find and return the number of vertices in the tree containing vertex v.
- find-value(v): Compute and return $g(v)$.
- find-min(v): Find and return the ancestor w of v of minimum value $g(w)$. In case of a tie, choose the vertex w closest to the root.
- change-value(v, x): Add real number x to g(w) for each ancestor w of v. We follow the convention that $\infty + (-\infty) = 0$.
- link(v, w): Combine the trees containing vertices v and w by making w the parent of v. This operation does nothing if v and w are in the same tree or if v is not a tree root.
- cut(v): Break the tree containing v into two trees by deleting the edge from v to its parent. This operation does nothing if v is a tree root.

Algorithm A.1.

```
Unit-Flow($G$,$\Delta$,$U$,$h$,$w$)
. Initialization:
. . $\forall \{v,u\} \in E$, $f(u,v) = f(v,u) = 0$, $Q = \{v | \Delta(v) > d^W(v)\}$.
. . $\forall v$, $l(v) = 0$, and current($v$) is the first edge in its list of incident edges.
. While $Q$ is not empty
. . Remove the vertex $v$ at the front of $Q$.
. . Repeat
. . . Perform Tree-Push/Relabel($v$).
. . . If $w$ becomes active during this Tree-Push/Relabel operation
. . . . Add $w$ to the rear of $Q$
. . . End If
. . Until ex($v$) $= 0$ or $l(v)$ increases.
. . If $v$ is still active, add $v$ to the rear of $Q$.
. End While
```

```
Tree-Push/Relabel($v$)
  Applicability: $v$ is an active tree root (in the dynamic tree).
  Let $\{v, w\}$ be current($v$).
  If $l(v) = l(w) + 1$ and $r_f(v, w) > 0$ . . Case (1)
    Make $w$ the parent of $v$ by performing
      change-value($v, -\infty$), change-value($v, r_f(v, w)$), and link($v, w$).
    Push excess from $v$ to $w$ using Send($v$).
  Else (i.e. $l(v) \le l(w)$ or $r_f(v, w) = 0$) . . Case (2)
    If $\{v, w\}$ is not the last edge in $v$'s list of edges. . . Case ($2a$)
      Set current($v$) to be the next edge in $v$'s list of edges.
    Else (i.e. $(v, u)$ is the last edge of $v$) . . Case ($2b$)
      Set current($v$) to be the first edge of $v$'s list of edges.
      Perform cut($u$) and change-value($u, \infty$) for every child $u$ of $v$ (in the dynamic tree).
      Perform Relabel($v$).
    End If
  End If
```

```
Send($v$)
  Applicability: $v$ is active.
  While find-root($v$) $\neq v$ and ex($v$) $> 0$
    Send $\delta \leftarrow \min(\text{ex}(v), \text{find-value}(\text{find-min}(v)))$ units of supply along
      the tree path from $v$ by performing change-value($v, -\delta$)
    While find-value(find-min($v$)) $= 0$
      $u \leftarrow$ find-min($v$)
      Perform cut($u$) followed by change-value($u, \infty$).
    End While
  End While
```

```
Relabel($v$)
  Applicability: $v$ is active, and $\forall w \in V$, $r_f(v, w) > 0 \implies l(v) \le l(w)$.
  $l(v) \leftarrow l(v) + 1$.
```

In the rest of this section, we show how capping labels at h guarantees the running time to be $\tilde{O}(\mathbf{vol}(C) \cdot h)$ if C is the set of nodes with sinks saturated (and $\mathbf{vol}(C)$ is the unweighted volume of C).

LEMMA A.1. *Let $C = \{v : f(v) \ge d^W(v)\}$ be the set of saturated nodes when* Unit-Flow *finishes, the running time for* Unit-Flow *is $O(\mathbf{vol}(C) \cdot h \cdot \log |C|)$.*

Proof. Throughout the execution no dynamic tree will grow to have more than $|C|$ nodes, since we only link a tree rooted at v to another tree when v is active (i.e. has excess supply, thus already saturated, so $v \in C$). Thus the time per dynamic-tree operation is $O(\ln |C|)$.

The running time of Unit-Flow is mostly in the *Tree-Push/Relabel* calls. Each *Tree-Push/Relabel* invocation takes

(i) $O(1)$ time plus $O(1)$ dynamic-tree operations;

(ii) $O(1)$ dynamic-tree operations per cut operation in invocations of *Send*(v) and in Case ($2b$) plus time for *Relabel* in Case ($2b$).

To bound (ii), the total *Relabel* time is $O(|C|h)$ since we only need to raise the label of nodes in C for at most h times on each node. The total number of cut operations can be bounded as follows. Each cut operation can be attributed to a saturating push or to an edge scan for a *Relabel*. If we saturate an arc (v, w) during a push, we know $v \in C$ and $l(v) = l(w) + 1$. We won't be able to push along (v, w) (in either direction) until we raise the label of w to be above $l(v)$, so we can charge the cut to the *Relabel* of w. Similarly, we can charge the cut operations during the edge scan in *Relabel*(v) to that *Relabel*. In total, each node $v \in C$ will be charged $O(d(v))$ per *Relabel*, so $O(d(v)h)$ in total. The number of link operations is at most $|C| - 1$ more than the total number of cut operations, and thus can be bounded similarly. Summing over all vertices we get a bound of $O(\mathbf{vol}(C)h \ln |C|)$ for the running time from (ii).

The running time from (i) is $O(\ln |C|)$ times the number of times we add a vertex to Q (since we only call *Tree-Push/Relabel* when we remove a vertex from Q). Beyond the (at most $|C|$) vertices we add to Q during initialization (i.e. nodes starting with $\Delta(v) \geq d^W(v)$), a node v can only be added to Q when $l(v)$ increases or as a result of $\mathrm{ex}(v)$ increases above 0. The former happens at most $|C|h$ times in total, and the latter only happens during Case (1) when we send supply along a tree path. In this case, the number of nodes with excess increasing above 0 is at most 1 more than the number of cut operations, and we already bounded the total number of cut operations by $O(\mathbf{vol}(C)h)$ in the previous paragraph. The number of times we get to Case (1) is bounded by the total number of link operations, which again is bounded by $O(\mathbf{vol}(C)h)$ as in the previous paragraph. In total this bounds the total number of times a vertex is added to Q to be $O(\mathbf{vol}(C)h)$, and thus the running time from (i) is $O(\mathbf{vol}(C)h \ln |C|)$.

The additional running time if we need to find a low conductance cut will be $O(\mathbf{vol}(C))$ since it's just a sweeping cut algorithm using the vertex labels, see details in the proof of Theorem 4.1. ∎

A.2 Analysis of Unit-Flow The *Unit-Flow* procedure (Algorithm A.1) is a fairly straightforward implementation of the push-relabel framework with a cap on the labels: If a vertex has label h, and we cannot push remaining excess from the vertex, instead of relabeling it to $h+1$, we leave the vertex at label h and it's never considered as active from then on. Upon termination, we have a pre-flow f , and labels l on vertices. We make the following observations.

Observation A.1. *During the execution of* Unit-Flow*, we have*

(1) If v is active at any point, the final label of v cannot be 0. The reason is that v will remain active until either $l(v)$ is increased to h, or its excess is pushed out of v, which is applicable only when $l(v)$ is larger than 0 at the time of the push.

(2) Each vertex v is a sink that can absorb up to $d^W(v)$ units of supply, so we call the $f(v) - \mathrm{ex}(v) = \min(f(v), d^W(v))$ units of supply remaining at v the absorbed *supply. The amount of absorbed supply at v is between $[0, d^W(v)]$, and is non-decreasing. Thus any time after the point that v first becomes active, the amount of absorbed supply is $d^W(v)$. In particular any time the algorithm relabels v, there have been $d^W(v)$ units of supply absorbed by v.*

Upon termination of Unit-Flow *procedure, we have*

(3) For any edge $\{v, u\} \in E$, if the labels of the two endpoints differ by more than 1, say $l(v) - l(u) > 1$, then arc (v, u) is saturated. This follows directly from a standard property of the push-relabel framework, where $r_f(v, u) > 0$ implies $l(v) \leq l(u) + 1$.

Although *Unit-Flow* may terminate with $\mathrm{ex}(v) > 0$ for some v, we know all such vertices must have label h, as the algorithm only stops trying to route v's excess supply to sinks when v reaches level h. Thus we have the following lemma. Also note that once some supply is absorbed by a sink or is left as excess on a node at height h when Unit-Flow stops, even if $\Delta(\cdot)$ is incremental and we resume the execution of Unit-Flow, the absorbed supply remains absorbed, and similarly for the excess supply on node with label h. Thus it is well-defined to discuss these concepts at the time Unit-Flow terminates with some interim $\Delta(\cdot)$.

Lemma A.2. *Upon termination of* Unit-Flow *with input (G, Δ, h, U, β), where $\Delta(v) \leq \beta d^W(v)$ for all v, the pre-flow and labels satisfy*

(a) If $l(v) = h$, $(\beta + U)d^W(v) \geq f(v) \geq d^W(v)$;

(b) If $h - 1 \geq l(v) \geq 1$, $f(v) = d^W(v)$;

(c) If $l(v) = 0$, $f(v) \leq d^W(v)$.

Proof. By Observation A.1.(2), any vertex with label larger than 0 must have $f(v) \geq d^W(v)$. The algorithm terminates when there is no active vertices, i.e. $\mathrm{ex}(v) > 0 \implies l(v) = h$, so all vertices with label below h must have $f(v) \le d^W(v)$. Moreover, $f(v) \le wd^W(v)$ since at the beginning $f(v) = \Delta(v) \le \beta d^W(v)$, and the pre-flow routing can send at most an additional $U \cdot d^W(v)$ units of supply to v since that's the total capacity of edges incident to v for the flow problem. $\square$

Now we can prove the main theorem about *Unit-Flow*. Everything other than the running time largely follows the analysis of the unweighted version from [HRW20], and the running time follows mechanically from the Push-relabel analysis in [GT88] using the dynamic-tree data structure from [ST81]

THEOREM A.1. *Given $G, \Delta, h, U, \beta \ge 2$ such that $\Delta(v) \le \beta d^W(v)$ for all v,* Unit-Flow *terminates with a pre-flow f, where we must have one of the following three cases*

(1) f is a feasible flow, i.e. $\forall v : \mathrm{ex}(v) = 0$. All units of supply are absorbed by sinks.

(2) f is not a feasible flow, but $\forall v : f(v) \ge d^W(v)$, i.e., at least $\mathbf{vol}^W(G)$ units of supply are absorbed by sinks.

(3) If f satisfies neither of the two cases above, denote T as the set of nodes whose labels are h (i.e. largest possible label), and assume $h \ge \ln \mathbf{vol}^W(G)$, then we can find a cut $(A, \bar{A})$ such that

(a) $T \subseteq A$, and $(\beta + U)d^W(v) \ge f(v) \ge d^W(v)$ for all $v \in T$, and $f(v) = d^W(v)$ for all $v \in A \setminus T$, and $f(v) \le d^W(v)$ for all $v \notin A$.

(b) The cut's conductance $\Phi(A) = \frac{W(A, V\setminus A)}{\min\left(\mathbf{vol}^W(A), \mathbf{vol}^W(G) - \mathbf{vol}^W(A)\right)} \le \frac{20 \ln \mathbf{vol}^W(G)}{h} + \frac{\beta}{U}$.

(c) Let $\mathrm{ex}(T)$ be the total amount of excess supply (which can only be on nodes in T), i.e. $\mathrm{ex}(T) = \sum_v \max\left(0, f(v) - d^W(v)\right)$, we have $\mathbf{vol}^W(A) \ge \frac{\mathrm{ex}(T)}{(\beta - 1) + 10U \ln\left(\mathbf{vol}^W(G)\right)/h}$

Let $C = \left\{v : f(v) \ge d^W(v)\right\}$ be the set of saturated nodes when Unit-Flow finishes, the running time for Unit-Flow *is $O(\mathbf{vol}(C) \cdot h \cdot \log \mathbf{vol}(C))$. Moreover, the above guarantees hold in the case when Δ is incremental, i.e., after Unit-Flow finishes working with the supply from the current $\Delta(\cdot)$, new source supply can be added on nodes to increase $\Delta(\cdot)$, and the theorem holds when Unit-Flow finishes with any interim $\Delta(\cdot)$.*

Proof. We use the labels at the end of *Unit-Flow* to divide the vertices into groups

$$B_i = \{v | l(v) = i\}$$

If $B_h = \emptyset$, no vertex has positive excess, so all $|\Delta(\cdot)|$ units of supply are absorbed by sinks, and we end up with case (1).

If $B_h \ne \emptyset$, but $B_0 = \emptyset$, by Lemma A.2 every vertex v has $f(v) \ge d^W(v)$, so we have $\sum_v d^W(v) = \mathbf{vol}^W(G)$ units of supply absorbed by sinks, and we end up with case (2).

Case (3)**:** When both B_h and B_0 are non-empty, we compute the cut $(A, V \setminus A)$ as follows: Let $S_i = \cup_{j=i}^{h} B_j$ be the set of vertices with labels at least i. We sweep from h to 1, and let A be the first i such that $\Phi(S_i) \le 20(\frac{\ln \mathbf{vol}^W(G)}{h} + \frac{\beta}{U})$. The claims in (3)($a$) follow directly from $S_h \subseteq A \subseteq S_1$ and Lemma A.2.

For (3)(b), We will show that there must exists some S_i satisfying the conductance bound. For any i, an edge $e = \{v, u\}$ across the cut S_i, with $v \in S_i, u \in V \setminus S_i$, must be one of the two groups:

1. In the residual network, the arc (v, u) has positive residual capacity $r_f(v, u) > 0$, so $l(v) \le l(u) + 1$. But we also know $l(v) \ge i > l(u)$ as $v \in S_i, u \in V \setminus S_i$, so we must have $l(v) = i, l(u) = i - 1$. In the Push-relabel framework, such edges are called *eligible arcs.*

2. In the residual network, if $r_f(v, u) = 0$, then (v, u) is a saturated arc sending $U \cdot w(e)$ units of supply from S_i to $V \setminus S_i$.

Suppose the total edge weight of edges in the first group is $z_1(i)$, and the total weight of edges in the second group is $z_2(i)$. By the following region growing argument, we can show there exists some choice of $i = i^*$, such that

$$z_1(i^*) \le \frac{10 \min(\mathbf{vol}^W(S_{i^*}), \mathbf{vol}^W(G) - \mathbf{vol}^W(S_{i^*})) \ln \mathbf{vol}^W(G)}{h} \tag{A.1}$$

If $\mathbf{vol}^W(S_{\lfloor h/2\rfloor}) \leq \mathbf{vol}^W(G)/2$, we start the region growing argument from $i = h$ down to $\lfloor h/2\rfloor$. By contradiction, suppose $z_1(i) \geq \frac{10\mathbf{vol}^W(S_i)\ln\mathbf{vol}^W(G)}{h}$ for all $h \geq i \geq \lfloor h/2\rfloor$, which implies $\mathbf{vol}^W(S_i) \geq \mathbf{vol}^W(S_{i+1})(1 + \frac{10\ln\mathbf{vol}^W(G)}{h})$ for all $h > i \geq \lfloor h/2\rfloor$. Since $\mathbf{vol}^W(S_h) = \mathbf{vol}^W(B_h) \geq 1$ and $h \geq \ln\mathbf{vol}^W(G)$, we will have $\mathbf{vol}^W(S_{\lfloor h/2\rfloor}) \geq (1 + \frac{10\ln\mathbf{vol}^W(G)}{h})^{h/2} \gg \mathbf{vol}^W(G)$, which gives a contradiction. The case where $\mathbf{vol}^W(S_{\lfloor h/2\rfloor}) > \mathbf{vol}^W(G)/2$ is symmetric, and we run the region growing argument from $i = 1$ up to $\lfloor h/2\rfloor$ instead.

For any i, we can bound $z_2(i)$ as follows. Since the pre-flow pushes $z_2(i)U$ units of supply in total from S_i to $V \setminus S_i$ along the saturated arcs in the second group, $z_2(i)U$ is at most $\sum_{v\in S_i}\Delta(v) + z_1(i)U$, i.e. the sum of the source supply in S_i plus the supply pushed into S_i along the eligible arcs in the first group (and the total capacity of those arcs is $z_1(i)U$). As $\Delta(v) \leq \beta d^W(v)$ for all v, we know

$$z_2(i) \leq \frac{\beta\mathbf{vol}^W(S_i)}{U} + z_1(i)$$

On the other hand, $z_2(i)U$ is at most $\sum_{v\in V\setminus S_i} f(v) + z_1(i)U$, as the $z_2(i)U$ units of supply pushed into $V \setminus S_i$ either remain at vertices in $V \setminus S_i$, or back to S_i along the reverse arcs of the eligible arcs in the first group, and the total capacity of these reverse arcs is also $z_1(i)U$. Since any $v \in V \setminus S_i$ is not with label h, thus $f(v) \leq d^W(v)$, then we get

$$z_2(i) \leq \frac{\mathbf{vol}^W(V \setminus S_i)}{U} + z_1(i)$$

The two upper-bounds of $z_2(i)$ together give

$$z_2(i) \leq \frac{\beta\min(\mathbf{vol}^W(S_i), \mathbf{vol}^W(G) - \mathbf{vol}^W(S_i))}{U} + z_1(i) \tag{A.2}$$

We know there exists some i^* such that $z_1(i^*)$ is bounded by (A.1), together with (A.2), we have

$$z_1(i^*) + z_2(i^*) \leq \min(\mathbf{vol}^W(S_{i^*}), 2m - \mathbf{vol}^W(S_{i^*}))(\frac{20\ln\mathbf{vol}^W(G)}{h} + \frac{\beta}{U})$$

thus $\Phi(S_{i^*}) \leq \frac{20\ln\mathbf{vol}^W(G)}{h} + \frac{\beta}{U}$, which completes the proof of $(3)(b)$.

For $(3)(c)$, we again use the observation that the supply ending in A (i.e. S_{i^*}) can only come from either the source supply started in A or sent into A via the reverse arcs of the eligible arcs in the first group. The former is at most $\beta\mathbf{vol}^W(A)$ and the latter is at most $z_1(i^*)U \leq \frac{10U\mathbf{vol}^W(A)\ln\mathbf{vol}^W(G)}{h}$ by (A.1). Moreover, since all nodes in A have their sinks saturated, we know the following about the excess

$$\mathrm{ex}(T) + \mathbf{vol}^W(A) \leq \beta\mathbf{vol}^W(A) + \frac{10U\mathbf{vol}^W(A)\ln\mathbf{vol}^W(G)}{h},$$

which gives the bound on $\mathrm{ex}(T)$ in the theorem statement.

The running time claim follows from Lemma A.1. ∎