Improved Tree Sparsifiers in Near-Linear Time *

Daniel Agassy† Dani Dorfman‡ Haim Kaplan§

November 11, 2025

Abstract

A *tree cut-sparsifier* T of quality α of a graph G is a single tree that preserves the capacities of all cuts in the graph up to a factor of α. A *tree flow-sparsifier* T of quality α guarantees that every demand that can be routed in T can also be routed in G with congestion at most α.

We present a near-linear time algorithm that, for any undirected capacitated graph $G = (V, E, c)$, constructs a tree cut-sparsifier T of quality $O(\log^2 n \log\log n)$, where $n = |V|$. This nearly matches the quality of the best known polynomial construction of a tree cut-sparsifier, of quality $O(\log^{1.5} n \log\log n)$ [Räcke and Shah, ESA 2014]. By the flow-cut gap, our result yields a tree flow-sparsifier (and congestion-approximator) of quality $O(\log^3 n \log\log n)$. This improves on the celebrated result of [Räcke, Shah, and Täubig, SODA 2014] (RST) that gave a near-linear time construction of a tree flow-sparsifier of quality $O(\log^4 n)$.

Our algorithm builds on a recent *expander decomposition* algorithm by [Agassy, Dorfman, and Kaplan, ICALP 2023], which we use as a black box to obtain a clean and modular foundation for tree cut-sparsifiers. This yields an improved and simplified version of the RST construction for cut-sparsifiers with quality $O(\log^3 n)$. We then introduce a near-linear time *refinement phase* that controls the load accumulated on boundary edges of the sub-clusters across the levels of the tree. Combining the improved framework with this refinement phase leads to our final $O(\log^2 n \log\log n)$ tree cut-sparsifier.

*This work was supported in part by Israel Science Foundation grant no. 1595-19 and the Blavatnik Family Foundation.

†Tel Aviv University, danielagassy@mail.tau.ac.il

‡Max Planck Institute for Informatics, ddorfman@mpi-inf.mpg.de

§Tel Aviv University, haimk@tauex.tau.ac.il

1 Introduction

A cut-sparsifier of a weighted undirected graph $G = (V, E, w)$ is a sparse graph that preserves the cuts of G up to some factor. Formally $H = (V_H, E_H, w_H)$ is a cut-sparsifier of G of quality α if $V \subseteq V_H$ and for every $S \subseteq V$ it holds that $\mathrm{cap}(S, V \setminus S) \leq \mathrm{mincut}_H(S, V \setminus S) \leq \alpha \cdot \mathrm{cap}(S, V \setminus S)$, where

$$\mathrm{mincut}_H(S, V \setminus S) := \min_{U:\ S \subseteq U,\, V \setminus S \subseteq V_H \setminus U} \mathrm{cap}_H(U, V_H \setminus U).$$

A stronger notion is of flow-sparsifiers. We say that $H = (V_H, E_H, w_H)$, where $V \subseteq V_H$, is a flow-sparsifier of $G = (V, E, w)$ of quality α if every multi-commodity flow problem that can be routed in G with congestion 1, can be also routed in H with congestion 1 and, moreover, every multi-commodity flow problem that can be routed in H with congestion 1, can be also routed in G with congestion α.

A concept closely related to tree flow-sparsifiers is that of *congestion-approximators*. An α-congestion-approximator of G is a collection of cuts $\{(S_i, V \setminus S_i)\}_{i=1}^{k}$ that approximately capture the optimal congestion needed to route multi-commodity flows in G. Formally, for any demand matrix Q, consider the minimum ratio (over all cuts $(S_i, V \setminus S_i)$) between the capacity of the cut and the total demand separated by it, *i.e.*, $\frac{\mathrm{cap}(S_i, V \setminus S_i)}{\mathrm{dem}_Q(S_i, V \setminus S_i)}$. The collection $\{(S_i, V \setminus S_i)\}$ is an α-congestion-approximator if this ratio α-approximates the minimum congestion required to route Q in G. Tree flow-sparsifiers arising from hierarchical decompositions yield congestion approximators of the same quality, consisting of only a near-linear number of cuts. In a breakthrough result, Sherman [She13] showed that congestion-approximators of polylogarithmic quality suffice to compute approximate maximum flow in near-linear time.

Räcke [Räc02] proved the existence of tree flow-sparsifiers of quality $O(\log^3 n)$. The construction is based on a hierarchical decomposition of G. This result was soon improved to a polynomial time algorithm for computing a tree flow-sparsifier of quality $O(\log^2 n \log\log n)$ [HHR03]. This series of works has culminated in an almost linear time (that is, $O\left(m^{1+o(1)}\right)$ time) construction of a tree flow-sparsifier of quality $O(\log^4 n)$ by Räcke, Shah, and Täubig [RST14], which was later improved to a near-linear time (that is, $\tilde{O}(m)$ time[1]) construction due to Peng's near-linear time approximate max flow algorithm [Pen16]. This $O(\log^4 n)$ bound is the best known for near-linear time constructions of a tree flow-sparsifier and of congestion-approximators of near-linear size.

Even though flow-sparsifiers are robust, for many applications a cut-sparsifier suffices (see, *e.g.*, [CKS04, EKLN07, AGG+09, BFK+14, VDBCK+24]). The best known polynomial time construction by Räcke and Shah [RS14] gives a tree cut-sparsifier of quality $O(\log^{1.5} n \log\log n)$. Grid graphs impose an $\Omega(\log n)$ lower bound on the quality of tree cut-sparsifiers and flow-sparsifiers [BL97, MadHVW97].

A tree flow-sparsifier T of quality α is also a tree cut-sparsifier of the same quality. Therefore, [RST14, Pen16] implies a near-linear time construction of a tree cut-sparsifier of quality $O(\log^4 n)$, and this is the best near-linear time construction to date. We give an $\tilde{O}(m)$ time construction of tree cut-sparsifier of quality $O(\log^2 n \log\log n)$, a tree flow-sparsifier of quality $O(\log^3 n \log\log n)$, and a congestion-approximator of near-linear size of quality $O(\log^3 n \log\log n)$. This improves for the first time upon the 2014 bound of [RST14, Pen16]. The following theorem is our main result.

Theorem 1.1. *Given a graph $G = (V, E)$ of m edges, there exists a randomized algorithm which takes $\tilde{O}(m)$ time and with high probability finds a weighted tree $T = (V_T, E_T, w_T)$ with $V \subseteq V_T$, such that T is a cut-sparsifier of G with quality $O(\log^2 n \log\log n)$. The same tree is also a flow-sparsifier*

[1] The $\tilde{O}$ notation is used to hide $\log^c n$ factors.

of G with quality $O(\log^3 n \log\log n)$, which in particular induces a congestion-approximator of G with quality $O(\log^3 n \log\log n)$ and near-linear size.

1.1 Techniques

We now elaborate on the tools and techniques that allows us to obtain Theorem 1.1. First, we develop a way to work directly with cuts rather than flows.

Demand State Formulation. Certifying that a hierarchical tree T is a flow-sparsifier of a graph G with quality α, amounts to showing that every multi-commodity flow problem Q routable in T with congestion 1, is routable in G with congestion α. The natural approach is to route the mass in G according to the structure of T: we partition G to clusters according to the laminar set family given by the nodes of T, and iteratively route flow in G from the boundary edges of each cluster to the boundary edges of its parent cluster. The process begins at the leaf clusters (corresponding to the vertices V) and proceeds upward, canceling demands whenever both endpoints lie inside the same cluster.

For the cut-sparsifier setting, we replace explicit routings by a notion of G *respecting* demand matrices. We say that G α-respects a demand matrix if for every cut $(S, \bar{S})$ the total amount of that demand that must cross $(S, \bar{S})$ is at most $\frac{1}{\alpha} \cdot \mathrm{cap}_G(S, \bar{S})$. To prove that T is a tree cut-sparsifier we need to show that G $\frac{1}{\alpha}$-respects every demand matrix that is 1-respected by T. To carry out this task we use the notion of *demand states* (Definition 3.13). A demand state assigns to each vertex a vector of commodity masses, and each update to the state corresponds to moving mass. Demand states allow us to formally define and maintain invariants, reason about how mass is redistributed across clusters, and rigorously charge these movements to the capacity of cuts. In contrast to flow-sparsifiers, these movements and corresponding charges are not necessarily achieved by explicit routing. While similar techniques were used in previous work, using this demand state formulation allows us to provide a clearer and more systematic presentation of the algorithms, making their structure explicit.

Expander Decomposition. Leveraging the "balanced cut or expander" primitive from [ADK25] with adaptively chosen expansion parameters, we develop a unified approach that allows us to simplify and strengthen the [RST14] framework to obtain a tree cut-sparsifier of quality $O(\log^3 n)$. Afterwards, we use the same approach to implement a variant of the refinement phase of [HHR03, RS14] in near-linear time, as we describe below.

Using this primitive from [ADK25], together with the fair-cuts computation [LNPS23], allows us to abstract away the details of the underlying algorithmic machinery and significantly simplify the construction of the tree cut-sparsifier, when compared to *e.g.*, [RST14]. This connects tree cut-sparsifier constructions to modern sparse-cut tools and shows how they can be constructed using these tools.

A Near-Linear Refinement Phase. A main caveat in the construction of [RST14] is that the boundary edges of the clusters are not *flow-linked*. The notion of a flow-linked set, introduced by [CKS05], means that it is possible to route an all-to-all multi-commodity flow between the nodes (or edges) of this set. The lack of this property in [RST14] adds a logarithmic factor to the quality of the resulting tree flow-sparsifier. For more details, see Section 2.4. In the context of tree cut–sparsifiers, flow-linkedness corresponds to *expansion*. By applying the "balanced cut or expander" primitive of [ADK25] to a given cluster S, we can test whether the boundary of S expands. If not, we split S along a balanced cut and recur on both sides. This recursive process, which we call the *refinement phase*, allows us to partition a cluster S into $\{S_1, \ldots, S_t\}$ such that in each S_i, the boundary edges expand.

Additionally, similar to [HHR03, RS14], we show that there exists a low-congestion flow from the inter-cluster edges of the refinement partition to the boundary of S. We do this by carefully using the guarantees supplied by [ADK25]. This requires overcoming several technical obstacles, as we need to meticulously control both the number of boundary edges and the number of vertices in the resulting sub-clusters. Importantly, we are able to perform this step in near-linear time, instead of polynomial time.

Previous constructions [HHR03, RS14] relied on intricate recursive schemes that handled "bad child events", a subcall that violated the recursion invariants, by restarting parts of the computation. In contrast, our approach is entirely modular: we alternately apply a *merge phase* (on which we elaborate in Section 2.5) and a refinement phase, each performing a single, independent partition of the current cluster (no backtracking is needed). This simple alternation yields a much cleaner and modular tree cut-sparsifier construction of quality $O(\log^2 n \log\log n)$.

1.2 Further Background

Our work builds on the tree flow-sparsifier constructions of [RST14] and [HHR03], combined with an expander decomposition result by [ADK25].

A crucial component in [RST14] was a variant of the cut-matching game, introduced by Khandekar, Rao, and Vazirani [KRV09]. The cut-matching game [KRV09] provides an $O(\log^2 n)$-approximate sparsest cut in near-linear time. By running a "non-stop" version of this game, [RST14] obtained *balanced* $O(\log^2 n)$-approximate sparse cuts, or alternatively, constructed clusters whose inter-cluster edges (not including boundary edges) are *flow-linked*.

A stronger cut strategy was developed by [OSVV08], achieving an $O(\log n)$-approximation for the sparsest cut problem. The expansion produced by this cut strategy follows from Cheeger's inequality [Che70] and therefore, at least intuitively, is unsuitable for flow routing purposes. This stands in contrast to the cut player of [KRV09], whose expansion follows from embedding a uniform demands matrix (all 1's) into the expanding set.

Agassy, Dorfman, and Kaplan [ADK23] were able to translate [OSVV08]'s improved cut strategy into a "balanced cut or expander" procedure of approximation $O(\log n)$, resulting in a near-linear time expander decomposition algorithm which improved on the prior expander decomposition algorithm by Saranurak and Wang [SW19] that uses [KRV09]'s cut strategy. In [ADK25], Agassy, Dorfman and Kaplan show how to extend their results to the case of generalized conductance. By generalized conductance, they refer to conductance with respect to a vertex measure. Formally, given a vertex measure $\mu : V \to \mathbb{R}_{\geq 0}$, and a set $S \subseteq V$, the expansion of a cut $(S, \bar{S})$ with respect to μ is defined as $\Phi_G^{\mu}(S, \bar{S}) := \frac{\mathrm{cap}(S,\bar{S})}{\min(\mu(S),\mu(\bar{S}))}$, where $\mu(U)$, for $U \subseteq V$, is defined as $\mu(U) := \sum_{v \in U} \mu(v)$. We use the result by [ADK25] in a black-box manner as explained in Section 2.5.

Typically, as in this work, hierarchical decompositions are computed *top-down* by recursively splitting clusters starting from the root V. Li, Rao, and Wang [LRW25] take the opposite, *bottom-up* approach, constructing a congestion approximator of quality $O(\log^{10} n)$. Their main tool is a weak expander decomposition: they build a sequence of partitions $P_1, P_2, \ldots, P_L$ of V, where each partition corresponds to a weak expander decomposition, and P_1 is the partition into singleton clusters and $P_L = \{V\}$. The collection of cuts from these partitions does not form a laminar family. Instead, the congestion-approximator's set of cuts consists of all intersections of sub-clusters drawn from different levels. A technical challenge is that expander-decomposition routines typically invoke approximate max-flow algorithms that themselves rely on the top-down congestion-approximator of [RST14]. The construction of [LRW25] avoids this potential circularity by maintaining "pseudo"-congestion-approximators derived from the partial hierarchy $P_1, \ldots, P_i$ and using them within the max-flow subroutines required for the weak expander decomposition.

We note that the "balanced cut or expander" primitive of [ADK25], which we use extensively, internally computes fair-cuts (approximate maximum flows) via the algorithm of [LNPS23]. This algorithm requires a congestion–approximator for the underlying graph G. By instantiating the congestion–approximator required by the fair-cut routine [LNPS23] with the construction of [LRW25], our tree cut–sparsifiers can be implemented *without* invoking the congestion–approximator of [RST14]. This means that one can also think of our work (in particular, Sections 4 and 5) as a simplification of the construction of [RST14] (which does not *depend* on it).

1.3 Related Work

When allowing to sparsify with a convex combination of trees, Räcke achieved a polynomial time algorithm for computing a flow-sparsifier [Räc08] of optimal $O(\log n)$ quality. However, some problems require a flow-sparsifier which is a single tree (*e.g.*, [KKM14, AGG+09, BFK+14]).

Using modern techniques of (dynamic) expander decomposition, and expander-hierarchies, [GRST21] achieved a fully dynamic, $n^{o(1)}$ update and $O(\log^{1/6} n)$ query time, data structure for maintaining a tree flow-sparsifier of quality $n^{o(1)}$.

Tree flow–sparsifiers have been extensively used in the context of oblivious routing schemes, that is, routings that are independent of the input demands and preassign flow paths for every pair of vertices. There has also been progress on hop-constrained routing, where paths are kept short. Ghaffari, Haeupler, and Žužić obtain a *static* oblivious scheme that, for any hop bound h, uses paths of length at most $h \cdot \text{poly}(\log n)$ and achieves congestion within $n^{o(1)}$ of the best routing that uses paths of length at most h [GHZ21]. Haeupler, Räcke, and Ghaffari introduce hop-constrained expander decompositions, from which they derive hop-constrained routing [HRG22]. These works are complementary to our goal of producing a single tree with provable cut guarantees and near-linear preprocessing time.

2 Overview of Our Results and Techniques

This section is organized as follows. Section 2.1 introduces key definitions used throughout the section (additional definitions and notation are deferred to Section 3). In Section 2.2, we detail our results using these definitions. In Section 2.3, we introduce a general framework for analyzing the quality of tree cut-sparsifiers. Section 2.4 reviews the key ideas and techniques from previous work. In Section 2.5, we outline the approach used to prove Theorem 2.9, and in Section 2.6, we describe the additional ideas that give the improved result stated in Theorem 2.8.

2.1 Preliminaries

We use log to denote the base-2 logarithm. Throughout the paper, we assume all graphs are undirected. Given an undirected graph $G = (V, E)$ and a set $S \subseteq V$, we denote by $\deg_G(v)$ the *degree* of v (for a weighted graph, this is the sum of the weights of the edges adjacent to v). The *volume* of a set S is defined as $\mathbf{vol}_G(S) := \sum_{v \in S} \deg_G(v)$. Given vertex weights $\mu : V \to \mathbb{R}_{\geq 0}$, and a set $U \subseteq V$, we denote the total weight of U as $\mu(U) := \sum_{v \in U} \mu(v)$. In almost all of our use cases, μ will be an indicator measure. That is, $\mu(v) = \mathbb{1}_{\{v \in X\}}$ for some $X \subseteq V$, and μ simply distinguishes the set X of vertices (formally, $\mu(v) = 1$ if $v \in X$ and otherwise $\mu(v) = 0$). For disjoint sets $A, B \subseteq V$, we denote by $E_G(A, B)$ the set of edges connecting A and B. We denote by $\text{cap}_G(A, B)$ the number of edges in $E_G(A, B)$, or the sum of their weights if the graph is weighted. We sometimes omit the subscript when the graph is clear from the context. Denote $\bar{A} = V \setminus A$. If A and $\bar{A}$ are both non-empty, then we call $(A, \bar{A})$ a *cut*.

Definition 2.1 (Multi-Commodity Flow Problem, Demand Matrix)**.** *Given a vertex set V, a* multi-commodity flow problem *Q on V is specified by a* demand matrix *$D_Q \in \mathbb{R}_{\geq 0}^{V \times V}$. Each vertex $u \in V$ is associated with a single* commodity*: the u'th row of D_Q describes how much of u's commodity is to be delivered to every vertex $v \in V$. In other words, $D_Q(u,v)$ denotes the amount of flow of commodity u that must be sent from u to v (i.e., u is the source of its commodity).*

Given $S \subseteq V$, we denote the demand of Q across the cut $(S, \bar{S})$ *by*

$$\mathrm{dem}_Q(S, \bar{S}) := \sum_{u \in S} \sum_{v \in \bar{S}} (D_Q(u,v) + D_Q(v,u)).$$

We identify a multi-commodity flow problem with its demand matrix, and use Q and D_Q interchangeably.

A specific flow problem which will be used multiple times is the *all-to-all* flow problem.

Definition 2.2 (All-to-all flow problem)**.** *Let $G = (V, E)$ and let $A \subseteq V$. The all-to-all flow problem on A is defined by the demand matrix* $D_A(u,v) := \begin{cases} \frac{1}{|A|}, & \text{if } u, v \in A, \\ 0, & \text{otherwise.} \end{cases}$.

We now define congestion, which quantifies the efficiency of a given flow.

Definition 2.3 (Congestion)**.** *Given a graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}_{>0}$ and a flow $f : E \to \mathbb{R}_{\geq 0}$ routing a multi-commodity flow problem Q (f does not necessarily abide capacity constraints), the* congestion *of f in G is*

$$\mathrm{cong}_G(f) := \max_{e \in E} \frac{f(e)}{c(e)},$$

where $f(e)$ is the total flow that crosses e, of all commodities. The congestion of Q in G, $\mathrm{cong}_G(Q)$, *is the minimum possible value of $\mathrm{cong}_G(f)$ over all flows f routing Q.*

For completeness, we restate below the definitions from Section 1 of cut–sparsifiers and flow–sparsifiers. Let $G = (V, E)$ be an undirected graph, and let $H = (V_H, E_H, w_H)$ be a weighted undirected graph with $V \subseteq V_H$.

Definition 2.4 (Cut-Sparsifier)**.** *We say that H is a cut-sparsifier of G of quality α if for every $S \subseteq V$ it holds that*

$$\mathrm{cap}_G(S, V \setminus S) \leq \mathrm{mincut}_H(S, V \setminus S) \leq \alpha \cdot \mathrm{cap}_G(S, V \setminus S),$$

where $\mathrm{mincut}_H(S, V \setminus S) = \min\{\mathrm{cap}_H(U, V_H \setminus U) \mid S \subseteq U,\ V \setminus S \subseteq V_H \setminus U\}$.

Definition 2.5 (Flow-Sparsifier)**.** *We say that H is a flow-sparsifier of G of quality α if for every multi-commodity flow problem Q on the vertex set V, it holds that*

$$\mathrm{cong}_H(Q) \leq \mathrm{cong}_G(Q) \leq \alpha \cdot \mathrm{cong}_H(Q).$$

Definition 2.6 (Respecting Cut)**.** *Let Q be a multi-commodity flow problem on $G = (V, E)$, with demand matrix D_Q. Let $(S, \bar{S})$ be a cut in G.*

We say that $(S, \bar{S})$ α-respects *Q if $\mathrm{cap}(S, \bar{S}) \geq \alpha \cdot \mathrm{dem}_Q(S, \bar{S})$. We say that G* α-respects *Q if*

$$\min_{\substack{S \subseteq V \\ \mathrm{dem}_Q(S,\bar{S}) > 0}} \frac{\mathrm{cap}(S, \bar{S})}{\mathrm{dem}_Q(S, \bar{S})} \geq \alpha.$$

Note that a G α-respecting a demand matrix Q does not necessarily imply that Q can be routed with congestion $1/\alpha$ in G. Due to the flow–cut gap [LR99], an additional $O(\log n)$ factor may be required.

Fact 2.7. *If the multi-commodity flow problem Q can be routed in G with congestion α, then $\mathrm{dem}_Q(B, W) \leq \alpha \cdot \mathrm{cap}_G(S, V \setminus S)$, for all cuts $(S, V \setminus S)$, i.e., G $\frac{1}{\alpha}$-respects Q.*

2.2 Our Contribution

We prove the following theorem:

Theorem 2.8. *Given a graph $G = (V, E)$ of m edges, there exists a randomized algorithm which takes $\tilde{O}(m)$ time and with high probability finds a weighted tree $T = (V_T, E_T, w_T)$ with $V \subseteq V_T$, such that T is a cut-sparsifier of G with quality $O(\log^2 n \log\log n)$.*

However, first, to gradually develop our techniques, we prove a weaker and simpler version of the theorem, in which we look for a tree cut-sparsifier of quality $O(\log^3 n)$:

Theorem 2.9. *Given a graph $G = (V, E)$ of m edges, there exists a randomized algorithm which takes $\tilde{O}(m)$ time and with high probability finds a weighted tree $T = (V_T, E_T, w_T)$ with $V \subseteq V_T$, such that T is a cut-sparsifier of G with quality $O(\log^3 n)$.*

A tree cut-sparsifier of G of quality α is also a tree flow-sparsifier of quality $O(\alpha \log n)$. This follows from the flow-cut gap theorem [LR99, AR98] (stated below), together with the fact that the flow-cut gap in a tree is 1.

Theorem 2.10 (Flow-Cut Gap [AR98])**.** *Let Q be a multi-commodity flow problem on $G = (V, E)$, with demand matrix D_Q. Define $\theta_G(Q) := \sup\{\theta : G \ \theta\text{-respects } Q\}$. Then, for a universal constant $c > 0$,*

$$\frac{c}{\log n} \mathrm{cong}_G(Q) \leq \frac{1}{\theta_G(Q)} \leq \mathrm{cong}_G(Q).$$

Using Theorem 2.10, we get that a tree cut-sparsifier T of G of quality α satisfies, for all flow problems Q,

$$\mathrm{cong}_T(Q) \leq \mathrm{cong}_G(Q) \leq \frac{\log n}{c \cdot \theta_G(Q)} \overset{(1)}{\leq} \frac{\alpha \log n}{c \cdot \theta_T(Q)} \overset{(2)}{=} \frac{\alpha \log n}{c} \mathrm{cong}_T(Q),$$

where Inequality (1) holds since T is a cut-sparsifier of quality α, and Equality (2) holds since the flow-cut gap of a tree is 1. Therefore, T is a flow-sparsifier of G of quality $O(\alpha \log n)$.

Hence, as a corollary of Theorem 2.8, we get:

Theorem 2.11. *Given a graph $G = (V, E)$ of m edges, there exists a randomized algorithm which takes $\tilde{O}(m)$ time and with high probability finds a weighted tree $T = (V_T, E_T, w_T)$ with $V \subseteq V_T$, such that T is a flow-sparsifier of G with quality $O(\log^3 n \log\log n)$, which in particular induces a congestion-approximator of G with quality $O(\log^3 n \log\log n)$ and near-linear size.*

The flow-cut gap for planar graphs is $O(\sqrt{\log n})$ [Rao99]. Therefore, for planar graphs, our construction achieves tree flow-sparsifier (and congestion-approximator) of quality $O(\log^{2.5} n \log\log n)$.

For simplicity, we present our algorithms in the setting where G is unweighted, but our constructions can be adapted to the case of polynomially bounded weights.

2.3 Certifying a Tree Cut-Sparsifier Versus Certifying a Tree Flow-Sparsifier

Our construction follows a recursive partitioning algorithm which takes as input a cluster $S \subseteq V$ and partitions it to sub-clusters $S_1, \ldots, S_t$. Applying this algorithm recursively gives a laminar decomposition of the set V, which naturally corresponds to a tree T. The root of T corresponds to the set V while the leaves correspond to individual vertices $v \in V$. As is standard in tree flow-sparsifier algorithms, the weight of an edge $(S, P(S))$ in T between the node corresponding to the cluster S and its parent $P(S)$ is $\mathrm{cap}_G(S, V \setminus S)$.

Consider first the task of certifying that such a tree T is a flow-sparsifier. We can do this by showing that the congestion of routing a demand matrix Q in T is comparable to the congestion of routing Q in G. Note that by choosing the capacities of the tree edges as we did, the inequality $\mathrm{cong}_T(Q) \leq \mathrm{cong}_G(Q)$ is immediate: we route the multi-commodity flow problem Q in T trivially along the unique path between each pair of vertices. If the flow problem can be routed in G with congestion c, then by Fact 2.7 it must be that, for every sub-cluster $S \in V_T$, the cut $(S, V \setminus S)$ c-respects Q: $\frac{\mathrm{cap}(S, V \setminus S)}{\mathrm{dem}_Q(S, V \setminus S)} \geq c$. The demand across each cut $(S, V \setminus S)$ corresponds exactly to the total amount of flow routed along the edge $(S, P(S))$ in the tree, so this ensures that the flow problem can be routed in the tree with congestion c. Therefore, when building a tree flow-sparsifier, our goal is to choose the tree such that the reverse inequality $\mathrm{cong}_G(Q) \leq \alpha \cdot \mathrm{cong}_T(Q)$ also holds. This means that given a demand matrix Q which can be routed in T we have to show how to route it in G with similar congestion. Q being routable in T means that all cuts $(S, V \setminus S)$, for $S \in V_T$, respect Q. Therefore, the intuition is that we look to include in the tree the sub-clusters which best capture the cut structure of the graph, in the sense that any demand matrix Q respected by them can be routed in G.

In the flow-sparsifier case, given a demand matrix Q which is routable in T, [HHR03, BKR03, RST14] used the tree structure to describe the routing of Q in the graph G. They did this by routing Q which originates at all the vertices of V (corresponding to leaves in T), toward the boundaries of sub-clusters S as they go up along the tree.[2] That is, when a sub-cluster $S \in T$ is partitioned into $\{S_1, \ldots, S_t\}$, they route the mass in G, from the boundary edges of $\{S_i\}_{i=1}^t$ (where they arrived in the previous iteration) to the boundary edges of S. We emphasize that this routing takes place in G, while the structure of T is used only as a "blueprint" to direct the intermediate flows. They then "pair-up" source and target demands of vertex pairs that are both in S before moving further upwards in the tree.

Turning now back to cut-sparsifiers, in order to certify that T is a tree cut-sparsifier, we no longer need to route the demand matrix Q (which can be routed in T) in G, but rather only show that G respects Q (see Claim 5.2 which essentially says that T is a tree cut-sparsifier of quality α if it $1/\alpha$-respects every demand matrix Q that is 1-respected by T). To show that demand matrix Q is respected by G we develop a technique of moving demands rather than flow. For this we define the notion of a *valid demand state* (see Definition 3.13).

In a demand state P each node $v \in V$ has its own vector of $|V|$ commodities (one for each vertex), where for each commodity it has some amount of mass that is waiting at v to be shipped out (or, if the amount is negative, waiting to be shipped into v). This representation is particularly convenient in the hierarchical setting: Even when we focus on a sub-cluster S, we maintain at each $v \in S$ information on the mass from all $|V|$ commodities including those of vertices that are not in S. In contrast, working directly with $|S| \times |S|$ demand matrices within a sub-cluster S would be inconvenient, as the local view would lose information about demands involving vertices outside S. The demand state formulation, on the other hand, always retains the complete global demand

[2] This is made formal using the subdivision graph (see Definition 3.1), where the demands will reside on the split nodes of the boundary edges. For the overview in this section, however, we omit these details for simplicity.

structure, allowing us to reason locally within clusters while still maintaining awareness of the external interactions.

We can then update the demand state by moving mass around, thus reasoning about intermediate steps in the routing scheme. That is, we again use the structure of T to direct the movement of demands. As we climb up the sparsifier T, we move the demands of the demand state P from internal edges to the boundary edges of sub-clusters $S \in V_T$. However, unlike in the flow-sparsifier setting, these movements do not necessarily correspond to an actual multi-commodity flow that is routable in the graph. In fact, we do not "pay" for the congestion of these routings, as in the flow-sparsifier analysis, but rather according to how much the graph respects them (see Definition 2.6). This charging scheme is detailed in Section 5.

2.4 Dealing With Bottleneck Cuts

As mentioned in Section 2.3, a key property of tree flow-sparsifier T is that for every cluster $S \in V_T$, we can move the demands currently in the inter-cluster edges F between sub-clusters $\{S_1, \ldots, S_t\}$ of S, to the boundary edges $B = E(S, V \setminus S)$ of S while incurring small congestion. An ideal scenario is when the algorithm guarantees that $F \cup B$ is *flow-linked* inside $G[S]$, meaning that one can route an all-to-all multi-commodity flow $Q_{F \cup B}$ (see Definition 2.2) between the edges of $F \cup B$ with low congestion. In this scenario, we can transfer the demands from the inter-cluster edges F to the boundary B in a two-step process:

- Spread the demands of $F \cup B$ uniformly by routing them according to $Q_{F \cup B}$.
- Route the demands of F to B according to the flow paths of the sub-matrix $Q_{F \cup B}(F, B)$.

Note that the uniform spreading in the first step ensures that demands whose source and target are in S are routed to the same uniform distribution over $F \cup B$ and are therefore canceled. Moreover, after the second step every boundary edge carries an equal share of the total demand. Because $S \in V_T$, the cut $E(S, V \setminus S)$ must respect Q, so the amount of mass that has to leave the cluster S is at most $E(S, V \setminus S)$. Therefore, the mentioned cancellation guarantees that the load (total demand per edge) is at most 1 on each boundary edge.

However, as discussed in [BKR03, HHR03, RS14, RST14], there may be sub-clusters $S \subseteq V$, for which we cannot find a partition $\{S_i\}_{i=1}^{t}$ such that $F \cup B$ is flow-linked in $G[S]$. A simple counter-example is that it might be the case that B itself is not flow-linked in $G[S]$, or maybe even disconnected there. In this case, no partition of $G[S]$ will work. We use the name *bottleneck cut* to refer to a cut which certifies that the boundary edges are not flow-linked. This name was introduced by [RS14], though their bottleneck cuts only had to satisfy a weaker requirement. A similar (yet not equivalent) notion appeared earlier in [BKR03, HHR03]. The algorithms of [BKR03, HHR03, RS14] had to make sure that the clusters have no bottleneck cuts. They achieved this by cutting clusters that have bottleneck cuts. We call this step the *refinement phase* [RS14]. Previous work implement the refinement phase in different ways, either by ensuring a "precondition" on the sub-clusters they recur into [BKR03], or encountering these cuts and handling them "on the fly" [HHR03, RS14]. In either case dealing with these cuts slowed down these early algorithms substantially.

Since ensuring that there is no bottleneck cut in each sub-cluster before recurring into it may be expensive (in terms of running time), [RST14] settled for something slightly weaker: They first ignore the boundary edges of S and find a partition of S into $\{S_1, \ldots, S_t\}$ such that the inter-cluster edges F are flow-linked. Then they find an approximate min-cut Y for the (single commodity) flow problem which routes flow from F to B. By adding to T a new sub-cluster whose boundary is Y, [RST14] are able to bound the amount of mass that has to cross the cut Y, since now the demand matrix Q is respected by Y in G.

Adding Y as a boundary of a sub-cluster in T is enough to show how to move the demands to the boundary edges B, such that the load on each boundary edge is multiplied by $1+\tau$ units (τ is a parameter $0 < \tau \leq 1$). However, the distribution of demands on the boundary edges is not uniform. Hence, demands whose source and target are in the sub-cluster do not cancel-out. Consequently, the load accumulates when going up along the tree, multiplied by a factor of at most $1+\tau$ in each iteration. Choosing $\tau = \frac{1}{\log n}$, such that the load is multiplied by a factor of $(1+\frac{1}{\log n})$ along the $O(\log n)$ levels of the tree, ensures that the load is constant. This method, however, incurs a degradation of $O(\frac{1}{\tau})$ per level in the quality of the tree flow-sparsifier, leading to an overall loss of $O(\frac{\log n}{\tau}) = O(\log^2 n)$.[3] The additional $O(\log^2 n)$ factor in their quality comes from the usage of [KRV09]'s cut-matching game for flow (so in total the quality is $O(\log^4 n)$).

In our first step below (Section 2.5) we deal with bottleneck cuts similarly to [RST14], but using primitives from the expander decomposition algorithm in [ADK25], which are the cut-analogue of [KRV09]'s cut-matching game. This allows us to get Theorem 2.9. Afterwards, in Section 2.6 we describe a better way to deal with bottleneck cuts, by implementing a version of the refinement phase in near-linear time. This allows us to improve the quality of the resulting tree cut-sparsifier and get Theorem 2.8.

2.5 A Tree Cut-Sparsifier of Quality $O(\log^3 n)$

We now adapt [RST14] to the *cut-sparsifier* setting. As outlined in Section 2.3, our goal here is not to route the demand matrix Q within G, but rather to ensure that G respects Q. In this formulation, the analogue of a set F of inter-cluster edges (within a cluster S) being flow-linked in [RST14] is that $G[S]$ respects the all-to-all demand matrix between edges of F (see Definition 2.2), or equivalently that F expands in $G[S]$ (see Definition 3.5 below). Therefore, our first goal is to obtain, efficiently, such a set F. For this step we ignore the boundary edges of S (which may be separated by bottleneck cuts). We use the following algorithmic tool:

Balanced-cut-or-expander procedure. To obtain such an F efficiently we use the "balanced cut or expander" primitive from [ADK25], which allows us to certify that the sub-cluster $G[S]$ respects the all-to-all flow problem on a subset $F \subseteq E$, or find a balanced cut which refutes this. We start with $F_1 = E(G[S])$ and iterate on $i = 1, \ldots$:

- Invoke the primitive on $G[S]$ to check if it respects the all-to-all flow problem on F_i.
- On an *expander* outcome, we stop and set $F := F_i$. This certifies the all-to-all flow problem on F is respected by $G[S]$.
- On a *balanced cut* outcome, we shrink F_i to a new set $F_{i+1} \subset F_i$. The sparsity guarantee ensures $|F_{i+1}| \leq (1 - 1/\log n)|F_i|$, so after $O(\log^2 n)$ iterations we must obtain the first case.

This step is the cut-analogue of [RST14, Lemma 3.1], but powered by [ADK25]'s tool so that the certification we obtain when F is good is expansion (according to Definition 3.5) rather than being flow-linked.

Routing from F to the boundary. Once F is fixed, we connect F to the boundary edges similarly to [RST14]. We compute a cut Y (via an auxiliary approximate flow problem) that separates F from B. For this purpose, we use the fair-cuts algorithm of [LNPS23], which serves as

[3] Choosing $\tau = \frac{1}{\log n}$ is optimal, up to a multiplicative constant. Let $k = \Theta(\log n)$ be the depth of the tree. We omit the proof of the following claim: For any sequence of $\tau_1, \ldots, \tau_k \leq 1$, it holds that $\sum_{i=1}^{k} \frac{1}{\tau_i} \prod_{j=i+1}^{k} (1+\tau_j) = \Omega(k^2)$.

an efficient approximate max-flow solver and provides slightly better congestion (see Lemma 4.2) compared to [RST14, Lemma 3.2], as well as simplifying the argument.

The usage of [ADK25] rather than a procedure inspired by the cut-matching game of [KRV09] in [RST14] allows us to improve the quality of the cut-sparsifier by a $\log n$ factor. Specifically, we still pay $O(\frac{\log n}{\tau}) = O(\log^2 n)$ like in Section 2.4, but the additional factor from the cut-matching component is now $O(\log n)$ instead of $O(\log^2 n)$. This gives a tree cut-sparsifier of quality of $O(\log^3 n)$, and establishes Theorem 2.9. In particular, as discussed after Theorem 2.9, a tree cut-sparsifier of quality α implies a tree flow-sparsifier of quality $O(\alpha \log n)$. Therefore, this tree is also a tree flow-sparsifier of quality $O(\log^4 n)$, matching the quality of the flow-sparsifier of [RST14].

The construction of the tree cut-sparsifier is detailed in Section 4. Importantly, we use the modular building blocks of the balanced sparse cut procedure [ADK25] and the fair-cuts algorithm [LNPS23] as a black box. This significantly simplifies the construction of the tree cut-sparsifier, when compared to, *e.g.*, [RST14], as it abstracts away many details of the underlying algorithmic tools.

In Section 5 we show that the constructed tree is indeed a tree cut-sparsifier. There, we explicitly define the invariant that holds when we progressively move the demands around the graph (following the structure of the tree). Additionally, we use a careful charging scheme to rigorously bound the quality of the tree cut-sparsifier.

2.6 A Tree Cut-Sparsifier of Quality $O(\log^2 n \log\log n)$

Following the naming of [HHR03, RS14], we call the partitioning algorithm from Section 2.5, that takes S and returns $\{S_i\}_{i=1}^t$, the *merge phase*. As explained in Section 2.5, this phase does *not* guarantee that the inter-cluster edges F and the boundary edges B are expanding together, just that the edges of F are expanding, and we can route from F to B. Recall from Section 2.4 that because the distribution of demands on the boundary edges is not uniform, source and target demands do not cancel-out, so the load on the boundary edges accumulates as we go up along the tree. This degrades the quality of the tree cut-sparsifier by an $O(\log n)$ factor. To get the improved Theorem 2.8, in Section 6 we show how to implement a version of the *refinement phase* (see Section 2.4) in near-linear time. Specifically, we show that given a cluster S we can partition it to sub-clusters $\{S_1, \ldots, S_t\}$ such that each sub-cluster respects the all-to-all multi-commodity flow problem on its boundary edges. This is a slightly weaker requirement than the precondition of [BKR03] and the requirement of [HHR03, RS14]. Additionally, we show that we can route from the inter-cluster edges of this refinement partition to the boundary of S. To perform this partition in near-linear time, we again use the result of [ADK25] which allows us to either find a balanced sparse cut (with respect to the all-to-all flow problem on the boundary of the current sub-cluster) or ensure that the boundary edges are expanding. Additionally, it lets us route from the sparse cut's edges to the boundary edges of one of the sides of the cut. Crucially, however, contrary to the merge phase, we get no guarantee that the size $|S_i|$ is smaller than $|S|$ by a constant factor (and in fact it may be that the partition is simply $\{S\}$).

Using the refinement phase, we amend the hierarchical decomposition tree by performing this refinement after each call to the merge phase algorithm. This gives a decomposition which alternates between a refinement phase partition, that allows to spread the demand uniformly on the boundary, and a merge phase partition that shrinks the clusters. Because each sub-cluster S_i in the refinement phase respects the all-to-all demand matrix on its boundary edges, we can move the demands to a uniform distribution on these boundary edges. Furthermore, since $S_i \in V_T$ (and by assumption Q is routable in T), the cut $E(S_i, V \setminus S_i)$ respects Q, we get that at most 1 unit of mass per edge needs to leave the cluster. This allows us to "reset" the load on each boundary edge to be 1 and

avoid accumulating the loads on the boundary edges as we go up the tree. By setting $\tau = 1$ in the merge phase, we improve the degradation (in quality) incurred by the merge phase by a factor of $O(\log n)$, to be $O(\log^2 n)$ (in total over all sub-clusters), instead of $O(\log^3 n)$. This is because we pay $O(\frac{\log n}{\tau}) = O(\log n)$, plus an additional $O(\log n)$ term for the cut-matching component like before. As for the degradation in quality incurred by the refinement phase, we adapt the analysis of [HHR03, RS14] to show that it is bounded by $O(\log^2 n \log\log n)$ (in total over all sub-clusters), which gives Theorem 2.8. In particular, our tree is also a tree flow-sparsifier of quality $O(\log^3 n \log\log n)$, which gives Theorem 2.11.

The rest of the paper is structured as follows. Section 3 contains additional definitions and notations, as well as some algorithmic tools we use. In Section 4 we detail the merge phase partition and the hierarchical decomposition for Theorem 2.9. In Section 5 we show how to update the demand states as we go up the tree in each application of the merge phase, and show that the resulting tree indeed suffices for Theorem 2.9. In Section 6 we detail the refinement phase and use it to prove Theorem 2.8.

3 Additional Preliminaries

3.1 General Definitions

We begin with defining the subdivision graph, which allows us to formally reason about the edges of a graph carrying demand and expanding.

Definition 3.1 (Subdivision graph)**.** *Given a graph $G = (V, E)$, the* subdivision graph *is denoted by $G' = (V', E')$. It is created by adding a* split vertex *in the middle of each edge, replacing it with a path of length 2. Formally, $V' := V \cup X_E$, where X_E are the split vertices $X_E := \{x_e \mid e \in E\}$. The new edges are $E' := \{(x_e, v) \mid v \in e\}$. Given a set of edges $F \subseteq E$, the corresponding split vertices are denoted by $X_F := \{x_f \mid f \in F\}$.*

Definition 3.2 (Indicator Measure)**.** *Let $G = (V, E)$ be a graph and let $U \subseteq V$ be a subset. We define the* weighting induced by U *as $\mu_U : V \to \mathbb{R}_{\geq 0}$, such that $\mu_U = \mathbb{1}_U$. In other words, for each $v \in V$, $\mu_U(v) = 1$ if $v \in U$, and 0 otherwise.*

In almost all of our use cases, we will only use such indicator measures. In these cases, one should think of the measure μ as a way of distinguishing a subset of the vertices.

3.2 Expansion

The following definition extends the standard notion of conductance to the vertex-weighted setting, where some vertices are more important than others. Specifically, when μ is an indicator measure, it lets us focus on a subset of the vertices (for example, those corresponding to flow terminals).

Definition 3.3 (Expansion with Vertex Weights)**.** *Let $G = (V, E)$ and $\emptyset \neq S \subset V$. Let $\mu : V \to \mathbb{R}_{\geq 0}$ be a weighting of the vertices. The expansion of the cut $(S, \bar{S})$ with respect to μ (sometimes called μ-expansion), denoted by $\Phi^{\mu}_G(S, \bar{S})$, is defined as*

$$\Phi^{\mu}_G(S, \bar{S}) := \frac{\text{cap}(S, \bar{S})}{\min(\mu(S), \mu(\bar{S}))},$$

whenever $\min(\mu(S), \mu(\bar{S})) > 0$. A cut with low μ-expansion is called μ-sparse *(the exact bound on the expansion is given in the context).*

The expansion of G with respect to μ (sometimes called μ-expansion) is defined to be

$$\Phi^{\mu}(G) := \min_{\substack{S \subseteq V \\ \min(\mu(S),\mu(\bar{S}))>0}} \Phi^{\mu}_{G}(S, \bar{S})$$

The next definition introduces the corresponding notion of an expander, *i.e.*, a graph that maintains sufficiently large expansion across all vertex subsets.

Definition 3.4 (Expander)**.** *Let $G = (V, E)$ and let μ be a weighting of the vertices. We say that G is an α-expander with respect to μ if $\Phi^{\mu}(G) \geq \alpha$.*

So far, expansion has been defined as a *property of a graph*. In many of our later arguments, however, we talk about *individual subsets* $A \subseteq V$ that expand well *within* a given graph G under some weighting. The next definition captures this idea: it says that a subset A *α-expands with respect to μ* if no cut of V separates A too unevenly, when the imbalance is measured according to the μ-measure of the vertices. This notion generalizes over the α-cut-linked notion of [CKS05].

Definition 3.5 (Subset of V Expands)**.** *Given a graph $G = (V, E)$, $\alpha > 0$, a weighting $\mu : V \to \mathbb{R}_{\geq 0}$ of the vertices, and a set of vertices $A \subseteq V$, we say that A α-expands with respect to μ in G if*

$$\forall_{\substack{S \subseteq V \\ \min(\mu(A \cap S),\mu(A \setminus S))>0}} : \frac{\mathrm{cap}(S, V \setminus S)}{\min(\mu(A \cap S), \mu(A \setminus S))} \geq \alpha.$$

In other words, if we let $\mu' = \mu \cdot \mu_A$, then $A \subseteq V$, α-expands in G with respect to μ, if and only if G is an α-expander with respect to μ'.

When $\mu \equiv 1$, we just say that A α-expands in G.

Note that in case $\mu = \mathbb{1}_X$ is an indicator measure, some set A is α-expanding with respect to $\mathbb{1}_X$ is equivalent to saying that the set $A \cap X$ is α-expanding. Definition 3.5 is the cut-analogue of the set A being α-flow-linked, in case each vertex $v \in A$ injects $\mu(v)$ units of flow. This approach is useful for us when we want to choose a measure $\mu = \mathbb{1}_X$ in advance and *then* find some set A such that $A \cap X$ expands.

The next remark bridges the two viewpoints introduced in Definitions 2.6 and 3.5: the notion of a set A that α-expands with respect to a measure μ, and the notion of a graph G that α-respects a corresponding flow problem defined by μ.

Remark 3.6. *Let $G = (V, E)$ and let μ be a weighting of the vertices. Let $A \subseteq V$ be a set of vertices with $\mu(A) > 0$ and let $\mu' = \mu \cdot \mu_A$. Consider the demand matrix Q given by $Q(u, v) = \frac{\mu'(u) \cdot \mu'(v)}{2\mu(A)}$ for each $u, v \in V$. Fix a cut $(S, \bar{S})$. It holds that $\mathrm{dem}_Q(S, \bar{S}) = \frac{\mu'(S) \cdot \mu'(\bar{S})}{\mu(A)}$. Assume $\mu'(S) \leq \mu'(\bar{S})$. Then, $\frac{1}{2} \leq \frac{\mu'(\bar{S})}{\mu(A)} \leq 1$. Plugging in,*

$$\Phi^{\mu'}(S, \bar{S}) \leq \frac{\mathrm{cap}(S, \bar{S})}{\mathrm{dem}_Q(S, \bar{S})} \leq 2 \cdot \Phi^{\mu'}(S, \bar{S}).$$

In other words, if A α-expands in G with respect to μ, then it certifies that G α-respects the demand matrix Q given above, and conversely if G α-respects Q, then A $\frac{\alpha}{2}$-expands with respect to μ in G.

In particular (by setting $\mu \equiv \mathbb{1}$), up to constant factors, A α-expands in G is equivalent to G α-respecting the all-to-all flow problem on A (recall Definition 2.2).

The following fact shows that if $G[A]$ is an expander with respect to a measure, then A expands in G with respect to this measure. Note that the converse is not true. The flow analogue of this claim is that if we can route an all-to-all flow problem on the vertices of A within $G[A]$ then we can do so in G, but not conversely.

Fact 3.7. *If $G[A]$ is an α-expander with respect to $\mu|_A$, then A α-expands with respect to μ in G.*

Proof. Indeed, let $S \subseteq V$ be such that $\min(\mu(A \cap S), \mu(A \setminus S)) > 0$. Then,

$$\frac{\mathrm{cap}(S, V \setminus S)}{\min(\mu(A \cap S), \mu(A \setminus S))} \geq \frac{\mathrm{cap}(A \cap S, A \setminus S)}{\min(\mu(A \cap S), \mu(A \setminus S))} \geq \alpha \ ,$$

where the last inequality is due to $(A \cap S, A \setminus S)$ being a cut in $G[A]$. □

The following fact shows that expansion is a monotone property, in the sense that subsets of an expanding sets also expand.

Fact 3.8. *Assume $A_1 \subseteq A_2 \subseteq V$ and A_2 α-expands with respect to μ. Then A_1 α-expands with respect to μ.*

Proof. The denominator satisfies $\min(\mu(A_1 \cap S), \mu(A_1 \setminus S)) \leq \min(\mu(A_2 \cap S), \mu(A_2 \setminus S))$, which gives the result. □

3.3 Cut-Matching Game

The following is a recent result of [ADK25], which generalizes [ADK23]. They show that in near-linear time we can get a balanced sparse cut, or certify that a large part of the graph is an expander, with respect to a general vertex measure.[4]

Theorem 3.9 ([ADK25, Theorem 5.2])**.** *Given a graph $G = (V, E)$ of m edges, a parameter $\phi > 0$, and a weighting $\mu : V \to \mathbb{R}_{\geq 0}$ of the vertices, there exists a randomized algorithm which takes $\tilde{O}(m)$ time and must end in one of the following three cases:*

1. *We certify that G has μ-expansion $\Phi^{\mu}(G) = \Omega(\phi)$, with high probability.*
2. *We find a cut $(A, \bar{A})$ in G of μ-expansion $\Phi^{\mu}_G(A, \bar{A}) = O(\phi \log n)$, and $\mu(A), \mu(\bar{A})$ are both $\Omega\left(\frac{\mu(V)}{\log n}\right)$, i.e, we find a relatively balanced μ-sparse cut.*
3. *We find a cut $(A, \bar{A})$ with $0 < \mu(\bar{A}) \leq \frac{\mu(V)}{2}$, $\Phi^{\mu}_G(A, \bar{A}) = O(\phi \log n)$, and with high probability, $G[A]$ is an $\Omega(\phi)$-expander with respect to μ.*

Routing from a cut. Let $G = (V, E)$, and let $(A, S \setminus A)$ be a cut of a cluster $S \subseteq V$. When we say that we *route flow in $G[A]$ from the cut* $(A, S \setminus A)$ to some destination subset $X \subseteq A$, we mean that there exists a flow f in the subgraph $G[A]$ such that each vertex $v \in A$ sends $\deg_{G[S]}(v) - \deg_{G[A]}(v)$ units of mass, i.e., one unit for every incident edge of v crossing the cut $(A, S \setminus A)$. The specific requirements on the destination (*i.e.*, the amount of mass that each vertex $x \in X$ receives), and congestion of this flow are stated in the corresponding context.

[4] Theorem 3.9 requires that $\mu(v)$ will be in $\left[\frac{1}{\mathrm{poly}(n)}, \mathrm{poly}(n)\right] \cup \{0\}$, which will hold for our usages.

Remark 3.10. *The authors of [ADK25] supply additional guarantees in Cases (2) and (3) of Theorem 3.9 as follows. First, in both cases* $\mu(A) \geq \frac{1}{4}\mu(V)$.

Additionally, in Case (2), there exist constant $c > 0$ *and a sequence of cuts* $S_t \subseteq A_t$, $0 \leq t < T'$ *with* $T' = O(\log^2 n)$, *such that* $A_0 = V$, $A_{t+1} = A_t \setminus S_t$ *and* $\mu(\bigcup_{t=0}^{T'-1} S_t) = O(\frac{\mu(V)}{\log n})$. *Each* S_t *is sparse:* $\Phi^{\mu}_{G[A_t]}(S_t, A_{t+1}) \leq c\phi \log n$. *The returned cut is simply* $\bar{A} = \bigcup_t S_t$. *Moreover, [ADK25, Corollary A.3] gives*

- *There exists a flow routable with congestion* $O(1)$ *in* $G[S_t]$, *such that each cut edge of* $E(S_t, A_t \setminus S_t)$ *is a source of one unit of flow and each vertex of* S_t *receives at most* $O(\phi \log n \cdot \mu(v))$ *units.*
- *There exists a flow routable with congestion* $O(1)$ *in* $G[A_t \setminus S_t]$, *such that each cut edge of* $E(S_t, A_t \setminus S_t)$ *is a source of one unit of flow and each vertex of* $A_t \setminus S_t$ *receives at most* $O(\phi \log n \cdot \mu(v))$ *units.*

In other words, in each step we can route flow from the cut edges $E(S_t, A_t \setminus S_t)$ *to either the side* S_t *or* $A_t \setminus S_t$.

In Case (3), this sequence of cuts exist as well, but they run a "trimming step" to get some larger cut $\bar{A} \supseteq \bigcup_t S_t$ *which is returned. Additionally, [ADK25, Corollary A.5] gives*

- *There exists a flow routable with congestion* $O(1)$ *in* $G[A]$, *such that each cut edge of* $E(A, \bar{A})$ *is a source of one unit of flow and each vertex of* A *receives at most* $O(\phi \cdot \mu(v))$ *units.*
- *There exists a flow routable with congestion* $O(1)$ *in* $G[\bar{A}]$, *such that each cut edge of* $E(A, \bar{A})$ *is a source of one unit of flow and each vertex of* $\bar{A}$ *receives at most* $O(\phi \log n \cdot \mu(v))$ *units.*

When we use Theorem 3.9, when Case (3) occurs, we sometimes prefer to have the guarantee that $\mu(\bar{A})$ is significantly smaller than $\mu(A)$. This can be done by simply terminating with Case (2) if $\mu(\bar{A})$ and $\mu(A)$ are balanced. This is captured by the following Corollary, which is a straightforward strengthening of Theorem 3.9. The only difference is the bound on $\mu(\bar{A})$ in Case (3).

Corollary 3.11 ([ADK25, Corollary 5.3])**.** *Given a graph* $G = (V, E)$ *of* m *edges, a parameter* $\phi > 0$, *and a weighting* $\mu : V \to \mathbb{R}_{\geq 0}$ *of the vertices, there exists a randomized algorithm which takes* $\tilde{O}(m)$ *time and must end in one of the following three cases:*

1. *We certify that* G *has* μ*-expansion* $\Phi^{\mu}(G) = \Omega(\phi)$, *with high probability.*
2. *We find a cut* $(A, \bar{A})$ *in* G *of* μ*-expansion* $\Phi^{\mu}_G(A, \bar{A}) = O(\phi \log n)$, *and* $\mu(A), \mu(\bar{A})$ *are both* $\Omega\left(\frac{\mu(V)}{\log n}\right)$, *i.e, we find a relatively balanced* μ*-sparse cut.*
3. *We find a cut* $(A, \bar{A})$ *with* $0 < \mu(\bar{A}) \leq \frac{\mu(V)}{\log n}$, $\Phi^{\mu}_G(A, \bar{A}) = O(\phi \log n)$, *and with high probability,* $G[A]$ *is an* $\Omega(\phi)$*-expander with respect to* μ.

Proof. If Theorem 3.9 terminated with Case (3) but $\frac{\mu(V)}{\log n} < \mu(\bar{A}) \leq \frac{\mu(V)}{2}$, we simply terminate with Case (2) instead. □

We emphasize that in Corollary 3.11, some of the flows given by Remark 3.10 in Case (2) may no longer exist. Specifically, in Case (2) the cut is no longer given directly as $\bar{A} = \bigcup_t S_t$, because the trimming step was applied on $\bigcup_t S_t$.

Remark 3.12. *In both Theorem 3.9 and Corollary 3.11, in Case (3), the fact that* $G[A]$ *is an* $\Omega(\phi)$*-expander with respect to* μ, *implies that* A $\Omega(\phi)$*-expands with respect to* μ *in* G *by Fact 3.7.*

3.4 Demand States

When we describe the solution to a multi-commodity flow problem, it will be useful to talk about *intermediate* stages of the flow. Analogously, we wish to talk about intermediate states of the distribution of demands when showing that a graph respects a given demand matrix. Intuitively, we imagine the vertices of the graph holding some mass (*demand*) of each commodity, which can be positive or negative. These demands move from one vertex to another according to demand matrices (*i.e.*, multi-commodity flow problems), incurring some congestion (or, in the cut-sparsifier case, *charge*, as detailed in Section 5) on the edges of the graph. When a positive demand meets a negative demand of the same commodity, they cancel out. This scheme is made formal in the definitions in this section.

Definition 3.13 (Demand State)**.** *Consider a set of commodities $\mathcal{K}$. We consider for each vertex $v \in V$ a* demand state on v*, which is a vector $P(v) \in \mathbb{R}^{\mathcal{K}}$, where $P(v,k)$, which may be either positive or negative, defines the amount of commodity $k \in \mathcal{K}$ which resides at $v \in V$ and needs to be shipped out. We denote by $\|P(v)\|_1 = \sum_{k\in\mathcal{K}} |P(v,k)|$ the* load *on v.*

A demand state on G *(or simply a* demand state *when G is clear from context) is a collection of demand states on each $v \in V$, i.e., $P \in \mathbb{R}^{V\times\mathcal{K}}$, where $P(v,\cdot) := P(v)$ is v's demand state.*

For a set $A \subseteq V$, we say that a demand state P on G is supported on A *if for all $k \in \mathcal{K}$, $P(v,k) = 0$ whenever $v \notin A$.*

As explained below in Remark 3.15, demand states generalize demand matrices. Unlike demand matrices (see Definition 2.1), where each vertex serves as the source of its own commodity and source-sink pairs are explicitly encoded in the matrix, in a demand state the commodities are independent of the vertices, and multiple vertices may act as sources of the same commodity (similarly, the same vertex may act as a source of multiple commodities).

Intuitively, we think of a demand state as the current state of mass residing at the vertices of the graph, where positive and negative demands "await" to be canceled-out, while a multi-commodity flow problem and its demand matrix describe an explicit *movement* of mass between pairs of vertices.

Definition 3.14 (Valid Demand State)**.** *We say that a demand state on G is* valid *if $\sum_{u\in V} P(u,k) = 0$ for each $k \in \mathcal{K}$. That is, in total, no mass of commodity k is created or destroyed.*

The demand *of P across a cut $(A, B = \bar{A})$ is*

$$\mathrm{dem}_P(A,B) := \sum_{k\in\mathcal{K}} |\sum_{u\in A} P(u,k)| = \sum_{k\in\mathcal{K}} |\sum_{u\in B} P(u,k)|.$$

We extend the notion of a cut (resp., graph) respecting a demand matrix, to a cut (resp., graph) respecting valid demand state using this definition: A cut $(S,\bar{S})$ in G α-respects P if $\mathrm{cap}(S,\bar{S}) \geq \alpha \cdot \mathrm{dem}_P(S,\bar{S})$. G α-respects P if

$$\min_{\substack{S\subseteq V \\ \mathrm{dem}_P(S,\bar{S})>0}} \frac{\mathrm{cap}(S,\bar{S})}{\mathrm{dem}_P(S,\bar{S})} \geq \alpha.$$

Remark 3.15. *Note that the notion of a valid demand state generalizes the notion of a demand matrix from Definition 2.1. Indeed, a demand matrix is a $V \times V$ matrix Q where $Q(u,v)$ is the amount of u's commodity that has to go to v. Such a specification corresponds to a valid demand state P in which $\mathcal{K} = V$, and the amount shipped out of u of u's commodity is $P(u,u) =$*

$\sum_{v \in V \setminus \{u\}} Q(u,v)$, *and the amount shipped to* v *of* u*'s commodity is* $P(v,u) = -Q(u,v)$, $v \neq u$. *The standard definitions of congestion and demand across a cut of* Q *equal those of* P.

On the other hand, given a valid demand state, we can define the corresponding multi-commodity flow problem (according to Definition 2.1) by adding to the graph a source vertex and sink vertex corresponding to each commodity, with appropriate edge capacities according to the demands in P. *This reduction will not be used in this paper, so we omit the full details.*

Definition 3.16 (Sum and Negation)**.** *We define the sum of two demand states* P_1 *and* P_2 *on the same set of commodities* $\mathcal{K}$ *to be* $(P_1 + P_2)(u,k) = P_1(u,k) + P_2(u,k)$. *Note that if* P_1 *and* P_2 *were valid demand states, then* $P_1 + P_2$ *is also a valid demand state.*

We define the negation $-P$ *of a demand state* P *on the set of commodities* $\mathcal{K}$ *by* $(-P)(u,k) = -P(u,k)$. *Note that if* P *was a valid demand state, then* $-P$ *is also a valid demand state.*

The following fact, which we use extensively, is a direct consequence of Definitions 3.14 and 3.16:

Fact 3.17. *Fix a cut* $(S, \bar{S})$. *For two valid demand states* P_1, P_2 *on the commodities* $\mathcal{K}$, *we have* $\mathrm{dem}_{P_1+P_2}(S,\bar{S}) \leq \mathrm{dem}_{P_1}(S,\bar{S}) + \mathrm{dem}_{P_2}(S,\bar{S})$.

Additionally, if G α_1*-respects* P_1 *and* α_2*-respects* P_2, *then it* $\frac{1}{\frac{1}{\alpha_1}+\frac{1}{\alpha_2}}$*-respects* $P_1 + P_2$.

Suppose we have a demand state P. Given a demand matrix Q (recall Definition 2.1 of a multi-commodity flow problem), we can "move" the demand vectors in P according to the requirements of Q. Each vertex $u \in V$ will send $Q(u,v)$ units in total to $v \in V$, and we split the demands evenly according to the distribution of commodities in the demand vector $P(u)$. In other words, $\frac{P(u,k)}{\|P(u)\|_1} \cdot Q(u,v)$ units of commodity $k \in \mathcal{K}$ will be shipped to v. If $P(u,k) < 0$ this corresponds to shipping units of commodity k from v to u. This results in a demand state $P^{\uparrow Q}$, called the updated demand state:

Definition 3.18 (Updated Demand State)**.** *Let* P *be a (not necessarily valid) demand state on the set of commodities* $\mathcal{K}$ *and let* Q *be a demand matrix (i.e., a multi-commodity flow problem). We define the* updated *demand state* $P^{\uparrow Q}$ *on* $\mathcal{K}$ *as follows:*

$$P^{\uparrow Q}(u,k) = P(u,k) - \frac{P(u,k)}{\|P(u)\|_1} \cdot \sum_{v \in V} Q(u,v) + \sum_{v \in V} \frac{P(v,k)}{\|P(v)\|_1} \cdot Q(v,u).$$

Example. Suppose the demand state of a vertex u is $P(u) = (20, -70, 10)$, representing three commodities. If a flow problem Q routes 30 units from u to a vertex v, then the updated demand state moves $30 \times (0.2, -0.7, 0.1)$ units of demand from u to v, leaving u with a demand state of $P^{\uparrow Q}(u) = (14, -49, 7)$.

We will make frequent use of the following fact. It states that because we only move the mass around, no mass of any commodity is created or destroyed. This means that the total mass of each commodity is the same in P and $P^{\uparrow Q}$, so it is zero in $P - P^{\uparrow Q}$. Therefore, $P - P^{\uparrow Q}$ is a valid demand state. The full proof of the fact is deferred to Appendix A.

Fact 3.19. *In the notation of Definition 3.18:*

1. $P - P^{\uparrow Q}$ *is a valid demand state.*
2. *For any cut* $(S, \bar{S})$ *of* V, $\mathrm{dem}_{(P - P^{\uparrow Q})}(S,\bar{S}) \leq \mathrm{dem}_Q(S,\bar{S})$.

Remark 3.20. *Let* P *be a (not necessarily valid) demand state on the set of commodities* $\mathcal{K}$. *Of special interest are the following cases:*

1. *Q is a scaled all-to-all flow on some subset $A \subseteq V$: $D_Q(u,v) = \frac{\|P(u)\|_1}{|A|}$ if $u, v \in A$, and otherwise $D_Q(u,v) = 0$.* I.e., *instead of sending 1 unit out of u in total, like in the all-to-all case (see Definition 2.2), we send $\|P(u)\|_1$ units. In this case, because the demand vector $P(u)$ for $u \in A$, was spread uniformly among all vertices of A, the new demand vector of each vertex, $P^{\uparrow Q}(v)$ is the same for all $v \in A$ (i.e., it does not depend on v). Specifically, $P^{\uparrow Q}(v) = \frac{1}{|A|}\sum_{u \in A} P(u)$ for $v \in A$. This corresponds to the fact that mass was distributed uniformly.*

2. *Q moves demands from a set $A \subseteq V$ to $V \setminus A$: For any $u \in A$, $D_Q(v,u) = 0$ for all $v \in V$, and also $\sum_{v \in V \setminus A} D_Q(u,v) = \|P(u)\|_1$. In this case, note that $P^{\uparrow Q}$ is supported on $V \setminus A$.*

4 The Merge Phase

In this section we detail our algorithm for constructing the hierarchical decomposition for Theorem 2.9, whose main partitioning step will be an important tool for the proof of Theorem 2.8 as well.

Our construction closely follows that of [RST14]. For the readers familiar with [RST14], we mention the main differences:

1. The "first partition" step (Theorem 4.1) is translated to the cut-sparsifier case (which improves the quality of the cut-sparsifier by a factor of $\log n$), and its proof is significantly simplified, as the "heavy algorithmic tools" of a *balanced cut or expander* theorem [ADK25] and a *fair cuts* computation [LNPS23] are used in a black box manner.

2. The proof of the main lemma used in the second partition step (Lemma 4.6) is improved and simplified by using the results of [LNPS23].

As explained in Section 2, the hierarchical decomposition is constructed by recursively applying the "merge phase" partitioning procedure to input clusters $S \subseteq V$. Each input cluster $S \subseteq V$ is partitioned in two levels. At the first level we define the direct children of S to be disjoint clusters L and R such that $L \cup R = S$, and at the second level the children of L and of R, respectively, are defined to be disjoint clusters $L_1, \ldots, L_\ell$ and $R_1, \ldots, R_r$ (satisfying $L_1 \cup \cdots \cup L_\ell = L$ and $R_1 \cup \cdots \cup R_r = R$). The size of the clusters L_i and R_i is bounded as $|L_i| \leq \frac{2}{3}|S|$ and $|R_i| \leq \frac{2}{3}|S|$.

The two-level partitioning is produced in two steps. First (Section 4.1), we define a partitioning of the cluster S into disjoint sets $Z_1, \ldots, Z_z$, where each cluster satisfies $|Z_i| \leq \frac{2}{3}|S|$. Next (Section 4.2), we partition S into two disjoint sets L, R. The second-level clusters $L_1, \ldots, L_\ell$ and $R_1, \ldots, R_r$ are defined as $L_i = L \cap Z_i$ and $R_i = R \cap Z_i$, taking only non-empty intersections (see Figure 1).

The intuition behind the two-level partition is as follows. As discussed in Section 2, we would want to partition S into sub-clusters $\{S_1, \ldots, S_t\}$ of size $|S_i| \leq c \cdot |S|, c < 1$, so that the inter-cluster edges F and the boundary edges B of S, satisfy that $F \cup B$ expands. However, this may not be possible for a cluster S if B does not expand in $G[S]$. Consequently, the first partition into $Z_1, \ldots, Z_z$ ignores B and focuses on reducing the size of the clusters, while ensuring that F, the set of inter-cluster edges, expands. In Section 5, we describe how to move the demands that need to leave each cluster $S \in V_T$, to the cluster's boundary edges. Specifically, we wish to move demands that reside on F (*i.e.*, boundary edges of sub-clusters) to B. Therefore, we use a *fair-cut* computation [LNPS23] to find an approximate min-cut, $Y \subseteq E$, that blocks the flow problem of routing from F to B. We then add this cut to the tree (*i.e.*, find a partition $L \cup R$ induced by Y), thereby forcing the given demand matrix to respect this cut. This allows us to bound the amount of mass that has to cross the cut Y. See Section 5 for details on the usage of this partition.

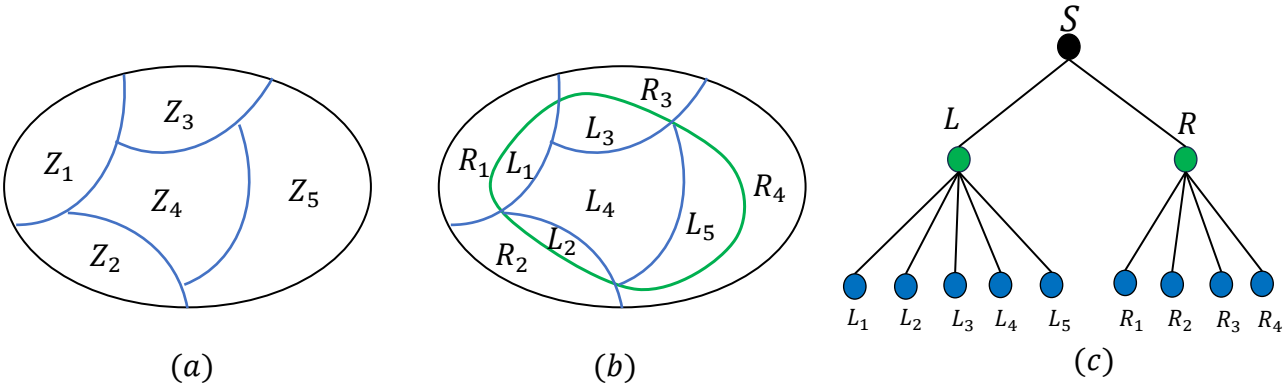


Figure 1: (a) Illustration of the first partitioning step, where a cluster S is divided into smaller clusters $Z_1, \ldots, Z_z$. (b) Illustration of the second partitioning step, where the cluster S is first split into two disjoint clusters L and R (shown in green), and then each of L and R is further subdivided according to the Z-partition into sub-clusters $L_1, \ldots, L_\ell$ and $R_1, \ldots, R_r$ (shown in blue). (c) The corresponding hierarchical-tree view, where S has two children L and R, each of which splits into its second-level sub-clusters.

As mentioned above, the first partition into $Z_1, \ldots, Z_z$ ignores the boundary edges of the cluster and focuses only on the induced subgraph $G[S]$. Afterwards, the second partition[5] into $L \cup R$, takes into account the boundary edges of S (*i.e.*, edges of $E_G(S, V \setminus S)$), as well as the inter-cluster edges from the first partition.

4.1 Partition of $G[S]$

We first partition the cluster S into clusters $Z_1, \ldots, Z_z$ such that edges between the clusters Z_i are "well-connected" in G, in the sense that the following theorem make precise. Furthermore, the clusters Z_i are small (that is, each Z_i satisfies $|Z_i| \leq \frac{2}{3}|S|$). Formally,

Theorem 4.1 (Improved version of [RST14, Theorem 3.1])**.** *There is an algorithm* merge-phase-1 *that partitions the induced subgraph $G[S]$ of a cluster S into disjoint connected components $Z_1, \ldots, Z_z$ such that:*

1. $Z_1 \cup \cdots \cup Z_z = S$,
2. $\forall i \in \{1, \ldots, z\} : |Z_i| \leq \frac{2}{3}|S|$, *and*
3. *with high probability, the set of split vertices corresponding to the inter-cluster edges*

$$X_F = \left\{ x_{(u,v)} \mid u \in Z_i, v \in Z_j, (u,v) \in E, i \neq j \right\}$$

α-expands in $G[S]'$ for $\alpha = \Omega\left(\frac{1}{\log n}\right)$ (recall the definition of the subdivision graph in Definition 3.1).

The algorithm runs in time $\tilde{O}(m)$ where m denotes the number of edges of $G[S]$.

[5] Note that the partition into $L \cup R$ corresponds to the first layer of the hierarchical tree (see Figure 1).

Note that Theorem 4.1 has $\alpha = \Omega\left(\frac{1}{\log n}\right)$, instead of $\alpha = \Omega\left(\frac{1}{\log^2 n}\right)$ in [RST14, Theorem 3.1], which leads (as shown below) to the improvement by a factor of $\log n$ in the quality of the tree cut-sparsifier. The partitioning algorithm in Theorem 4.1 is independent of the edges outside $G[S]$. Therefore, we can restrict our attention to the induced subgraph $G[S]$: in the rest of this section, $G = (V, E)$ refers to $G[S]$, and n, m denote the number of vertices and edges, respectively, in $G[S]$. Following the notation of [RST14], we say that a subset $F \subseteq E$ of edges induces a *balanced clustering* in G, if deleting F from G leaves every connected component with at most $\frac{2}{3}|V|$ vertices.

To establish Theorem 4.1, we first show the following lemma, which improves and unifies Lemmas 3.1 and 3.2 in [RST14].

Lemma 4.2 (Improved version of [RST14, Lemmas 3.1 and 3.2])**.** *Given a subset $F \subseteq E$ that induces a balanced clustering, we can find in time $\tilde{O}(m)$, either*

1. *a smaller edge set $\tilde{F} \subseteq E$ that induces a balanced clustering, with $|\tilde{F}| \leq |F| \cdot \left(1 - \frac{c}{\log n}\right)$ for some constant $c > 0$,* ***or***
2. *an edge set $\tilde{F} = \tilde{A} \cup \tilde{C}$, such that $\tilde{F}$ induces a balanced clustering, and with high probability $X_{\tilde{A}}$ $\Omega(1/\log n)$-expands in G'. Additionally there exists a flow in G' from nodes in $X_{\tilde{C}}$ to $X_{\tilde{A}}$ that can be routed with congestion $O(1)$ and:*
   - *every node $x_c \in X_{\tilde{C}}$ sends 1 unit of flow;*
   - *every node $x_a \in X_{\tilde{A}}$ receives at most 1 unit of flow.*

We begin by showing how Lemma 4.2 implies Theorem 4.1.

Proof of Theorem 4.1. We begin with $F = E$ (which induces a balanced clustering) and repeatedly invoke Lemma 4.2 on F until it terminates with Case (2). Each time the lemma terminates with Case (1), $|F|$ shrinks by a factor of at least $1 - \frac{c}{\log n}$, and still induces a balanced clustering. Consequently, after at most $O(\log^2 n)$ iterations, the procedure must terminate with Case (2).

Once Lemma 4.2 terminates with Case (2), $X_{\tilde{A}}$ $\Omega(1/\log n)$-expands in G', and we can route in G' with congestion $O(1)$ a flow such that each node in $X_{\tilde{C}}$ sends 1 unit of flow and each node in $X_{\tilde{A}}$ receives at most 1 unit of flow. By Corollary 4.4 below, this implies that edges in $\tilde{F} = \tilde{A} \cup \tilde{C}$ satisfy that $X_{\tilde{F}}$ $\Omega(1/\log n)$-expands in G'. Additionally, $\tilde{F}$ induces a balanced clustering whose connected components will be $\{Z_1, \ldots, Z_z\}$.[6] This proves Theorem 4.1. □

Next, we prove that $X_{\tilde{F}} = X_{\tilde{A}} \cup X_{\tilde{C}}$ $\Omega(1/\log n)$-expands in G'. Intuitively (see the discussion proceeding Definition 3.5), we wish to show that G' respects the all-to-all multi-commodity flow problem between the nodes of $X_{\tilde{F}}$ (see Definition 2.2). Using the fact that $X_{\tilde{A}}$ expands in G' (thus G' respects the all-to-all flow problem on $X_{\tilde{A}}$) and that there is a low congestion flow f from $X_{\tilde{C}}$ to $X_{\tilde{A}}$ (in particular G' respects the flow problem that moves mass from $X_{\tilde{C}}$ to $X_{\tilde{A}}$ according to f), we prove that G' respects the flow problem (*i.e.*, demand matrix) corresponding to the following transition of mass:

1. First, "move" the demands in $X_{\tilde{C}}$ to $X_{\tilde{A}}$ according to f.
2. Then, spread the demands in $X_{\tilde{A}}$ uniformly (using the fact that $X_{\tilde{A}}$ expands in G').

[6] After computing $\tilde{F}$ and the resulting partition $\{Z_1, \ldots, Z_z\}$, we re-define F to be the new set of inter-cluster edges $F := \{(u, v) \mid u \in Z_i, v \in Z_j, (u, v) \in E, i \neq j\}$. This set satisfies $F \subseteq \tilde{F}$ and thus $\Omega(1/\log n)$-expands in $G[S]'$ (see Fact 3.8).

3. Finally, "move" part of the (uniform) demands in $X_{\tilde{A}}$ back to $X_{\tilde{C}}$ according f (in reverse).

We show a slightly more general result:

Lemma 4.3. *Let G be a graph, and let $A, C \subseteq V$. Assume that A ϕ-expands in G, and additionally that there exists a flow routable with congestion c, such that each vertex $v \in C$ sends 1 unit of flow, and each vertex $u \in A$ receives at most a units. Then $A \cup C$ $\frac{\phi}{2\cdot(a+1+c\phi)}$-expands in G.*

Proof. Consider the all-to-all flow problem (according to Definition 2.2) $Q_{A\cup C}$ on $A \cup C$ in G. According to Remark 3.6, we should show that G respects $Q_{A\cup C}$. Let $P_1 := P_{A\cup C}$ be the corresponding valid demand state (see Remark 3.15). Note that for each $u \in A \cup C$, $\|P_1(u)\|_1 = 1 - \frac{1}{|A\cup C|} \leq 1$. Now, we use the given flow from C to A, denoted Q, to transport the demand state vectors of the vertices in $C \setminus A$ to vertices in A. We scale the flow paths of Q (*i.e.*, the rows of its demand matrix) such that each $u \in C \setminus A$ sends $1 - \frac{1}{|A\cup C|}$ units and the rest send 0. Let Q' be the scaled flow, and consider the updated demand state $P_2 := P_1^{\uparrow Q'}$ (recall Definition 3.18). Similar to Remark 3.20, P_2 is now supported only on A. Additionally, the load on each $u \in A$ is at most $a + 1$ units. Because Q can be routed with congestion c and Q' is a scaled down variant of Q, we get that in particular (see Fact 2.7) G $\frac{1}{c}$-respects Q', so by Fact 3.19, for any cut $(B, W = V \setminus B)$ we have $\mathrm{dem}_{(P_1-P_2)}(B, W) \leq c \cdot \mathrm{cap}_G(B, W)$.

Next, since A ϕ-expands in G, Remark 3.6 implies that G ϕ-respects the all-to-all flow on A, denoted Q_A. Scaling this flow problem by at most $a + 1$, we can get a flow problem which routes $\|P_2(u)\|_1$ units out of each $u \in A$, and spreads them uniformly on A. This scaled problem is $\frac{\phi}{a+1}$-respected by G. Updating P_2 using this scaled flow gives P_3, which by Remark 3.20, has the same demand vectors $P_3(u)$ for all $u \in A$. Because P_1 is a valid demand state, so are P_2 and P_3 (see Fact 3.19). This means that $P_3 \equiv 0$.

Similar to before, we get that for any cut (B, W) we have $\mathrm{dem}_{(P_2-P_3)}(B, W) \leq \frac{a+1}{\phi} \cdot \mathrm{cap}_G(B, W)$. Therefore, using Fact 3.17, for any cut (B, W), we have

$$\begin{aligned}\mathrm{dem}_{Q_{A\cup C}}(B, W) = \mathrm{dem}_{P_1}(B, W) &\leq \mathrm{dem}_{(P_1-P_2)}(B, W) + \mathrm{dem}_{(P_2-P_3)}(B, W) \\ &\leq \left(c + \frac{a+1}{\phi}\right) \cdot \mathrm{cap}_G(B, W) = \frac{a+1+c\phi}{\phi} \cdot \mathrm{cap}_G(B, W),\end{aligned}$$

which gives the result using Remark 3.6.

□

As a direct corollary of Lemmas 4.2 and 4.3, we get:

Corollary 4.4. *The set $X_{\tilde{F}}$ $\Omega(1/\log n)$-expands in G'.*

To prove Lemma 4.2 we need the following claim. It is a slightly more general rephrasing of Claim 1 of [RST14]. We provide its proof for completeness.

Claim 4.5 ([RST14, Claim 1])**.** *Let $F \subseteq E$ be a subset that induces a balanced clustering. Let $A \subseteq F$ be arbitrary. Let $C \subseteq E$ be a set of edges that separates A and $F \setminus A$. Then, either $A \cup C$ or $(F \setminus A) \cup C$ induces a balanced clustering.*

Proof. Consider the graph $G \setminus C$ where we remove the edges in C. Denote by B the set of vertices that can reach any edge of A in this graph, and by B' the set of vertices that can reach any edge of $F \setminus A$ in this graph. Because C separates A from $F \setminus A$, $B \cap B' = \emptyset$. Assume w.l.o.g that $|B| \leq |B'|$. In particular, $|B| \leq \frac{1}{2}|V|$. Note that each connected component of the graph $G \setminus ((F \setminus A) \cup C)$, is either a subset of a connected component of $G \setminus F$, or it is a subset of B. In particular all connected components have size at most $\frac{2}{3}|V|$. □

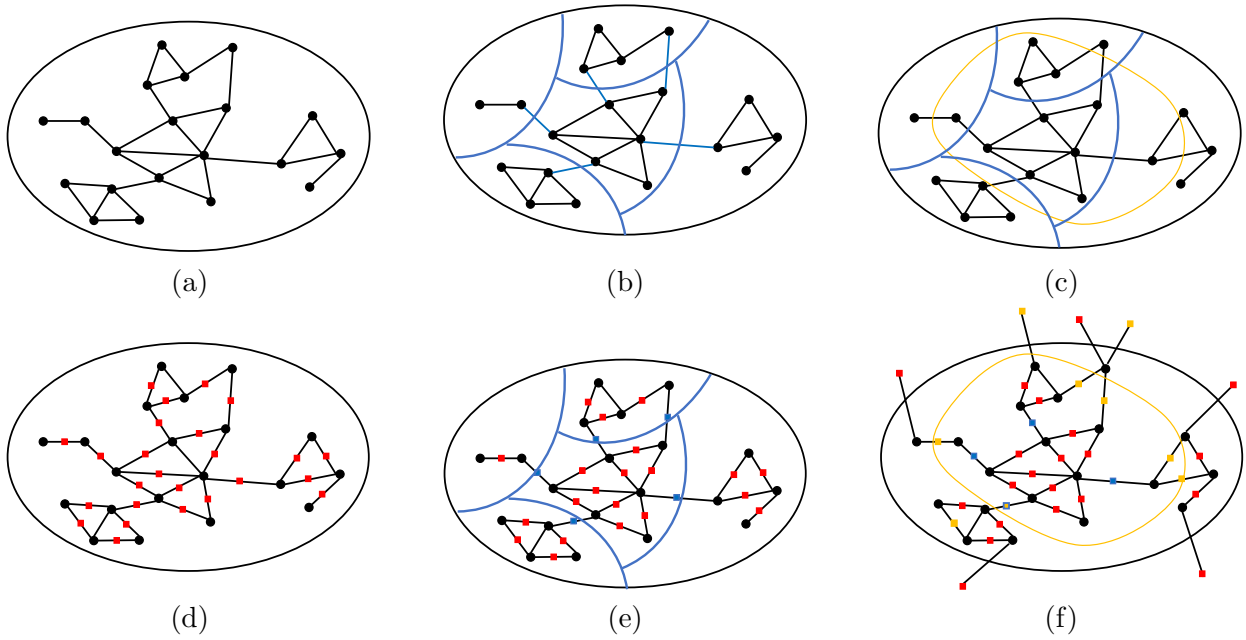


Figure 2: Partitioning of cluster S. Figures (a)-(c) represent G, figures (d)-(f) represent the split graph. Black ovals depict vertices and squares depict split nodes. Figures (b) and (e) correspond to the first partition. Blue edges (and blue squares) depict inter-cluster edges of the first partition (that is, F and X_F, respectively). Figure (f) depicts the second partition (which is computed in $G'[S']$). The core L is enveloped in gold. Split nodes in X_Y are filled with gold. Split nodes in $X_{F\cap Y}$ are colored in blue and gold.

Proof of Lemma 4.2. Given the subset $F \subseteq E$ that induces a balanced clustering, we use Corollary 3.11 on G' with $\phi := \Theta\left(\frac{1}{\log n}\right)$ and $\mu := \mu_{X_F}$. It terminates with one of three cases:

1. With high probability, $\Phi^{X_F}(G') = \Omega(\frac{1}{\log n})$. In this case X_F $\Omega(\frac{1}{\log n})$-expands in G', and we have Case (2) of the lemma (with $\tilde{A} = F, \tilde{C} = \emptyset$).

2. We find a cut $(A', \bar{A}')$ in G' of μ-expansion $\Phi^{X_F}_{G'}(A', \bar{A}') = O(\phi \log n)$, and $|X_F \cap A'|, |X_F \setminus A'|$ are both $\Omega(\frac{|F|}{\log n})$. We choose the hidden constant in the definition of $\phi = \Theta\left(\frac{1}{\log n}\right)$ such that

$$\Phi^{X_F}_{G'}(A', \bar{A}') \leq \frac{1}{2}\ , \tag{1}$$

and both

$$|X_F \cap A'|, |X_F \setminus A'| \geq 2c_1 \cdot \frac{|F|}{\log n}\ , \tag{2}$$

for some $c_1 > 0$. Let $C' \subseteq E'$ be the set of edges that cross the cut $(A', \bar{A}')$ in G'. Let $C \subseteq E$ be the corresponding set in G: $C := \{e \in E \mid \exists c \in C' : x_e \in c\}$. We have $|C| \leq |C'|$. Inequality (1) is equivalent to $|C'| \leq \frac{1}{2}\min(|X_F \cap A'|, |X_F \setminus A'|)$. Let $A \subseteq E$ be the set of edges corresponding to A': $A := \{e \in E \mid x_e \in A'\}$. We have $|F \cap A| = |X_F \cap A'|$ and

$|F \setminus A| = |X_F \setminus A'|$. Therefore,

$$|C| \leq \frac{1}{2}\min(|F \cap A|, |F \setminus A|) \ . \tag{3}$$

Since the set of edges C' separated A' from $\bar{A}'$ in G', the edges in C separate the edges in A from the edges in $E \setminus A$ in the graph G. Claim 4.5 gives that either $(F \cap A) \cup C$ or $(F \setminus A) \cup C$ induces a balanced clustering. W.l.o.g it is $(F \cap A) \cup C$. We have

$$\begin{aligned}|(F \cap A) \cup C| &\leq |F| - |F \setminus A| + |C| \leq |F| - |F \setminus A| + \frac{1}{2}\min(|F \cap A|, |F \setminus A|) \\ &\leq |F| - \frac{1}{2}|F \setminus A| \leq |F| \cdot \left(1 - \frac{c_1}{\log n}\right)\end{aligned}$$

where the second inequality follows from (3), and the last one follows from (2). So we get Case (1) of the lemma, for $\tilde{F} = (F \cap A) \cup C$.

3. We find a cut $(A', \bar{A}')$ in G' with $0 < |X_F \setminus A'| \leq \frac{|F|}{\log n}$, $\Phi^{X_F}_{G'}(A', \bar{A}') = O(\phi \log n)$, and with high probability, A' $\Omega(\phi)$-expands with respect to X_F in G. As $\phi = \Theta\left(\frac{1}{\log n}\right)$, denote by $c_2 > 0$ the constant such that $\Phi^{X_F}_{G'}(A', \bar{A}') \leq c_2$. Like in the previous case, denote by $C' \subseteq E'$ the set of edges that cross the cut $(A', \bar{A}')$ in G'. Let $C \subseteq E$ be the corresponding set in G: $C := \{e \in E \mid \exists c \in C' : x_e \in c\}$. We have $|C| \leq |C'|$. Because the cut is sparse with respect to μ_{X_F}, $|C'| \leq c_2 \cdot |X_F \setminus A'|$.

   From remark 3.10, we get that there exists a flow in G', routable with congestion $O(1)$, such that each split-node in X_C sends 1 unit of flow and each node $v \in A'$ receives at most $O(\phi \cdot \mu(v))$ units. Recall that $\phi = \Theta\left(\frac{1}{\log n}\right)$, so for large enough n, we may assume that each node receives at most $\mu(v)$ units of flow. In particular, only split nodes $v \in X_F$ can receive flow, and each receives at most one unit.

   As in the previous case, let $A := \{e \in E \mid x_e \in A'\}$. We have $|F \cap A| = |X_F \cap A'|$ and $|F \setminus A| = |X_F \setminus A'|$. Denote $\tilde{A} = F \cap A$, $\tilde{R} = F \setminus A$. Then, $\tilde{A} \cup \tilde{R} = F$ induces a balanced clustering.

   As in the previous case, observe that the set $C \subseteq E$ separates $\tilde{A}$ from $\tilde{R}$. By Claim 4.5, either $\tilde{A} \cup C$ or $\tilde{R} \cup C$ induces a balanced clustering.

   (a) Assume $\tilde{R} \cup C$ induces a balanced clustering. In this case, we will set $\tilde{F} := \tilde{R} \cup C$, and claim that we get Case (1) of the lemma. Indeed, we have $|\tilde{R} \cup C| \leq |\tilde{R}| + |C| = |F \setminus A| + |C| \leq (c_2 + 1) \cdot |X_F \setminus A'| \leq (c_2 + 1) \cdot \frac{|F|}{\log n} < |F| \cdot (1 - \frac{1}{\log n})$ for large enough n.

   (b) Assume $\tilde{A} \cup C$ induces a balanced clustering. We set $\tilde{C} := C$, $\tilde{F} = \tilde{A} \cup \tilde{C}$, and claim that we have Case (2) of the lemma. As mentioned above, the existence of the flow is guaranteed by Remark 3.10. Each node $x_a \in X_{\tilde{A}} = X_F \cap A'$ receives at most one unit of flow, and the rest receive zero.

   The fact that A' $\Omega(\phi)$-expands with respect to μ_{X_F} in G' means that G' is an $\Omega(\phi)$ expander with respect to $A' \cap X_F = X_{\tilde{A}}$, which by definition means that $X_{\tilde{A}}$ $\Omega(\phi)$-expands in G'. As $\phi = \Theta(\frac{1}{\log n})$, this means that $X_{\tilde{A}}$ $\Omega(\frac{1}{\log n})$-expands in G'. This shows we have Case (2) of the Lemma.

□

4.2 Finding a Bottleneck Cut

We denote by $Z_1, \ldots, Z_z$ the partition of S that is returned by Theorem 4.1. Recall from that theorem that F denotes the set of inter-cluster edges (*i.e.*, $F = \{(u,v) \in E \mid u \in Z_i, v \in Z_j, i \neq j\}$). The partition of S into $L \cup R$ is computed using both this initial clustering and the set of boundary edges $B = E(S, V \setminus S)$ of S.

As mentioned in Section 2, we want the set $F \cup B$ of inter-cluster edges and boundary edges to expand in $G[S]$. However, the existence of bottleneck-cuts may make such a partition impossible (in the cut-sparsifier case, a bottleneck cut is a cut that certifies that the boundary edges are not expanding within the cluster). Following [RST14], the solution that we use in this case, is the following: As Claim 5.2 shows, in order to certify that a tree is a cut-sparsifier, we should check that all flow problems which can be routed in the tree are respected by G. We therefore find a "problematic" cut $(L, R = S \setminus L)$ and add L, R to the tree. This forces every flow problem which is respected by T to respect the cut $(L, V \setminus L)$. Specifically, we will add a cut which "blocks" the sending of flow from X_F to X_B. As shown in Section 5, this will be enough.

Formally, the second partition is parameterized by $0 < \tau \leq 1$. We shift our focus from the graph $G[S]'$ to the graph $G'[S']$, where $S' := S \cup \{x_e \mid e \cap S \geq 1\}$. That is, in addition to the vertices of $G[S]'$, we also include the nodes x_e for every boundary edge $e \in B$ and connect x_e to v where $v \in e \cap S$.

Next, we define the weighting μ_τ on the vertices of $G'[S']$: the boundary split nodes are assigned a weight of τ, and the rest of the vertices nodes have weight 1. [7] As explained above, we are looking for a cut which blocks the sending of flow from X_F to X_B. We will therefore apply the following lemma on the sets X_F and X_B. The set of edges Y will be the cut edges that "block" the flow.

Lemma 4.6 (Improved version of [RST14, Lemma 3.9])**.** *Let $G = (V, E)$ be a graph, and let μ be a measure on V'. Consider two sets of split vertices X_A and X_B in G'. We can find a set X_Y of split vertices such that the following holds:*

- *The set X_Y separates X_A from X_B. This means that every path between nodes $x_a \in X_A$ and $x_b \in X_B$ in G' must contain a vertex $x_y \in X_Y$.*
- *There exists a multi-commodity flow in G' from nodes X_Y to nodes in X_A that can be routed with congestion 2, such that every node $x_y \in X_Y$ sends out $\mu(x_y)$ units of flow, and every node $x_a \in X_A$ receives at most $2\mu(x_a)$ units of flow.*
- *There exists a multi-commodity flow in G' from nodes X_Y to nodes in X_B that can be routed with congestion 2, such that every node $x_y \in X_Y$ sends out $\mu(x_y)$ units of flow, and every node $x_b \in X_B$ receives at most $2\mu(x_b)$ units of flow.*

The set X_Y can be computed in time $\tilde{O}(m)$.

Remark 4.7. *In the lemma above, the separator Y must satisfy $\mu(X_Y) \leq 2\min(\mu(X_A), \mu(X_B))$ for such flows to exist, as the total source mass is $\mu(X_Y)$ and the total sink capacities are $2\mu(X_A), 2\mu(X_B)$.*

Consider the flow network where we add to G' a source s connected to X_A (with edges (s, x_a) of capacity $\mu(x_a)$) and a sink t connected to X_B (with edges (x_b, t) of capacity $\mu(x_b)$). It can be seen that an exact min–cut Y that separates s from t in this network would satisfy the requirements of Lemma 4.6 with the constant 1 instead of 2. However, computing a min–cut is too expensive for our purposes. Instead, we use fair–cuts [LNPS23], which provide the same guarantees up to constant factors and can be computed in near-linear time.

[7] For non split nodes, $v \in S$, $\mu_\tau(v)$ does not play a role in the following discussion.

Definition 4.8 (Fair Cut, [LNPS23])**.** *Let $G = (V, E)$ be an undirected graph with edge capacities $c \in \mathbb{R}^E_{>0}$. Let s, t be two vertices in V. For any parameter $\alpha \geq 1$, we say that a cut (S, T) is a* α-fair (s,t)-cut *if there exists a feasible (s,t)-flow f such that $f(u,v) \geq \frac{1}{\alpha} \cdot c(u,v)$ for every $(u,v) \in E(S,T)$, where $u \in S$ and $v \in T$. In particular, f is an α-approximate max flow.*

Standard approximate max-flow algorithms have the unpleasant property that the flow provided can go backwards from T to S across the approximate min-cut (S, T). Fair-cuts give a much stronger guarantee: a Fair-cut relaxes the notion of a minimum cut by requiring that *every* edge that crosses the cut is roughly saturated. Thus, the flow can only cross the cut from S to T and the capacity of every cut edge is almost fully utilized. For all of our use cases, fair-cuts give the same guarantees as min-cut, up to a constant factor.

The following theorem, due to [LL25], is the state of the art result for the computation of fair-cuts.

Theorem 4.9 (Fair Cut, [LL25, Theorem 1])**.** *Given a graph $G = (V, E)$, two vertices $s, t \in V$ and $\epsilon \in (0, 1]$, we can compute with high probability a $(1+\epsilon)$-fair (s,t)-cut in $\tilde{O}(\frac{m}{\epsilon})$ time.*

Proof of Lemma 4.6. Consider the following flow problem: connect a source s to all split nodes in $x_a \in X_A$ with capacity $\mu(x_a)$ and connect all split nodes $x_b \in X_B$ to a target t with capacity $\mu(x_b)$. Using Theorem 4.9, we get a 2-fair (s,t)-cut $(S, \bar{S})$ in G' corresponding to a 2-approximate max flow f. We define X_Y as follows: for every cut edge $(v, x_e) \in E'(S, \bar{S})$ we add x_e to X_Y. This X_Y separates X_A from X_B. From the definition of fair-cuts and the fact that $\deg_{G'}(x_e) = 2$, for every $x_e \in X_Y$, it follows that each $x_e \in X_Y$ receives (and also sends) at least $\text{cap}(e)/2$ flow. Thus, by scaling f by a factor of 2 we get the desired outcome.

To see why X_Y satisfies the constraints, decompose f into paths. Since $(S, \bar{S})$ is a fair cut, each path crosses the cut exactly once (no flow can cross the cut backwards) and therefore can be assigned to a split vertex in X_Y. Thus, we can view f as two routings: one from s (i.e. X_A) to X_Y, and one from X_Y to t (i.e. X_B). Reversing the former flow gives the result. □

We define the second partition $(L,\ R)$ by a computing the subset X_Y of split nodes in $G'[S']$, as obtained from applying Lemma 4.6 on X_F and X_B (see Figure 2). The sets L and R are then defined as follows: R consists of all vertices in S that can reach a boundary node $x_b \in X_B \setminus X_Y$ in the graph $G'[S'] \setminus X_Y$, while L contains the remaining vertices. We adopt the notation of [RST14] and call L the *core* of S. Note that $X_F \setminus X_Y \subseteq L'$, where $L' := L \cup \{x_e \mid e = (u,v), u, v \in L\}$.

By applying Lemma 4.6 with μ_τ, we get the following theorem:

Theorem 4.10. *Let $0 < \tau \leq 1$. Define the weighting μ_τ on the split vertices of $G'[S']$, such that boundary split nodes have weight τ while the rest of the split nodes have weight 1. There exists an algorithm* merge-phase-2 *which finds a partition $L \cup R = S$, induced by a set of edges $Y \subseteq E$ as described above, such that,*

1. *There is a flow in $G'[S']$ from nodes in X_Y to nodes in X_B, routable with congestion 2, such that* every *node $x_y \in X_Y$ sends out $\mu_\tau(x_y)$ units of flow, and each node $x_b \in X_B$ gets at most 2τ units of flow.*

2. *There is a flow in $G'[S']$ from nodes in X_Y to nodes in X_F, routable with congestion 2, such that* every *node $x_y \in X_Y$ sends out $\mu_\tau(x_y)$ units of flow, and each node $x_f \in X_F$ gets at most 2 units of flow.*

3. *The nodes in X_F are separated from X_B in $G'[S'] \setminus X_Y$. This means that every path in $G'[S']$ that starts at $x_b \in X_B$ and ends at $x_f \in X_F$ contains a node from X_Y (which may be x_b or x_f).*

4. *The nodes in L are separated from X_B in $G'[S'] \setminus X_Y$.*

The algorithm runs in time $\tilde{O}(m)$ where m denotes the number of edges of $G'[S']$.

5 The Charging Scheme

Let T be the tree corresponding to the hierarchical decomposition from Section 4. In this section we show that T is indeed a tree cut-sparsifier with quality $O(\log^3 n)$, proving Theorem 2.9. We define the capacity in T of an edge connecting a cluster S to its parent cluster $P(S)$, as $\text{cap}_G(S, V \setminus S)$. In section 2 we mentioned that the method used to show that a tree is a cut-sparsifier is to prove that every multi-commodity flow problem that can be routed in the tree is respected by the graph. We begin by explaining this formally.

Consider the following claim for tree flow-sparsifiers.

Claim 5.1. *If every multi-commodity flow problem (recall Definition 2.1) R on V, that can be routed in T with congestion 1, can also be routed in G with congestion α, then T is a tree flow-sparsifier with quality α.*

Proof. Let Q be a multi-commodity flow problem on V. We have to show that $\text{cong}_T(Q) \leq \text{cong}_G(Q) \leq \alpha \cdot \text{cong}_T(Q)$. The first inequality is clear: We route the flow in T naturally (a unit from u to v moves along the unique path connecting them). Therefore, there exists an edge $(S, P(S))$ of T (of capacity $\text{cap}_G(S, V \setminus S)$) such that $\text{dem}_Q(S, V \setminus S) = \text{cong}_T(Q) \cdot \text{cap}_G(S, V \setminus S)$. Moreover, since Q can be routed in G with congestion $\text{cong}_G(Q)$, it follows from Fact 2.7 that $\text{dem}_Q(S, V \setminus S) \leq \text{cong}_G(Q) \cdot \text{cap}_G(S, V \setminus S)$. Therefore, we conclude that $\text{cong}_T(Q) \leq \text{cong}_G(Q)$.

For the other inequality, let R denote the multi-commodity flow problem corresponding to $\frac{1}{\text{cong}_T(Q)} \cdot Q$. This flow can be routed in T with congestion 1. Hence,

$$\alpha \geq \text{cong}_G\left(\frac{1}{\text{cong}_T(Q)} \cdot Q\right) = \frac{\text{cong}_G(Q)}{\text{cong}_T(Q)},$$

as required. □

The following is the corresponding claim for tree cut-sparsifiers. This will be our main tool to show that T is a tree cut-sparsifier.

Claim 5.2. *If for every multi-commodity flow problem R on V, that is 1-respected by T, we have that G $\frac{1}{\alpha}$-respects R (recall Definition 2.6) then T is a tree cut-sparsifier with quality α.*

Proof. Fix a cut (B, W). We have to show that $\text{cap}_G(B, W) \leq \text{mincut}_T(B, W) \leq \alpha \cdot \text{cap}_G(B, W)$. The first inequality is clear given the definition of the capacities in T. Indeed, consider any cut (B', W') in T, where $B \subseteq B', W \subseteq W'$. For each $b \in B, w \in W$, the path from the leaf $\{b\}$ to $\{w\}$ in T must contain an edge $(S, P(S))$ that crosses the cut (B', W') of T. Assume without loss of generality that $S \in B'$ and $P(S) \in W'$. Then, $\text{cap}_G(b, w)$ will be counted in the capacity $\text{cap}_T(S, P(S)) = \text{cap}_G(S, V \setminus S)$.

Next, we will show the other inequality. Consider the flow network $T' = T \cup \{s, t\}$, where s is connected to all nodes in W and t is connected to all nodes in B with edges of infinite capacity. The maximum (s, t)-flow f in this network has value $\text{mincut}_T(B, W)$. Separate this flow into paths, and denote by R the flow problem corresponding to the resulting flow demands (each demand has a distinct commodity, whose source is a node $b \in B$ and whose target is a node $w \in W$). Clearly, the congestion of routing R in T is 1, which means (see Fact 2.7) that R is 1-respected by T. Therefore,

we have $\frac{\text{cap}_G(B,W)}{\text{dem}_R(B,W)} \geq \frac{1}{\alpha}$. Note that $\text{dem}_R(B,W)$, the total demand that crosses the cut, is exactly the value of the flow $|f| = \text{mincut}_T(B,W)$. Therefore, we have $\text{mincut}_T(B,W) \leq \alpha \cdot \text{cap}_G(B,W)$, as we wanted. □

We will prefer to work with demand states instead of multi-commodity flow problems. To this end, we'll use the following Corollary:

Corollary 5.3. *Assume every valid demand state P (recall Definition 3.14) on V, that is 1-respected by T, is $\frac{1}{\alpha}$-respected by G. In other words, we have for any cut (B,W),*

$$\frac{\text{cap}_G(B,W)}{\text{dem}_P(B,W)} \geq \frac{1}{\alpha} .$$

Then T is a tree cut-sparsifier with quality α.

Proof. Follows from Claim 5.2 and Remark 3.15. □

Therefore, we have to show that every valid demand state that is respected by T is $O(\log^3 n)$-respected by G. To this end, we use T as a *blueprint* that guides how demands are moved (in G) and charged to the capacity of cuts. Specifically, we update the demand state on G iteratively as we go up along the tree structure, such that after processing a sub-cluster S, the demand state is no longer supported on any vertex of S, apart from it's boundary split nodes.[8] As mentioned in Section 2, we will "pay" not for the congestion of routing the demands, but rather according to how much the graph respects the corresponding demand transitions.

The construction below follows [RST14], with the required modifications for the case of cuts instead of flows. In contrast to [RST14], our presentation makes use of demand states, which provides a convenient and precise framework for describing the dynamic movements of the demands as we go up along the decomposition tree and for analyzing how the graph respects them.

Fix a cut $(B \subseteq V, W = V \setminus B)$ in G, and fix a demand state P on V, which is 1-respected by T. Our goal is to show that $\text{dem}_P(B,W) \leq O(\log^3 n) \cdot \text{cap}_G(B,W)$. We do this by charging the demand crossing the cut to the cut edges, such that the total charges dominate the demand, and each edge is charged $O(\log^3 n)$.

We work in the subdivision graph $G' = (V' = V \cup X_E, E')$. Recall that for a cluster $S \subseteq V$, we define the corresponding cluster in the subdivision graph as $S' := S \cup \{x_e \mid e \cap S \geq 1\}$, which also includes the split nodes of the boundary edges of S. Recall that the graph $G'[S']$ is the subdivision graph restricted to the set S'.

Let $(B' = B \cup \{x_e \mid e \cap B \geq 1\}, W' = V' \setminus B')$ be the corresponding cut in G' (see Figure 3).[9] Intuitively, we want to convert the demand state P into a demand state P' on G', and then charge the edges crossing (B', W') to cover the demand $\text{dem}_{P'}(B', W')$, which would be bounded from below by $\text{dem}_P(B,W)$.

[RST14] essentially used Claim 5.1 to prove that T is a tree flow-sparsifier, and to this end they showed how to route the corresponding valid demand state in G with low congestion. Though they did not explicitly use the notation of demand states, their key method (rephrased in this notation) was akin to maintaining a collection of demand states, each supported on some sub-cluster S' of the subdivision graph, corresponding to a vertex of T. The original state P was converted to a demand state $P_{\{v\}'}$ supported on $\{v\}'$[10], for each vertex $v \in V$ (these are the leaves in the decomposition

[8] As will be explained soon, we work in the subdivision graph.

[9] Note that we arbitrarily chose to include the split vertices of the edges crossing (B,W) in B'.

[10] The notation $\{v\}'$ refers to the cluster S' induced by the single vertex set $S = \{v\}$. Precisely, this includes the vertex v as well as it's adjacent split nodes.

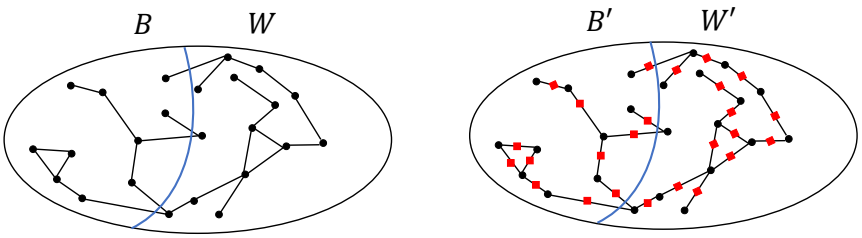


Figure 3: Figure (a) - a cut (B, W) in G. Figure (b) - the corresponding cut (B', W') in G'.

tree T). They then showed how to merge the demand states according to the structure of the decomposition tree.

We proceed similarly. We combine the demand states when we go up the tree T, by constructing a series of flow problems (*i.e.*, demand matrices) to update one demand state to the next (recall Definition 3.18). Instead of showing that these matrices can be routed with low congestion, we charge the edges of the cut (B', W') to cover the demands that these demand matrices induce across the cut (so that the charge incurred at each step is at least the demand of the demand matrix across the cut). This is made explicit below.

Initially, each edge of (B', W') is charged 0. We convert the given valid demand state P to multiple demand states $P_{\{v\}'}$, one for each vertex $v \in V$. For each $v \in V$, its demand vector $P(v)$ is evenly distributed among the split nodes x_e, incident to v. Formally, each split node x_e receives the demand vector $P_{\{v\}'}(x_e) = \frac{1}{\deg(v)} P(v)$. Observe that $\|P(v)\|_1 = \mathrm{dem}_P(\{v\}, V \setminus \{v\}) \leq \mathrm{cap}_G(\{v\}, V \setminus \{v\}) = \deg_G(v)$, where the inequality is due to the fact that $\{v\}$ is a cluster in T and that P is 1-respected by T. In particular, each split node x_e has a total load of at most 1: $\left\|P_{\{v\}'}(x_e)\right\|_1 = \frac{1}{\deg(v)} \left\|P_{\{v\}'}\right\|_1 \leq 1$. Each non-split vertex of V a load of 0.

We combine demand states as we go up the decomposition tree T. We consider each cluster in T, and replace the demand states $P_{S'_1}, \ldots, P_{S'_r}$ that are supported on its sub-clusters $S'_1, \ldots, S'_r$ with a new demand state that is supported on S'. This is done in Theorem 5.5 below. Note that our demand states are not valid demand states. They will, however, satisfy the following invariant:

Invariant 5.4. *For a cluster S and $\alpha \geq 1$, we say that a demand state $P_{S'}$* satisfies the invariant *with load α if*

(1) It is supported on the boundary of S:

$$\forall k \in \mathcal{K}, \forall v \notin X_B : P_{S'}(v, k) = 0 \ ,$$

where B are the boundary edges of S.

(2) No demands were created or destroyed (relative to P):

$$\forall k \in \mathcal{K} : \sum_{v \in S'} P_{S'}(v, k) = \sum_{v \in S} P(v, k) \ .$$

(3) The load on each vertex is at most α:

$$\forall v \in S', \|P_{S'}(v)\|_1 \leq \alpha \ .$$

Note that $P_{S'}$ satisfying the invariant does not necessarily imply that $P_{S'}$ is a valid demand state. It can be seen that $P_{\{v\}'}$ satisfies the invariant with $\alpha = 1$ for the cluster $\{v\}$.

Recall from Section 4 that the partition of a cluster S is comprised of two levels. In the first level, S is cut into two sets L and R, and in the second level L and R are partitioned into $L_1, \ldots, L_\ell$ and $R_1, \ldots, R_r$, respectively. Further recall the notation from Section 4: The set F denotes the *inter-cluster edges* from Theorem 4.1, the set Y denotes the *cut edges* from Theorem 4.10 and the set B denotes the *boundary edges* $E(S, V \setminus S)$. These sets may intersect. In Section 5.1, we describe a procedure that is the core component of our argument, and "combines" demand states satisfying Invariant 5.4 for sub-clusters of a cluster S to a demand state that satisfies it for S.

The properties of this process are summarized in the following theorem:

Theorem 5.5. *Let S be a cluster, and let $S_1, \ldots, S_t$ be its sub-clusters (i.e., $\{S_1, \ldots S_t\} = \{L_1, \ldots, L_\ell\} \cup \{R_1, \ldots, R_r\}$), computed by Theorems 4.1 and 4.10 with parameter $0 < \tau \leq 1$. For each $1 \leq i \leq t$, assume that $P_{S_i'}$ is a demand state satisfying Invariant 5.4 for S_i with load $\alpha \geq 1$. Let $P_{S'}^{\text{before}} = \sum_{i=1}^{t} P_{S_i'}$ be their sum. Then, we can replace $P_{S'}^{\text{before}}$ with a demand state $P_{S'}^{\text{after}}$ which satisfies the following:*

1. *$P_{S'}^{\text{after}}$ satisfies Invariant 5.4 for S with load $\alpha' = \alpha \cdot (1 + c \cdot \tau)$ for some constant $c > 0$.*
2. *$G'[S']$ $\Omega\left(\frac{\tau}{\alpha \log n}\right)$-respects $P_{S'}^{\text{before}} - P_{S'}^{\text{after}}$.*

Remark 5.6. *Result (2) of Theorem 5.5 means that if we apply a charge on the edges $e \in E(B', W') \cap G'[S']$ (to cover the demand of $P_{S'}^{\text{before}} - P_{S'}^{\text{after}}$ across the cut (B', W')) as follows:*

$$\text{charge}_S := \frac{\text{dem}_{\left(P_{S'}^{\text{before}} - P_{S'}^{\text{after}}\right)}(B', W')}{\text{cap}(E(B', W') \cap G'[S'])},$$

then we have that $\text{charge}_S = O\left(\frac{\alpha \log n}{\tau}\right)$. Note that the charged edges are in the subdivision graph.

In other words, Theorem 5.5 shows that the demand states supported on the sub-clusters $S_1, \ldots, S_t$ can be merged into a single demand state on S, while only slightly increasing the load (by a factor of $1 + O(\tau)$) on the boundary edges of S. The cost of this operation is that we add $O\left(\frac{\alpha \log n}{\tau}\right)$ to the charge of each edge $e \in E(B', W') \cap G'[S']$. The proof of this theorem is given in Section 5.1.

Next, we use Theorem 5.5 to show Theorem 2.9.

Proof of Theorem 2.9. We prove that the conditions of Corollary 5.3 are satisfied. Let P, (B, W), and (B', W') be as in the discussion following Corollary 5.3. Set $\tau = \frac{1}{\log n}$. We process P in a bottom-up manner: first we convert it to a set of demand states $P_{\{v\}'}$ as shown above. Then, for each cluster in T, we apply Theorem 5.5 and Remark 5.6, accumulating the charges incurred at each application of the theorem. Let $\text{charge}_S(e) := \text{charge}_S$ denote the charge assigned to an edge $e \in E(B', W') \cap G'[S']$ when Theorem 5.5 is applied to cluster S.

After applying the theorem for the final cluster V, the resulting demand state $P_{V'}$ is actually the zero demand state on V'. This follows from Invariant 5.4(1), since the cluster V has no boundary edges. We now explain why the accumulated charges are enough to cover $\text{dem}_P(B, W)$.

Denote by $\mathcal{Q}$ the set of current "active" clusters, such that initially $\mathcal{Q}_0 = \{\{v\} \mid v \in V\}$. Then in each iteration we remove $S_1, \ldots, S_t$ from $\mathcal{Q}$ and add S. Let $P_{\mathcal{Q}}$ be the sum of all demand states $P_{S'}$, for $S \in \mathcal{Q}$.

Claim 5.7. *In each step, $P_{\mathcal{Q}}$ is a valid demand state on V'. That is, $\sum_{v\in V'} P_{\mathcal{Q}}(v,k) = 0$, for every $k \in \mathcal{K}$.*

Proof. By induction, for each $S \in \mathcal{Q}$, $P_{S'}$ satisfies Invariant 5.4. Therefore, for each $k \in \mathcal{K}$,

$$\begin{aligned}\sum_{v\in V'} P_{\mathcal{Q}}(v,k) &= \sum_{v\in V'}\sum_{S\in\mathcal{Q}} P_{S'}(v,k) = \sum_{S\in\mathcal{Q}}\sum_{v\in V'} P_{S'}(v,k)\\ &\overset{(3)}{=} \sum_{S\in\mathcal{Q}}\sum_{v\in S'} P_{S'}(v,k) \overset{(4)}{=} \sum_{S\in\mathcal{Q}}\sum_{v\in S} P(v,k) = \sum_{v\in V} P(v,k) = 0,\end{aligned}$$

where Equality (3) holds since $P_{S'}$ is supported on S' and Equality (4) uses Invariant 5.4(2). □

The following claim, whose proof is deferred to Appendix A, shows that $\mathrm{dem}_P(B,W)$ and $\mathrm{dem}_{P_{\mathcal{Q}_0}}(B',W')$ are the same up to some minor error that arises due to split nodes on the cut (B',W') of G'.

Claim 5.8. $\mathrm{dem}_P(B,W) \le \mathrm{dem}_{P_{\mathcal{Q}_0}}(B',W') + \mathrm{cap}_G(B,W)$.

Consider a replacement of $\mathcal{Q}$ by $\mathcal{Q}' = \mathcal{Q} \cup \{S\} \setminus \{S_1, S_2, \ldots, S_t\}$ when we apply Theorem 5.5. Recall that $P^{\text{before}}_{S'} = \sum_{i=1}^{t} P_{S'_i}$ is replaced by $P^{\text{after}}_{S'}$. Let $P^{\text{before}}_{\mathcal{Q}'} := P_{\mathcal{Q}}$ be the sum of active demand states before the change and let $P^{\text{after}}_{\mathcal{Q}'} := P_{\mathcal{Q}} - P^{\text{before}}_{S'} + P^{\text{after}}_{S'}$ be the sum after the change. We have $\mathrm{dem}_{\left(P^{\text{before}}_{\mathcal{Q}'} - P^{\text{after}}_{\mathcal{Q}'}\right)}(B',W') = \mathrm{dem}_{\left(P^{\text{before}}_{S'} - P^{\text{after}}_{S'}\right)}(B',W')$. This means that after accumulating all the charges in the applications of Theorem 5.5 on all clusters $S \in T$, the total charge satisfies

$$\begin{aligned}\sum_{e\in E(B',W')} \mathrm{charge}(e) &= \sum_{e\in E(B',W')}\sum_{S\in T} \mathrm{charge}_S(e) = \sum_{S\in T}\sum_{e\in E(B',W')\cap G'[S']} \mathrm{charge}_S(e)\\ &\ge \sum_{S\in T} \mathrm{dem}_{\left(P^{\text{before}}_{S'} - P^{\text{after}}_{S'}\right)}(B',W') \overset{(2)}{\ge} \mathrm{dem}_{\left(P_{\mathcal{Q}_0} - P_{V'}\right)}(B',W') = \mathrm{dem}_{P_{\mathcal{Q}_0}}(B',W')\\ &\ge \mathrm{dem}_P(B,W) - \mathrm{cap}_G(B,W)\ ,\end{aligned}$$

where Inequality (2) uses Fact 3.17. By charging an additional unit to each cut edge, we get $\mathrm{dem}_P(B,W) \le \sum_{e\in E(B',W')} \mathrm{charge}(e)$.

To conclude, we bound the total charge on each edge as follows. The height of the decomposition tree is $O(\log n)$ and $\tau = \frac{1}{\log n}$, which means that α is a constant for each cluster. Therefore, by Theorem 5.5 and Remark 5.6, each edge is charged $O(\log^2 n)$ for each cluster it is contained in, for a total charge of $O(\log^3 n)$ per edge, which gives $\mathrm{dem}_P(B,W) = O(\log^3 n)\cdot \mathrm{cap}_{G'}(B',W') = O(\log^3 n)\cdot \mathrm{cap}_G(B,W)$, as we wanted. Thus, the conditions of Corollary 5.3 are satisfied, and Theorem 2.9 follows. □

5.1 Proof of Theorem 5.5

The goal of this section is to prove Theorem 5.5. Recall the notations of this theorem: $S \in T$ is a cluster, and let $S_1, \ldots, S_t$ be its sub-clusters (*i.e.*, $\{S_1, \ldots S_t\} = \{L_1, \ldots, L_\ell\} \cup \{R_1, \ldots, R_r\}$), computed by Theorems 4.1 and 4.10 with parameter $0 < \tau \le 1$. For $i = 1 \ldots t$, $P_{S'_i}$ is a demand state satisfying Invariant 5.4 for S_i with a given load $\alpha \ge 1$.

Let $P^{\text{before}}_{S'} := \sum_{i=1}^{t} P_{S'_i}$ be the sum of the demand states on the sub-clusters. By Invariant 5.4(1), all the non-zero demands in $P^{\text{before}}_{S'}$ are residing at nodes $X_F \cup X_Y \cup X_B$ of S' (these are the boundary edges of $\{S_i\}_{i=1}^{t}$). Given these demand vectors, we wish to "solve" some of these demands by updating the demand states such that a positive demand and a negative demand of the

same commodity are moved to the same vertex, thus canceling out, and then "send" the remaining demands to X_B (the boundary of S). In other words, we replace $P^{\text{before}}_{S'}$ by a demand state $P^{\text{after}}_{S'}$, where the demands of $P^{\text{after}}_{S'}$ lie on X_B (*i.e.*, satisfying Invariant 5.4 for S), and we show that $G'[S']$ $\Omega(\frac{\tau}{\alpha \log n})$-respects $P^{\text{before}}_{S'} - P^{\text{after}}_{S'}$.

This uses a sequence of demand state updates:

1. Denote $P^{(1)}_{S'} := P^{\text{before}}_{S'}$. First, we deal with demands that originated inside L, *i.e.*, with the demand states $P^{(1)}_{L'} = \sum_i P_{L'_i}$, where $L = \bigcup_i L_i$. We move the demands of $P^{(1)}_{L'}$ that reside on $X_Y \cap L'$ to X_F, using the flow guaranteed by Property 2 of Theorem 4.10. This flow is routable in the graph and in particular, by Fact 2.7, respected by the graph.

2. Then, we use the all-to-all multi-commodity flow problem on X_F to update the demand state and spread the demands uniformly on X_F. By Theorem 4.1 we know that this problem is respected by the graph. This means that positive and negative demands of the same commodity cancel out, and we are only left with demand that has to enter or leave the cluster L. Because L is a node in the hierarchical decomposition tree T and P is respected by T, the total amount of leftover demand is at most $\text{cap}_G(L, V \setminus L)$.

3. Then, we use the reverse flow from step (1) to route the demands back to X_Y.[11]

4. Finally, we take all the demands that are currently on X_Y (either from $P_{R'_i} := P^{(1)}_{S'} - P^{(1)}_{L'}$ or the resulting updated demands), and use the flow ensured by Property 1 of Theorem 4.10 to route them to X_B. This flow is routable in the graph and, by Fact 2.7, respected by the graph.

In all of these steps, we show that $G'[S']$ respects the updates, not necessarily bounding the actual congestion that would be incurred when routing these flows. We split the above process into lemmas.

Let $P^{(1)}_{L'} := \sum_{i=1}^{\ell} P_{L'_i}$ be the sum of the demand states on the sub-clusters within the core L. The following lemma captures steps 1 and 2 in the above process.

Lemma 5.9. *We can replace $P^{(1)}_{L'}$ by a demand state $P^{(3)}_{L'}$ which satisfies the following:*

1. *$P^{(3)}_{L'}$ is supported and spread uniformly on X_F. Formally, $P^{(3)}_{L'}(x_e) = \frac{1}{|X_F|} \sum_{v \in L'} P^{(1)}_{L'}(v)$, for $x_e \in X_F$, and $P^{(3)}_{L'}(v) = 0$ for $v \notin X_F$. Note that $P^{(3)}_{L'}$ is no longer supported on L'.*

2. *$G'[S']$ $\Omega\left(\frac{\tau}{\alpha \log n}\right)$-respects $P^{(1)}_{L'} - P^{(3)}_{L'}$.*

Proof. Initially, Invariant 5.4(3) gives that every boundary split-node $x_b \in X_B$ in the graph $G'[S']$ has load at most α in $P^{(1)}_{S'}$, while vertices of $(X_F \cup X_Y) \setminus X_B$ may have load at most 2α (each of these vertices is a boundary node of two of the sub-clusters $\{S_i\}_{i=1}^t$, so two sub-clusters may have placed a load of α on it). That is, $\sum_{k \in \mathcal{K}} |P^{(1)}_{S'}(x_b, k)| \leq \alpha$ for boundary nodes $x_b \in X_B$, and 2α for nodes in $(X_F \cup X_Y) \setminus X_B$. Due to the invariant, $P^{(1)}_{L'}$ is supported only on L'.

The first step is to move the demands of $P^{(1)}_{L'}$ from $X_Y \cap L'$ to X_F. Specifically, each vertex $x_y \in X_Y \cap L'$ will send $P^{(1)}_{L'}(x_y, k) = \sum_{i=1}^{\ell} P_{L'_i}(x_y, k)$ units of commodity k to X_F, for every $k \in \mathcal{K}$. As mentioned above, this is at most 2α for each vertex.

[11] In fact, we route to only some edges of Y (see the proof for details).

We use Property 2 of Theorem 4.10 for the set Y, and "move" the demands that reside at these vertices to vertices in X_F. Property 2 says that there exists a flow which can be routed with congestion $O(1)$ in $G'[S']$ such that every node $x_y \in X_Y$ sends $\mu_\tau(x_y)$ units of mass, which is at least τ (if $y \in B$, then this is tight, otherwise $\mu_\tau(x_y) = 1 \geq \tau$), and every node in X_F receives at most 2 units of mass. Every such $x_y \in X_Y$ has at most 2α units of mass that it should send. Therefore, by scaling the given flow we can route this mass to nodes in X_F with congestion $O(\frac{\alpha}{\tau})$. By Fact 2.7 we get that this routing is $\Omega(\frac{\tau}{\alpha})$-respected by $G'[S']$. In particular, we can replace the demand state $P^{(1)}_{L'}$ with the updated demand state $P^{(2)}_{L'}$ after this routing, such that $G'[S']$ $\Omega(\frac{\tau}{\alpha})$-respects $P^{(1)}_{L'} - P^{(2)}_{L'}$. To be precise, we use the fact that $P^{(1)}_{L'} - P^{(2)}_{L'}$ is a valid demand state (recall Fact 3.19).

After this step, any node in X_F has load of at most $2 \cdot \frac{2\alpha}{\tau} + 2\alpha \leq \frac{6\alpha}{\tau}$ units in $P^{(2)}_{L'}$ (as we scaled the flow by $\frac{2\alpha}{\tau}$ and there was a load of at most 2α in the beginning). Note that at this stage, $P^{(2)}_{L'}$ is supported only on the vertices X_F (since $P^{(1)}_{L'}$ was supported on $X_F \cup X_Y$ and we moved the demands on $X_Y \cap L'$). This is similar to Remark 3.20. Note that $P^{(2)}_{L'}$ is not supported only on L', as some vertices of X_F may lie outside of L'.

Next, we update $P^{(2)}_{L'}$ by an all-to-all multi-commodity flow on the set X_F, in order to spread the demand state $P^{(2)}_{L'}$ uniformly on X_F. This is similar to Remark 3.20. In other words, consider the average demand vector $N_F := \frac{1}{|X_F|}\sum_{v \in X_F} P^{(2)}_{L'}(v)$. Theorem 4.1 gives that X_F $\Omega(\frac{1}{\log n})$-expands in $G'[S']$. Using Remark 3.6, we have that the all-to-all flow problem on X_F,[12] is $\Omega(\frac{1}{\log n})$-respected by $G'[S']$. If we scale this problem by $O(\frac{\alpha}{\tau})$, we get that the updated demand state is $P^{(3)}_{L'}$, where each demand vector is N_F. The valid demand state $P^{(2)}_{L'} - P^{(3)}_{L'}$ is therefore $\Omega(\frac{\tau}{\alpha \log n})$-respected by $G'[S']$. Using Fact 3.17, we get that $P^{(1)}_{L'} - P^{(3)}_{L'}$ is $\Omega(\frac{\tau}{\alpha \log n})$-respected by $G'[S']$.

Finally, due to Fact 3.19, $P^{(1)}_{L'} - P^{(2)}_{L'}$ is a valid demand state and therefore:

$$N_F = \frac{1}{|X_F|}\sum_{v \in X_F} P^{(2)}_{L'}(v) = \frac{1}{|X_F|}\sum_{v \in V'} P^{(2)}_{L'}(v) = \frac{1}{|X_F|}\sum_{v \in V'} P^{(1)}_{L'}(v) = \frac{1}{|X_F|}\sum_{v \in L'} P^{(1)}_{L'}(v).$$

□

Let $P^{(3)}_{L'}$ be the demand state from Lemma 5.9. All demands in $P^{(3)}_{L'}$ are distributed uniformly among vertices in X_F. Hence, positive demand units and negative demand units of the same commodity are "paired-up" and cancel out.

Denote by $B^R \subseteq E(S, V \setminus S)$ the set of boundary edges that are incident to a node in R. Note that B^R may not be all of B, as some boundary edges can be incident to a node in L, as long as these boundary edges are also in Y (see Figure 4). Let $\tilde{Y} := Y \setminus B^R$.

A key observation is that because L is node in the hierarchical decomposition tree T, and the original P is respected by T, the amount of mass that has to cross the cut $(L, V \setminus L)$ is bounded by the capacity of this cut. This is the reason we included L in the hierarchical decomposition tree. We first prove this slightly more general claim:

Claim 5.10. *Let $X \subseteq V'$ and let $\tilde{P}$ be a demand state that is supported on X and distributed uniformly on it. Let P be a demand state that is 1-respected by T. Let $L \in V_T$ be a cluster in*

[12] Recall from Definition 2.2 that this is the flow problem where each vertex in X_F sends $\frac{1}{|X_F|}$ of its unique commodity to all other vertices in X_F.

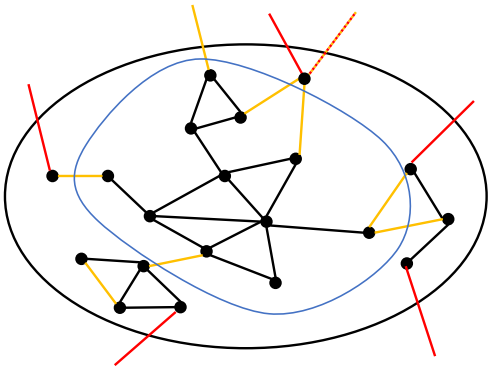


Figure 4: Visualization of the sets L, R, B^R, Y, and $\tilde{Y}$ in a cluster S. The core L is the region outlined in blue. Red edges correspond to B^R, and golden edges correspond to Y. The golden-red edge lies in $B^R \cap Y$. The set $\tilde{Y}$ is defined as $\tilde{Y} := Y \setminus B^R$.

the hierarchical decomposition and assume that $\sum_{v \in X} \tilde{P}(v) = \sum_{v \in L} P(v)$. *Then,* $\sum_{v \in X} \left\| \tilde{P}(v) \right\|_1 \leq \text{cap}(L, V \setminus L)$.

Proof. Because $\tilde{P}$ is spread uniformly on X, we have for each commodity $k \in \mathcal{K}$:

$$\left| \sum_{v \in X} \tilde{P}(v,k) \right| = \sum_{v \in X} \left| \tilde{P}(v,k) \right|.$$

Therefore,

$$\begin{aligned} \sum_{v \in X} \left\| \tilde{P}(v) \right\|_1 &= \sum_{v \in X} \sum_{k \in \mathcal{K}} \left| \tilde{P}(v,k) \right| = \sum_{k \in \mathcal{K}} \sum_{v \in X} \left| \tilde{P}(v,k) \right| \\ &= \sum_{k \in \mathcal{K}} \left| \sum_{v \in X} \tilde{P}(v,k) \right| = \sum_{k \in \mathcal{K}} \left| \sum_{v \in L} P(v,k) \right| = \text{dem}_P(L, V \setminus L). \end{aligned}$$

L corresponds to a node in T, and P is 1-respected by T. Therefore, we must have $\text{dem}_P(L, V \setminus L) \leq \text{cap}_G(L, V \setminus L)$, which gives the result. □

We can now prove the following corollary:

Corollary 5.11. *The total load on the nodes* X_F *in* $P^{(3)}_{L'}$ *satisfies*

$$\sum_{v \in X_F} \left\| P^{(3)}_{L'}(v) \right\|_1 \leq \text{cap}_G(\tilde{Y}).$$

Proof. For each commodity $k \in \mathcal{K}$, using Invariant 5.4(2), we have:

$$\sum_{v \in L'_i} P_{L'_i}(v,k) = \sum_{v \in L_i} P(v,k).$$

Therefore,

$$\sum_{v \in V'} P^{(1)}_{L'}(v,k) = \sum_{v \in L'} P^{(1)}_{L'}(v,k) = \sum_{v \in L} P(v,k),$$

where the first equality holds since $P^{(1)}$ is supported on L'. Because the updates that moved the demand vectors from X_Y to X_F and the all-to-all routings do not change the total demand (*i.e.*, the difference $P^{(3)}_{L'} - P^{(1)}_{L'}$ is a valid demand state), we also get

$$\sum_{v \in X_F} P^{(3)}_{L'}(v,k) = \sum_{v \in V'} P^{(3)}_{L'}(v,k) = \sum_{v \in V'} P^{(1)}_{L'}(v,k) = \sum_{v \in L} P(v,k),$$

where the first equality is because $P^{(3)}_{L'}$ is supported on X_F. Therefore, we get that the demand state vectors satisfy $\sum_{v \in X_F} P^{(3)}_{L'}(v) = \sum_{v \in L} P(v)$.

Applying Claim 5.10 with $X := X_F, \tilde{P} := P^{(3)}_{L'}$ gives that $\sum_{v \in X_F} \left\| P^{(3)}_{L'}(v) \right\|_1 \leq \mathrm{cap}_G(L, V \setminus L)$. Since $\tilde{Y}$ is a super-set of the edges that leave the core L in G (see Figure 4), we must have $\mathrm{cap}_G(L, V \setminus L) \leq \mathrm{cap}_G(\tilde{Y})$, which gives the result. □

Next, we distribute the remaining demands in $P^{(3)}_{L'}$ of nodes in X_F to nodes in $X_{\tilde{Y}}$.

Lemma 5.12. *We can replace $P^{(3)}_{L'}$ by a demand state $P^{(4)}_{L'}$ which satisfies the following:*

1. *$P^{(4)}_{L'}$ is supported on $X_{\tilde{Y}}$. Moreover, $\left\| P^{(4)}_{L'}(x_e) \right\|_1 \leq 1$, for every $x_e \in X_{\tilde{Y}}$.*
2. *$G'[S']$ $\Omega(\frac{\tau}{\log n})$-respects $P^{(3)}_{L'} - P^{(4)}_{L'}$.*

Proof. By Property 2 of Theorem 4.10 and since X_F $\Omega(\frac{1}{\log n})$-expands in $G'[S']$, the multi-commodity flow problem which "sends" one unit of flow from every $x_y \in X_{\tilde{Y}} \subseteq X_Y$ to a uniform distribution over X_F is $\Omega(\frac{\tau}{\log n})$-respected by $G'[S']$. By reversing these flows, we can distribute the demands of $P^{(3)}_{L'}$ remaining on X_F uniformly on the nodes in $X_{\tilde{Y}}$. By Corollary 5.11, each node $x_e \in X_{\tilde{Y}}$ receives at most one unit of demand. The new demand state $P^{(4)}_{L'}$ satisfies that $P^{(3)}_{L'} - P^{(4)}_{L'}$ is $\Omega(\frac{\tau}{\log n})$-respected by $G'[S']$. □

We are ready to prove Theorem 5.5.

Proof of Theorem 5.5. Denote $P^{(1)}_{R'} := \sum_{i=1}^{r} P_{R'_i}$. Observe that $P^{(1)}_{S'} = P^{(1)}_{L'} + P^{(1)}_{R'}$. We apply Lemmas 5.9, and 5.12 consecutively. Let $P^{(4)}_{S'} = P^{(4)}_{L'} + P^{(1)}_{R'}$ be the resulting demand state. Observe that $P^{(4)}_{S'}$ is not supported on any node in $X_F \setminus X_Y$. Additionally, $P^{(4)}_{L'}$ is supported only on $X_{\tilde{Y}}$.

To summarize the state so far, note that only vertices in $X_B \cup X_Y$ have non-zero demands. The vertices of $X_B \setminus X_Y \subseteq B^R$ have load at most α (this load comes from $P^{(1)}_{R'}$). The vertices of $X_Y \setminus X_B$ have load at most $2\alpha + 1$: at most 2α from $P^{(1)}_{R'}$ and at most 1 unit from $P^{(4)}_{L'}$. Crucially, the vertices in $X_B \cap X_Y$ also have at most α units of demand in $P^{(4)}_{S'}$. Indeed, if $x_e \in X_B \cap X_Y$ is incident to a node $v \in R$ (i.e., $x_e \in X_{B^R}$), then x_e carries at most α units from $P^{(1)}_{R'}$ and no additional units from $P^{(4)}_{L'}$. Otherwise, x_e is incident to a node $v \in L$, so $x_e \in X_{\tilde{Y}}$. Therefore, x_e does not have any demand from $P^{(1)}_{R'}$, and has at most $1 \leq \alpha$ units from $P^{(4)}_{L'}$.

Claim 5.13. *We can route a flow in $G'[S']$ which moves the demands in $P^{(4)}_{S'}$ from $X_Y \setminus X_B$ to the boundary nodes X_B, such that each node $x_b \in X_B$ receives at most $6\alpha\tau$ units of demand (in addition to at most α units of existing load already on these vertices in $P^{(4)}_{S'}$). This flow can be routed with congestion $O(\alpha)$.*

Proof. Property 1 of Theorem 4.10 promises a flow routable in $G'[S']$ with congestion $O(1)$, which sends 1 unit of flow from every node $x_y \in X_Y \setminus X_B$, such that every node $x_b \in X_B$ receives at most 2τ units of flow. To get the flow for Claim 5.13, we scale this flow by $2\alpha + 1 \leq 3\alpha$. □

We can therefore update $P^{(4)}_{S'}$ with the routing of Claim 5.13 to get the new demand state $P^{(5)}_{S'}$, such that $P^{(4)}_{S'} - P^{(5)}_{S'}$ is $\Omega(\frac{1}{\alpha})$-respected by $G'[S']$.

We can now wrap up the proof of Theorem 5.5. The resulting demand state $P^{\text{after}}_{S'}$ in Theorem 5.5 is $P^{(5)}_{S'}$. It satisfies Invariant 5.4 with $\alpha' = \alpha \cdot (1 + 6\tau)$. Indeed, by Claim 5.13, the demands of $P^{(5)}_{S'}$ only reside on nodes of X_B and the demand on each node is bounded by α'.

Finally, by Fact 3.17, we can see that $G'[S']$ γ-respects $P^{(1)}_{S'} - P^{(5)}_{S'}$, where

$$\gamma = \frac{1}{O\left(\frac{\alpha \log n}{\tau}\right) + O\left(\frac{\log n}{\tau}\right) + O(\alpha)} = \Omega\left(\frac{\tau}{\alpha \log n}\right).$$

This completes the proof of Theorem 5.5. □

Remark 5.14. *In the context of oblivious routing schemes, the construction of [RST14] was oblivious in the sense that the flow paths they construct between each pair of vertices s, t were independent of the demands.*

Similarly, in our case, the charges on the edges of (B, W) may be defined for each pair s, t, independently of the demands of P. The construction guarantees that for any valid demand state P, which is $\Omega(1)$-respected by G (and thus by T), if we add up the "oblivious" charges incurred by each s, t pair, scaled up by the demands of P, we will incur a charge of $O(\log^3 n)$ on each edge.

6 Improved Tree Cut-Sparsifier

In this section, we prove Theorem 2.8, establishing an improved tree cut-sparsifier of quality $O(\log^2 n \log\log n)$.

We begin by summarizing the results from Sections 4 and 5:

Theorem 6.1. *There is an algorithm "merge-phase" that given a cluster S and $0 < \tau \leq 1$, runs in $\tilde{O}(m)$ time and partitions S into sub-clusters $S_1, \ldots, S_k$, such that:*

1. *$|S_i| \leq \frac{2}{3}|S|$ for each i, and*
2. *For every collection of demand states $\{P_{S'_i}\}_{i=1}^k$ that satisfy Invariant 5.4 with load α, we can replace $P^{\text{before}}_{S'} = \sum_{i=1}^k P_{S'_i}$ by a demand state $P^{\text{after}}_{S'}$ that satisfies Invariant 5.4 with load $\alpha' = \alpha \cdot (1 + c \cdot \tau)$ for some constant $c > 0$, and*
3. *$G'[S']$ $\Omega(\frac{\tau}{\alpha \log n})$-respects $P^{\text{before}}_{S'} - P^{\text{after}}_{S'}$.*

In this section we show how to improve the quality of the resulting tree cut (and flow) sparsifier by a factor of $\Theta\left(\frac{\log n}{\log\log n}\right)$, thus proving Theorem 2.8.

Recall from Section 4 that we paid a factor of $\frac{1}{\tau}$ in the quality of the tree cut-sparsifier, in order to accumulate a factor of just $1 + O(\tau)$ to the load of each boundary edge as we move up the tree. This forced us to choose $\tau = O(\frac{1}{\log n})$ to make the total load $O(1)$ in each step. We wish to avoid paying this additional factor of $\frac{1}{\tau} = \Omega(\log n)$. Recall that even though each sub-cluster S is a node in the hierarchical decomposition tree (and therefore the total amount of demand that has

to leave it is at most the capacity of its boundary, because P is respected by T), the distribution of demands on the boundary of S is not uniform, so positive and negative demands of the same commodity do not cancel out.

We amend this using a variation on the "refinement phase" technique of [BKR03, HHR03, RS14],[13] to ensure that the all-to-all flow problem (recall Definition 2.2) between the boundary edges of the sub-clusters we recur to, is respected by the sub-clusters, meaning that the boundary edges expand in these sub-clusters (see Remark 3.6).[14]

The resulting scheme is a two-level decomposition: given an input cluster S, we first run one level of the merge phase decomposition described in Theorem 6.1, with $\tau = 1$. This partitions S to sub-clusters $\{S_1, \ldots, S_k\}$, such that $|S_i| \leq \frac{2}{3}|S|$. This merge-phase partition of S corresponds to 2 layers in the hierarchical decomposition (see Figure 5). On each S_i we then run the refinement phase described below. This partitions S_i into $\{S_{i,1}, \ldots, S_{i,k_i}\}$, such that the boundary edges of $S_{i,j}$ (*i.e.*, $E(S_{i,j}, V \setminus S_{i,j})$) expand within $G'[S'_{i,j}]$. Formally, we denote the boundary edges of $S_{i,j}$ as $B_{i,j} := E(S_{i,j}, V \setminus S_{i,j})$ and prove that $X_{B_{i,j}}$ $\Omega\left(\frac{1}{f(S_{i,j})\cdot\log n}\right)$-expands in $G'[S'_{i,j}]$, where f is defined in Theorem 6.2. The reason we need the merge phase is that we have no guarantee that the size of $S_{i,j}$ is smaller than S_i by a constant factor. Therefore, we repeat by performing an alternation between the merge phase on each $S_{i,j}$ and then the refinement phase on each produced sub-cluster, etc.

Even though $\tau = 1$, in each sub-cluster $S_{i,j}$ created by the refinement phase, we can send the demands to a uniform distribution on the boundary edges of $S_{i,j}$, so positive and negative demands cancel out and we are left with at most a load of 1 on each boundary edge. This effectively "resets" the load on each boundary edge so it does not accumulate when we go up along the hierarchical tree. This scheme allows us to improve the charge incurred, in total over all merge phase sub-clusters, from $O(\log^3 n)$ (by setting $\tau = \frac{1}{\log n}$) to $O(\log^2 n)$ ($\tau = 1$) per cut edge. As for the refinement phase sub-clusters, we will show that they incur a total of $O(\log^2 n \log\log n)$ charge per cut edge.

The main difference from [BKR03, HHR03, RS14] is that we are able to run the refinement phase in near-linear time. Additionally, we avoid dealing with signaled events from recursive sub-calls, leading to a simpler algorithm. Explicitly, we prove the following theorem:

Theorem 6.2. *There is an algorithm "refinement-phase" that given a cluster S and a parameter $\sigma \geq \frac{3}{2}|S|$, runs in $\tilde{O}(m)$ time and partitions S into sub-clusters $S_1, \ldots, S_k$, such that:*

1. *With high probability, for each i, the sub-cluster $G'[S'_i]$, $\Omega\left(\frac{1}{f(S_i)\cdot\log n}\right)$-respects the all-to-all flow problem[15] on X_{B_i}. Here $f(S_i) := c_f \log\left(\frac{\sigma}{|S_i|}\right) \cdot \log\log n$, where $c_f \geq 1$ is a constant which will be defined below.*
2. *We can route in $G'[S']$ a multi-commodity flow problem where each inter-cluster split node $x_f \in X_F$ (where $F = \{(u,v) \in E \mid u \in S_i, v \in S_j, i \neq j\}$) sends one unit of flow, and each boundary split node in X_B (where $B = E(S, V \setminus S)$) receives at most $O(1)$ units of flow. This can be routed with congestion $O(\log n)$.*

We prove this theorem in Section 6.1. In the rest of this section we show how to improve the cut-sparsifier result.

[13] [HHR03] called it the "bandwidth phase" while [BKR03] called it the "assure-precondition" subroutine.

[14] This requirement is weaker than what was ensured in [BKR03, HHR03, RS14]. First, we only require the all-to-all flow problem to be respected by the sub-cluster, not routable as in [BKR03, HHR03]. Additionally, they ensured that a slightly stronger flow problem is routable, which implies the all-to-all problem on the boundary edges is routable (but not conversely). However, the boundary edges being expanding will be enough for our purposes.

[15] See Definition 2.2. Additionally, recall the definition of a graph respecting a flow problem in Definition 2.6.

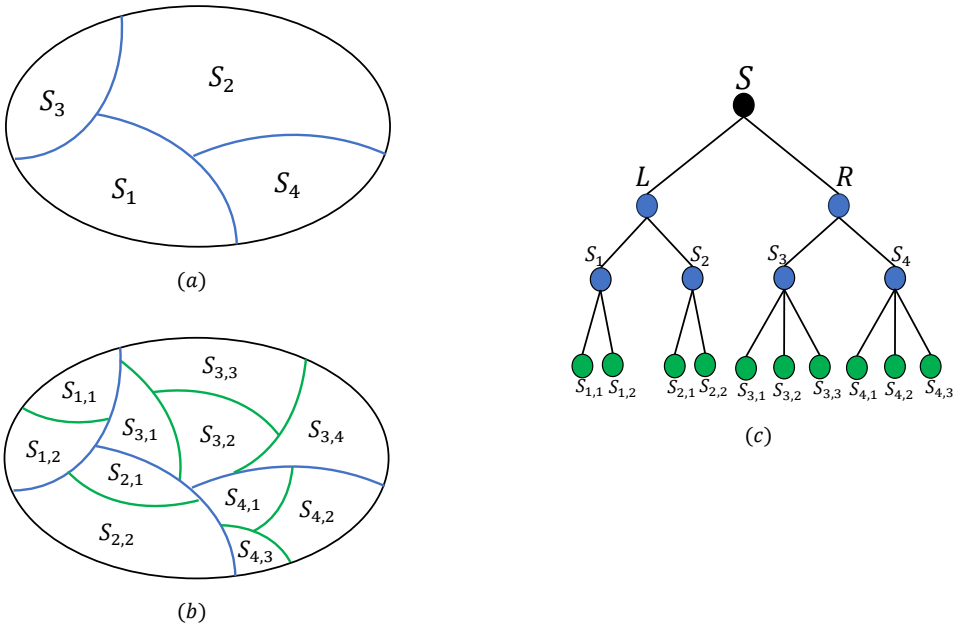


Figure 5: The two-step partition of a cluster S. The merge phase (Theorem 6.1) is shown in Figure (a). The refinement phase (Theorem 6.2) is then applied to the sub-clusters, as shown in Figure (b). The corresponding hierarchical tree is given in Figure (c). Note that the merge phase contributes two levels to the hierarchical tree due to the partition into L (the core) and R.

Our construction begins with the merge phase (Theorem 6.1) on $S = V$ with $\tau = 1$. Then, we run the refinement phase (Theorem 6.2) on each of the resulting sub-clusters S_i, with $\sigma = |S|$. We proceed with the merge phase with $\tau = 1$ on each resulting sub-cluster $S_{i,j}$, and then with the refinement phase on $S_{i,j,k}$, etc. In each subsequent run of the refinement phase, we choose σ to be the size of the previous cluster created by a refinement phase, *i.e.* the parent of the current sub-cluster (*e.g.*, when running on $S_{i,j,k}$, we take $\sigma = |S_{i,j}|$, not $|S_{i,j,k}|$.). Because S_i is a sub-cluster created in a merge phase, it will satisfy $|S_i| \leq \frac{2}{3} \cdot |V|$, ensuring the required condition on σ. We proceed recursively in this manner, running both stages on successively smaller clusters S, stopping when $|S| = 1$. Let T be the resulting hierarchical decomposition. Note that because the merge phase reduces the size of the sub-clusters by a constant fraction, the depth of T is logarithmic in n.

We begin by re-introducing the relevant notation from Section 5. As per Corollary 5.3, we fix a cut $(B \subseteq V, W = V \setminus B)$ in G and a valid demand state P on V, which is 1-respected by T. We work in the subdivision graph $G' = (V' \cup X_E, E')$. We define for a cluster $S \subseteq V$, the corresponding cluster in the subdivision graph as $S' := S \cup \{x_e \mid e \cap S \geq 1\}$ and let $G'[S']$ be the subdivision graph restricted to the set S'. We let $(B' = B \cup \{x_e \mid e \cap B \geq 1\}, W' = V' \setminus B')$ be the cut corresponding to (B, W) in G'. We will charge the edges $E(B', W')$ in G' to cover the demand $\mathrm{dem}_{P'}(B', W')$, where P' will be the appropriate demand state on G'.

Like in Section 5, we convert the valid demand state P to demand states $P'_{\{v\}}$ for each vertex $v \in V$, by distributing the demand vector of v to the split nodes neighboring v evenly. These demand states satisfy Invariant 5.4 with $\alpha = 1$. If we denote the sum of these demand states as $P_{\mathcal{Q}_0}$, Claim 5.8 shows that $\mathrm{dem}_P(B, W) \leq \mathrm{dem}_{P_{\mathcal{Q}_0}}(B', W') + \mathrm{cap}_G(B, W)$.

We will first prove the following improvement of Theorem 5.5.

Theorem 6.3. *Let S be a cluster, and let $S_1, \ldots, S_t$ be its merge-phase sub-clusters, computed with $\tau = 1$. For each $1 \le i \le t$, let $S_{i,1}, \ldots, S_{i,t_i}$ be the refinement-phase sub-clusters of S_i, computed with $\sigma = |S|$. For each $1 \le i \le t, 1 \le j \le t_i$, assume that $P_{S'_{i,j}}$ is a demand state satisfying Invariant 5.4 for $S_{i,j}$ with a given $\alpha \ge 1$. Let $P^{\text{before}}_{S'} = \sum_{i=1}^{t}\sum_{j=1}^{t_i} P_{S'_{i,j}}$ be their sum. We can replace $P^{\text{before}}_{S'}$ with a demand state $P^{\text{after}}_{S'}$ which satisfies Invariant 5.4 for S with $\alpha' = O(1)$, while charging each edge of the cut $E(B', W') \cap G'[S'_{i,j}]$ (for each i, j) at most $O(\alpha \cdot \log n \cdot f(S_{i,j}))$ to cover the difference in demands.*[16] *That is, we can define for each $e \in E(B', W') \cap G'[S']$ a $\text{charge}(e)$, such that $\sum_{e \in (B', W')} \text{charge}(e) \ge \text{dem}_{(P^{\text{before}}_{S'} - P^{\text{after}}_{S'})}(B', W')$ and each $e \in E(B', W') \cap G'[S'_{i,j}]$ is charged at most $O(\alpha \cdot \log n \cdot f(S_{i,j}))$. Note that the charged edges are in the subdivision graph.*

Proof. Let $P^{(1)}_{S'_{i,j}} := P_{S'_{i,j}}$. We apply the first guarantee of Theorem 6.2 on each of the clusters $S_{i,j}$ $(1 \le i \le t, 1 \le j \le t_i)$, to distribute the demand states $P^{(1)}_{S'_{i,j}}$ uniformly on the boundary edges of the cluster. Formally, define the average demand vector $N_{i,j} := \frac{1}{|X_{B_{i,j}}|} \sum_{v \in X_{B_{i,j}}} P^{(1)}_{S'_{i,j}}(v, \cdot)$, where $B_{i,j} = E(S_{i,j}, V \setminus S_{i,j})$ are the boundary edges of the cluster $S_{i,j}$. Consider the demand state $P^{(2)}_{S'_{i,j}}$ which is supported on $X_{B_{i,j}}$, where each demand vector is $N_{i,j}$. By construction, $P^{(2)}_{S'_{i,j}}$ satisfies Invariant 5.4 for $S_{i,j}$ with the same parameter α as $P^{(1)}_{S'_{i,j}}$. The updated demand state of $P^{(1)}_{S'_{i,j}}$ with respect to the all-to-all flow problem on $X_{B_{i,j}}$ scaled by α is $P^{(2)}_{S'_{i,j}}$ (see Remark 3.20). Using the first guarantee of Theorem 6.2, and by Facts 3.19 and 2.7, the valid demand state $P^{(1)}_{S'_{i,j}} - P^{(2)}_{S'_{i,j}}$ is $\Omega\left(\frac{1}{\alpha \cdot f(S_{i,j}) \cdot \log n}\right)$-respected by $G'[S'_{i,j}]$. Therefore, by charging $O(\alpha \cdot f(S_{i,j}) \cdot \log n)$ to each edge of $E(B', W') \cap G'[S'_{i,j}]$, we are able to cover the difference in demands $\text{dem}_{P^{(1)}_{S'_{i,j}}}(B', W') - \text{dem}_{P^{(2)}_{S'_{i,j}}}(B', W')$.

Now, all demands in $P^{(2)}_{S'_{i,j}}$ are distributed uniformly among vertices in $X_{B_{i,j}}$. Hence, positive demand units and negative demand units of the same commodity are "paired-up" and cancel out.

Next, we use the fact that $S_{i,j}$ is a sub-cluster in the hierarchical decomposition tree T, which means that because the original P is respected by T, the amount of demand P induces across the cut $(S_{i,j}, V \setminus S_{i,j})$ is at most the capacity of this cut. This is made explicit in the following claim:

Claim 6.4. *The total load in $P^{(2)}_{S'_{i,j}}$ on the nodes $X_{B_{i,j}}$ satisfies*

$$\sum_{v \in X_{B_{i,j}}} \left\| P^{(2)}_{S'_{i,j}}(v) \right\|_1 \le \text{cap}_G(B_{i,j}).$$

Proof. For each commodity $k \in \mathcal{K}$, by Invariant 5.4(2),

$$\sum_{v \in S'_{i,j}} P^{(1)}_{S'_{i,j}}(v, k) = \sum_{v \in S_{i,j}} P(v, k).$$

The update by the all-to-all flow problem does not change the total demand (*i.e.*, $P^{(1)}_{S'_{i,j}} - P^{(2)}_{S'_{i,j}}$ is a

[16] $f(S_{i,j})$ is defined in Theorem 6.2. Additionally, note that each edge in $G'[S']$ is part of exactly one sub-cluster $G'[S'_{i,j}]$.

valid demand state), so we get

$$\sum_{v \in X_{B_{i,j}}} P^{(2)}_{S'_{i,j}}(v,k) = \sum_{v \in S'_{i,j}} P^{(2)}_{S'_{i,j}}(v,k) = \sum_{v \in S_{i,j}} P(v,k) \ ,$$

where the first equality holds because $P^{(2)}_{S'_{i,j}}$ is supported on $X_{B_{i,j}}$. The result now follows by applying Claim 5.10 with $X := X_{B_{i,j}}, L := S_{i,j}$ and $\tilde{P} := P^{(2)}_{S'_{i,j}}$.

□

Using the claim, we have that the total load $P^{(2)}_{S'_{i,j}}$ puts on $X_{B_{i,j}}$ is at most $\mathrm{cap}_G(B_{i,j})$, and because the demand vectors are the same, we get that the load on each vertex is at most 1.

For every $1 \leq i \leq t$, let $P^{(2)}_{S'_i} := \sum_{j=1}^{t_i} P^{(2)}_{S'_{i,j}}$. Denote the inter-cluster edges inside S'_i by $F_i := \{(u,v) \in E \mid u \in S_{i,j_1}, v \in S_{i,j_2}, j_1 \neq j_2\}$ and the boundary edges of the cluster by $B_i := E(S_i, V \setminus S_i)$. We have that $P^{(2)}_{S'_i}$ is supported on $X_{F_i} \cup X_{B_i}$, such that the load on each of $x_f \in X_{F_i}$ is at most 2, and the load on each $x_b \in X_{B_i}$ is at most 1.

Next, we use the second guarantee from Theorem 6.2. This allows us to route in $G'[S'_i]$, one unit of mass from each $x_f \in X_{F_i}$ to X_{B_i}, such that each vertex $x_b \in X_{B_i}$ receives at most $O(1)$ units of mass. Scaling this flow by 2 gives that we can route two units from each inter-cluster edge to the boundary. This flow can be routed with congestion $O(\log n)$. Hence, by Fact 2.7, we can replace $P^{(2)}_{S'_i}$ with the updated demand state obtained after this routing, denoted $P^{(3)}_{S'_i}$, while incurring a charge of $O(\log n)$ on the edges $E(B', W') \cap G'[S'_i]$. Consequently, $P^{(3)}_{S'_i}$ satisfies Invariant 5.4 for S_i with $\alpha = O(1)$, as the demand state is supported on nodes of X_{B_i} and the load on each node is constant.

Denote $P^{(3)}_{S'} := \sum_{i=1}^{t} P^{(3)}_{S'_i}$. We now use Theorem 6.1 on the merge-phase partition $\{S_1, \ldots, S_t\}$. It allows us to replace $P^{(3)}_{S'}$ with the updated $P^{(4)}_{S'}$ which satisfies Invariant 5.4 for S with $\alpha = O(1)$, while charging each edge of $E(B', W') \cap G'[S']$ at most $O(\log n)$ to cover the difference in demands.

This $P^{\text{after}}_{S'} := P^{(4)}_{S'}$ is the demand state required by 6.3, and the total charge on each edge of $E(B', W') \cap G'[S']$ is dominated by $O(\alpha \cdot f(S_{i,j}) \cdot \log n)$. □

Using Theorem 6.3, Theorem 2.8 follows exactly like in Section 5. We give the full details for completeness.

Proof of Theorem 2.8. We deal with P in a bottom-up manner: first we convert it to a set of demand states $P_{\{v\}'}$ as shown in Section 5. Then, we use Theorem 6.3 for each cluster in T. We accumulate the charges incurred during each application of the theorem.

Denote by $\mathcal{Q}$ the set of current "active" clusters, such that initially $\mathcal{Q}_0 = \{\{v\} \mid v \in V\}$. Then in each iteration we remove $\{S_{i,j} \mid 1 \leq i \leq t, 1 \leq j \leq t_i\}$ from $\mathcal{Q}$ and add S to $\mathcal{Q}$. Let $P_{\mathcal{Q}}$ be the sum of all demand states $P_{S'}$, for $S \in \mathcal{Q}$. Using Claim 5.7, we have that $P_{\mathcal{Q}}$ is always a valid demand state on V'. Claim 5.8 shows that $\mathrm{dem}_P(B, W) \leq \mathrm{dem}_{P_{\mathcal{Q}_0}}(B', W') + \mathrm{cap}_G(B, W)$.

Consider a replacement of $\mathcal{Q}$ by $\mathcal{Q}' = \mathcal{Q} \cup \{S\} \setminus \{S_{i,j} \mid 1 \leq i \leq t, 1 \leq j \leq t_i\}$ when we apply Theorem 6.3. Recall that $P^{\text{before}}_{S'} = \sum_{i=1}^{t} \sum_{j=1}^{t_i} P_{S'_{i,j}}$ is replaced by $P^{\text{after}}_{S'}$. Let $P^{\text{before}}_{\mathcal{Q}'} := P_{\mathcal{Q}}$ be the sum of active demand states before the change and $P^{\text{after}}_{\mathcal{Q}'} := P_{\mathcal{Q}} - P^{\text{before}}_{S'} + P^{\text{after}}_{S'}$ be the sum after the change. We have $\mathrm{dem}_{\left(P^{\text{before}}_{\mathcal{Q}'} - P^{\text{after}}_{\mathcal{Q}'}\right)}(B', W') = \mathrm{dem}_{\left(P^{\text{before}}_{S'} - P^{\text{after}}_{S'}\right)}(B', W')$. This means that after accumulating all the charges in the applications of Theorem 5.5 on all clusters $S \in T$,

the total charge satisfies

$$\sum_{e\in E(B',W')} \text{charge}(e) = \sum_{e\in E(B',W')} \sum_{S\in T} \text{charge}_S(e) = \sum_{S\in T} \sum_{e\in E(B',W')\cap G'[S']} \text{charge}_S(e)$$
$$\geq \sum_{S\in T} \text{dem}_{(P^{\text{before}}_{S'} - P^{\text{after}}_{S'})}(B',W') \geq \text{dem}_{(P_{\mathcal{Q}_0} - P_{V'})}(B',W').$$

where charge_S is the charge incurred when applying the theorem on the cluster S, and the last inequality uses Fact 3.17. Next, note that by Invariant 5.4(1), because $P_{V'}$ satisfies the invariant for V, it holds that $P_{V'} = 0$. Thus, we have

$$\sum_{e\in E(B',W')} \text{charge}(e) \geq \text{dem}_{P_{\mathcal{Q}_0}}(B',W') \geq \text{dem}_P(B,W) - \text{cap}_G(B,W).$$

By charging an additional unit to each cut edge, we get $\text{dem}_P(B,W) \leq \sum_{e\in E(B',W')} \text{charge}(e)$.

We can bound the total charge on each edge as follows. The load α is $O(1)$ for each cluster, as it starts as $\alpha = 1$ and is constant after each application of Theorem 6.3. Therefore, each edge (in the subdivision graph) $e \in E(B',W')$, is charged by Theorem 6.3, $O(\log n \cdot f(S))$ for each refinement-phase cluster S that contains it (*i.e.*, $e \in G'[S']$). In particular, an edge $e \in E(B',W')$ is only charged by clusters S which lie on a root-to-leaf path in T.

Consider a path in the decomposition hierarchy T, $S_1 = V, \ldots, S_k$, where S_{i+1} is a refinement-phase sub-cluster of S_i (that is, we ignore the merge-phase sub-cluster between them). Note that for all $1 \leq i < k$, when S_{i+1} was created (as a refinement of some sub-cluster of S_i), we used $\sigma = |S_i|$ in Theorem 6.2. So, $f(S_{i+1}) = c_f \log\left(\frac{|S_i|}{|S_{i+1}|}\right) \cdot \log\log n$. Therefore, it holds that

$$\sum_{i=1}^{k} f(S_i) = f(S_1) + \sum_{i=1}^{k-1} f(S_{i+1}) = c_f \log\log n \cdot \left(\log\left(\frac{2n}{|S_1|}\right) + \sum_{i=1}^{k-1} \log\left(\frac{|S_i|}{|S_{i+1}|}\right)\right)$$
$$= c_f \log\log n \cdot \log\left(\frac{2n}{|S_k|}\right) = O(\log n \cdot \log\log n)$$

Therefore, the total charge is $O(\log^2 n \cdot \log\log n)$ per edge, which gives $\text{dem}_P(B,W) = O(\log^2 n \cdot \log\log n) \cdot \text{cap}_{G'}(B',W') = O(\log^2 n \cdot \log\log n) \cdot \text{cap}_G(B,W)$, as we wanted. Theorem 2.8 then follows from Corollary 5.3. □

As a direct corollary of Theorem 2.8, we get Theorem 2.11.

6.1 Refinement Phase

The goal of this section is to prove Theorem 6.2. Before delving into the proof in Section 6.1.2, in Section 6.1.1 we provide a high level overview of the techniques.

6.1.1 Overview

Given a cluster $S \subseteq V$, we would like to check if its boundary edges $B_S = E(S, V \setminus S)$ expands within $G[S]$.[17] If they are not, we would like to find a sparse cut $(A, S \setminus A)$ with respect to the corresponding weighting $\mu_S = \mathbb{1}_{B_S}$, that certifies that the boundary edges do not expand. We will

[17] The formal argument is performed in the subdivision graph (*i.e.*proving that X_{B_S} expands in $G'[S']$). See Section 6.1.2 for details. For this overview, we omit these details for simplicity.

then recur on both A and $S \setminus A$ (with the new weightings $\mu_A = \mathbb{1}_{B_A}$ and similarly $\mu_{S\setminus A} = \mathbb{1}_{B_{S\setminus A}}$). We stop when we find a sub-cluster in which the boundary edges expand (equivalently, the sub-cluster expands with respect to the weighting of the boundary edges). This can be facilitated using Theorem 3.9. Note that when we recur into a sub-cluster (*e.g.*, $A \subseteq S$), the new cut edges $E(S, A\setminus S)$ are now considered boundary edges of the sub-clusters A and $S \setminus A$. However, because the cut was sparse, the capacity of these cut edges is small compared to the already-present boundary edges, which ensures that the total capacity of the boundary edges decreases. Note that we must carefully adjust the target expansion ϕ when we invoke Theorem 3.9 on the different sub-clusters, to ensure that the final sub-clusters $S_i \subseteq S$, $\Omega\left(\frac{1}{f(S_i)\cdot\log n}\right)$-expand with respect to the boundary edges.

We can imagine the sequence of cuts and recursions as a binary tree whose root is S, and where in each step, the children of the current node are the two sides of the cut. Because the cut is balanced, we can ensure that the depth of the tree is poly-logarithmic, so the total running time is $\tilde{O}(m)$.

As for the second result of Theorem 6.2, we would like to use the flows that are promised by Remark 3.10 to be able to route from the cut edges $E(A, S \setminus A)$ in each iteration, to the boundary edges of A or of $S \setminus A$. In this manner, we could route the flow from the inter-cluster edges F to the boundary edges B_S by going over the binary tree from the leaves upwards: each time we encounter a cut $(A, D \setminus A)$ of the current sub-cluster $D \subseteq S$, we route the mass currently residing on the cut edges to the boundary edges of either A or $D \setminus A$. Note that the demands on the boundary edges accumulate as we perform this step. In particular, after accounting for all relevant details, we obtain that routing a unit of flow from each edge of $E(A, D \setminus A)$ to A adds $O\left(\frac{1}{f(D)}\right)$ units of load to the boundary edges of A (and similarly if we route to the boundary of $D \setminus A$). Assuming the load on the boundary edges of A and on the cut edges $E(A, D \setminus A)$ (accumulated in the lower levels of the binary tree) was α, this means that when we go upwards in the tree, the load on the boundary of D will now be $\alpha' = \alpha \cdot \left(1 + \frac{O(1)}{f(D)}\right)$.

If we could always route to the side with the smaller number of vertices (*i.e.*, route to A if $|A| \leq |D \setminus A|$ and to $|D \setminus A|$ otherwise), this approach would work. The reasoning is as follows. Since $f(D)$ is defined as $c_f \cdot \log\left(\frac{\sigma}{|D|}\right) \log\log n$ and $\sigma \geq \frac{3}{2}|S|$, it follows that if $|D| \leq |S| \cdot 2^{-j}$ for some $j \geq 1$, it implies $f(D) = \Omega(j \log\log n)$. This ensures that for each edge e, if we consider the sequence of clusters D_i (along the binary tree) in which the load on e increases (*i.e.*, when e is a boundary edge on the side with fewer vertices), the product $\prod_i \left(1 + \frac{1}{f(D_i)}\right)$ behaves like $\prod_{j=1}^{\log n}\left(1 + \frac{1}{j\log\log n}\right)$, so it remains bounded by a constant. Hence, the total load accumulated on each edge, determined by this product, is constant.

However, there is a problem with this approach. Specifically, as detailed in Remark 3.10, due to the internal algorithms of Theorem 3.9, the returned sparse cut $(A, S \setminus A)$ of a cluster S is created from a sequence of cuts $\{S_t\}_{t=0}^{T'-1}$, where if we define $A_0 = S$, $A_{t+1} = A_t \setminus S_t$, we have that $S_t \subseteq A_t$ is the cut we remove from the leftover cluster at step t. All of these cuts are sparse (*i.e.*, $(S_t, A_t \setminus S_t)$ is a sparse cut of A_t), and additionally we can route from the cut edges to both sides (S_t and $A_t \setminus S_t$).

Ultimately, when $A = S\setminus(\bigcup_t S_t)$, we can route flow from the cut edges $(A, S\setminus A)$ to the side $S\setminus A$ (see Figure 6(2*a*)). However, if we attempt to route instead towards A, we may incur congestion on the same edges multiple times (specifically, T' times), preventing a routing with sufficiently low congestion. We emphasize that this is not an issue in $S \setminus A$, because each cut is removed from the graph after it is processed: we route from $E(S_t, A_t \setminus S_t)$ to S_t **within** $G[S_t]$, and then S_t is removed from the remaining graph. As a result, the corresponding flow routings remain disjoint

(see Figure 6(2*a*)).

This routing is good for us if $|S \setminus A| \leq |A|$, but we have no such guarantee from Theorem 3.9 (as the theorem only bounds $\mu(S \setminus A)$, which is the number of boundary edges). Instead, we choose to do something slightly more nuanced: we stop the sequence of cuts $\{S_t\}_{t=0}^{T'}$ early at some $T_0 < T'$, as soon as its size $\left|\bigcup_{t=0}^{T_0} S_t\right|$ is a constant fraction of $|S|$. In other words, we find the first T_0 such that $\left|\bigcup_{t=0}^{T_0} S_t\right| = \Theta(|S|)$. Denote $A = S \setminus \left(\bigcup_{t=0}^{T_0} S_t\right)$. Assuming that at this step, both sides of the cut are of size $\Theta(|S|)$, this is enough to ensure that the recursion depth is poly-logarithmic (even though $\mu(S \setminus A) = \mu\left(\bigcup_{t=0}^{T_0} S_t\right)$ may be arbitrarily small), and allows us to bound $|S \setminus A| = \left|\bigcup_{t=0}^{T_0} S_t\right| = \Theta(|S|)$ (see Figure 6(2*b*)).

The final obstacle with this approach is that it may be that while $\left|\bigcup_{t=0}^{T_0-1} S_t\right|$ is small (say $o(|S|)$), it's possible $\left|\bigcup_{t=0}^{T_0} S_t\right|$ is already too large (say $(1 - o(1)) \cdot |S|$). In this case we perform two steps: we denote $A_1 := S \setminus \left(\bigcup_{t=0}^{T_0-1} S_t\right)$, which is a cut of S, and then further cut A_1 into $(A_2 := A_1 \setminus S_{T_0}, S_{T_0})$. Routing from $E(A_1, S \setminus A_1)$ to boundary of the side $S \setminus A_1$ is done like before (as mentioned, routing to the side $S \setminus A_1$ does not incur multiple congestion on the same edges, because in $S \setminus A_1 = \bigcup_{t=0}^{T_0-1} S_t$, the cuts are removed at each step). Routing from $E(A_2, S \setminus A_2)$ to the boundary of A_2 is possible because S_{T_0} now represents a single inner cut rather than a sequence of cuts. Consequently, we do not incur the congestion multiple times (see Figure 6(2*c*)).

This provides a high-level overview of the partitioning algorithm when Theorem 3.9 terminated with Case (2). A similar, though slightly simpler, argument applies when the theorem terminates with Case (3).

In total, this partitioning scheme guarantees that we always route from the cut edges to the boundary edges of a side whose size is at most $c \cdot |S|$. This suffices to show that the total accumulated load on each boundary edge remains bounded by a constant.

The full details are provided in Section 6.1.2. It begins with Corollary 6.5, a refined version of Theorem 3.9, which bounds the number of vertices on the side of the cut to which flow from the cut edges can be routed. Using this corollary, we then present the details of the aforementioned binary tree and prove Theorem 6.2.

6.1.2 Proof of Theorem 6.2

We are going to apply Theorem 3.9 on the cluster $S \subseteq V$. However, as discussed in Section 6.1.1, we require a slightly refined corollary of this theorem. Following the intuition presented in Section 6.1.1, one should think about the measure ν used in the corollary below as simply being the number of vertices $\nu(A) = |A|$.[18] Recall the notion of routing from a cut to a set from Section 3.3.

Corollary 6.5. *Given a graph $G = (V, E)$ of m edges, a parameter $\phi > 0$, and two weightings $\mu, \nu : V \to \mathbb{R}_{\geq 0}$ of the vertices, there exists a randomized algorithm which takes $\tilde{O}(m)$ time and must end in one of the following cases:*

1. *We certify that G has μ-expansion $\Phi^{\mu}(G) = \Omega(\phi)$, with high probability.*

2a. *We find a cut $(A, V \setminus A)$ of V with μ-expansion $\Phi^{\mu}_G(A, V \setminus A) = O(\phi \log n)$, and $\mu(V \setminus A) = \Omega\left(\frac{\mu(V)}{\log n}\right)$. Additionally $\nu(V \setminus A) \leq \frac{1}{4}\nu(V)$. There exists a flow routable with congestion $O(1)$*

[18] As will be described below, we use Corollary 6.5 on the subdivision graph and $\nu(A)$ will actually be the number of "non-split vertices" in A, *i.e.*, $\nu(A) = |A \cap V|$.

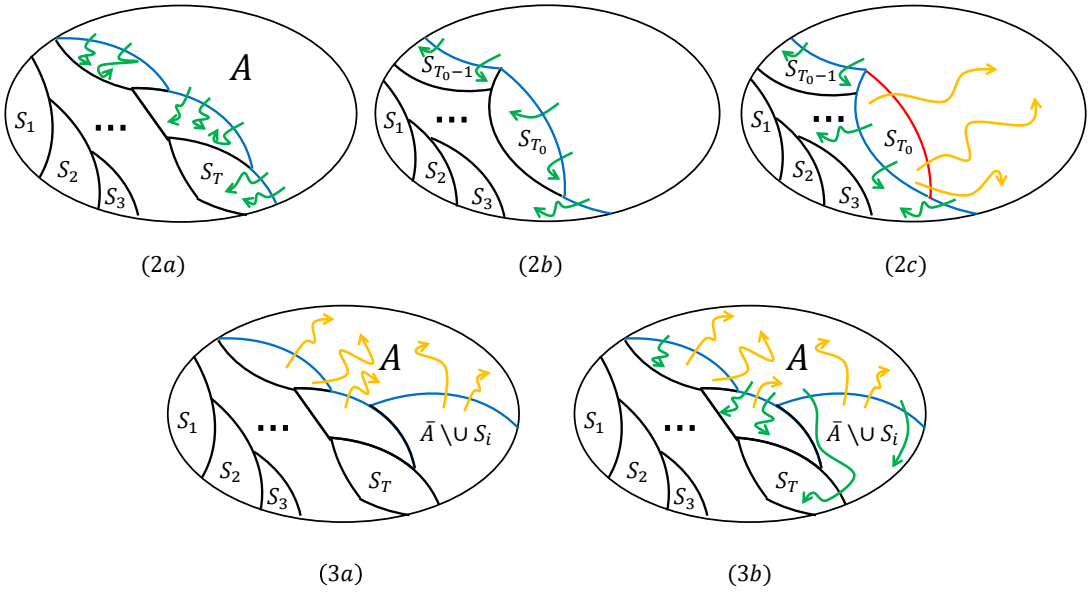


Figure 6: The 5 cases of Corollary 6.5. In all cases except case (2*c*), the left side of the blue cut corresponds to $V \setminus A$, where in case (2*c*) it corresponds to $V \setminus A_1$. The green flow depicts the guaranteed flow from Corollary 6.5. In cases (2*c*) and (3*b*) the corollary guarantees an additional flow, depicted in gold. The red cut in case (2*c*) corresponds to the cut $(A_1 \setminus A_2, A_2)$.

in $G[V \setminus A]$, such that each cut edge of $E(A, V \setminus A)$ is a source of one unit of flow[19] *and each $v \in V \setminus A$ receives at most $O(\phi \log n \cdot \mu(v))$ units.*

2b. We find a cut $(A, V \setminus A)$ of V with μ-expansion $\Phi^{\mu}_G(A, V \setminus A) = O(\phi \log n)$. Additionally $\frac{1}{4}\nu(V) \leq \nu(V \setminus A) \leq \frac{1}{2}\nu(V)$. There exists a flow routable with congestion $O(1)$ in $G[V \setminus A]$, such that each cut edge of $E(A, V \setminus A)$ is a source of one unit of flow and each $v \in V \setminus A$ receives at most $O(\phi \log n \cdot \mu(v))$ units.

2c. We find a cut $(A_1, V \setminus A_1)$ of V and a cut $(A_2, A_1 \setminus A_2)$ of A_1, with μ-expansion $\Phi^{\mu}_G(A_1, V \setminus A_1), \Phi^{\mu}_G(A_2, A_1 \setminus A_2) = O(\phi \log n)$. Additionally, $\mu(A_2) \geq \frac{1}{4}\mu(V)$ and $\nu(A_1 \setminus A_2) \geq \frac{1}{4}\nu(V)$. There exists a flow routable with congestion $O(1)$ in $G[V \setminus A_1]$, such that each cut edge of $E(A_1, V \setminus A_1)$ is a source of one unit of flow and each $v \in V \setminus A_1$ receives at most $O(\phi \log n \cdot \mu(v))$ units. There also exists a flow routable with congestion $O(1)$ in $G[A_2]$, such that each cut edge of $E(A_2, A_1 \setminus A_2)$ is a source of one unit of flow and each $v \in A_2$ receives at most $O(\phi \log n \cdot \mu(v))$ units.

3a. We find a cut $(A, V \setminus A)$ with μ-expansion $\Phi^{\mu}_G(A, V \setminus A) = O(\phi \log n)$. Additionally, $0 < \mu(V \setminus A) \leq \frac{\mu(V)}{2}$ and $\nu(V \setminus A) \geq \frac{1}{2}\nu(V)$. There exists a flow routable with congestion $O(1)$ in $G[A]$, such that each cut edge of $E(A, V \setminus A)$ is a source of one unit of flow and each $v \in A$ receives at most $O(\phi \cdot \mu(v))$ units.

3b. We find a cut $(A, V \setminus A)$ with μ-expansion $\Phi^{\mu}_G(A, V \setminus A) = O(\phi \log n)$, and with high probability, $G[A]$ is an $\Omega(\phi)$-expander with respect to μ. Additionally, $\nu(V \setminus A) \leq \frac{1}{2}\nu(V)$. There exists

[19] Formally, there is a flow in $G[V \setminus A]$ routable with congestion $O(1)$, such that each $v \in V \setminus A$ sends $\deg_G(v) - \deg_{G[V \setminus A]}(v)$ units of flow and receives at most $O(\phi \log n \cdot \mu(v))$ units. The rest of the flows in this corollary are defined similarly.

a flow routable with congestion $O(1)$ in $G[V \setminus A]$, such that each cut edge of $E(A, V \setminus A)$ is a source of one unit of flow and each $v \in V \setminus A$ receives at most $O(\phi \log n \cdot \mu(v))$ units. Additionally, there exists a flow routable with congestion $O(1)$ in $G[A]$, such that each cut edge of $E(A, V \setminus A)$ is a source of one unit of flow and each $v \in A$ receives at most $O(\phi \cdot \mu(v))$ units.

Proof. We apply Theorem 3.9 on G. If it terminated with case (1), we are done. Otherwise, first assume it terminated with case (2). Recall from Remark 3.10 that $\bar{A}$ is given as $\bar{A} = \bigcup_t S_t$, where if we look at the sequence of vertex sets $A_0 = V, A_{t+1} = A_t \setminus S_t$, each (S_t, A_{t+1}) is a sparse cut. We split into cases according to $\nu(\bar{A})$. If $\nu(\bar{A}) \leq \frac{1}{4}\nu(V)$, we terminate with case (2a) (see Figure 6).

Next, assume that $\nu(\bar{A}) > \frac{1}{4}\nu(V)$. We consider the first $T_0 \geq 0$ such that $\nu\left(\bigcup_{t=0}^{T_0} S_t\right) > \frac{1}{4}\nu(V)$ (*i.e.*, $\nu\left(\bigcup_{t=0}^{T_0-1} S_t\right) \leq \frac{1}{4}\nu(V)$). If $\nu\left(\bigcup_{t=0}^{T_0} S_t\right) \leq \frac{1}{2}\nu(V)$ we terminate with case (2b), returning the cut $\left(V \setminus \bigcup_{t=0}^{T_0} S_t, \bigcup_{t=0}^{T_0} S_t\right)$.

Next, assume that $\nu\left(\bigcup_{t=0}^{T_0} S_t\right) > \frac{1}{2}\nu(V)$. This means that $\nu\left(S_{T_0}\right) > \frac{1}{4}\nu(V)$. We set $A_1 = V \setminus (\bigcup_{t=0}^{T_0-1} S_t)$ and $A_2 = A_1 \setminus S_{T_0}$ and terminate in case (2c). The guarantee $\mu(A_2) \geq \frac{1}{4}\mu(V)$ is given by Remark 3.10.

This concludes case (2) of Theorem 3.9. Next, assume Theorem 3.9 terminated with case (3). This means that with high probability $G[A]$ is an $\Omega(\phi)$-expander with respect to μ. Additionally, the cut $(A, V \setminus A)$ is a result of the trimming step (see Remark 3.10) and satisfies the properties mentioned in Remark 3.10. In particular, we have $0 < \mu(V \setminus A) \leq \frac{\mu(V)}{2}$. We split into cases according to $\nu(V \setminus A)$. If $\nu(V \setminus A) \geq \frac{1}{2}\nu(V)$, we terminate with case (3a). Otherwise, we terminate with case (3b).

In all cases, the required flows are given by Remark 3.10. □

We now proceed to prove Theorem 6.2. The proof follows the overview presented in Section 6.1.1.

Proof of Theorem 6.2. We are given a cluster $S \subseteq V$ and a parameter $\sigma \geq \frac{3}{2}|S|$. We do not apply Corollary 6.5 directly on the subdivision graph $G'[S']$. Instead, we will define the graph $G_{\tilde{S}}$ to be the graph $G[S]$ together with the boundary split nodes from $G'[S']$, but without the inner split nodes. Formally, we denote the boundary edges of S by $B_S := E(S, V \setminus S)$, and the edges in $G[S]$ by $E_S := \{e \in E \mid e \subseteq S\}$. We define $X_{B_S} := \{x_b \mid b \in B_S\}$, and $\tilde{S} = S \cup X_{B_S}$ then the graph $G_{\tilde{S}}$ is given by $G_{\tilde{S}} := (\tilde{S}, E_{\tilde{S}})$, where $E_{\tilde{S}} := E_S \cup \{(v, x_b) \mid v \in S, b \in B_S, v \in b\}$. We define the weighting μ_S on $G_{\tilde{S}}$ as $\mu_S = \mathbb{1}_{X_{B_S}}$. We always use $\nu(v) = \mathbb{1}_V$, *i.e.*, ν counts the number of non-split vertices.

We then apply Corollary 6.5 on $G_{\tilde{S}}$ with $\phi_S = \Theta\left(\frac{1}{\log n \cdot f(S)}\right)$. [20] The exact constant c_ϕ in $\phi_S = \frac{c_\phi}{\log n \cdot f(S)}$ will be $c_\phi \leq 1$ and is defined below. We split into cases according to the result of the corollary (see Figure 6). If Corollary 6.5 terminates with case (1), we terminate.

Otherwise, Corollary 6.5 returns a cut $(\tilde{A}, \tilde{S} \setminus \tilde{A})$ which satisfies $\Phi^{\mu_S}_{G_{\tilde{S}}}(\tilde{A}, \tilde{S} \setminus \tilde{A}) = O(\phi_S \log n)$, or (in case (2c)) two cuts $(\tilde{A}_1, \tilde{S} \setminus \tilde{A}_1), (\tilde{A}_2, \tilde{A}_1 \setminus \tilde{A}_2)$ which satisfy the sparsity condition. Denote $A := \tilde{A} \cap S$ (or $A_i := \tilde{A}_i \cap S$). Unless Corollary 6.5 terminated with case (3b), the refinement-phase recurs on all sub-clusters: A and $S \setminus A$ (in case (2c), $S \setminus A_1, A_1 \setminus A_2, A_2$). In case (3b), we only

[20] The reason we prefer to apply Corollary 6.5 on $G_{\tilde{S}}$ instead of $G'[S']$ is that applying it on the latter may create some *edge cases* which complicate the argument. For example, it may turn out that an edge $e = (v, u) \in E_S$ has both v and u on the same side of the cut while x_e is on the other.

recur on $S \setminus A$. [21] Note that in the recursive step on a sub-cluster $R \subseteq S$, we define μ_R on $G_{\tilde{R}}$ according to the boundary edges of this cluster, *i.e.*, $E(R, V \setminus R)$, which includes some of the previous boundary edges and newly cut edges. Additionally, we proceed with $\phi_R = \Theta\left(\frac{1}{\log n \cdot f(R)}\right)$.

Note that we do not recur into sub-clusters S with $\mu(S) = 0$. For such sub-clusters, we simply return the partition $\{S\}$, as it trivially satisfies both requirement of Theorem 6.2.

The algorithm is summarized in Algorithm 1.

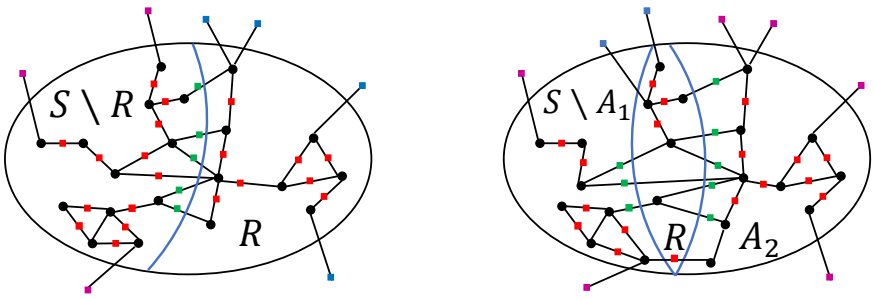
Algorithm 1 Refinement Phase

function REFINE(G, S, σ)
  $f(S) \leftarrow c_f \log\left(\frac{\sigma}{|S|}\right) \cdot \log\log n$.
  Call Corollary 6.5 on $G_{\tilde{S}}$ with $\phi_S = \Theta\left(\frac{1}{\log n \cdot f(S)}\right)$ and $\mu_S = \mathbb{1}_{X_{B_S}}$.
  if we get case (1) **then return** $\{S\}$.
  else if we get cases $(2a), (2b)$ or $(3a)$ **then**
    Let $(\tilde{A}, \tilde{S} \setminus \tilde{A})$ be the returned cut.
    $A \leftarrow \tilde{A} \cap S$.
    return Refine$(G, A, \sigma) \cup$ Refine$(G, S \setminus A, \sigma)$.
  else if we get case $(2c)$ **then**
    Let $(\tilde{A}_1, \tilde{S} \setminus \tilde{A}_1)$ and $(\tilde{A}_2, \tilde{A}_1 \setminus \tilde{A}_2)$ be the returned cuts.
    $A_1 \leftarrow \tilde{A}_1 \cap S$, $A_2 \leftarrow \tilde{A}_2 \cap S$.
    return Refine$(G, S \setminus A_1, \sigma) \cup$ Refine$(G, A_1 \setminus A_2, \sigma) \cup$ Refine(G, A_2, σ).
  else if we get case $(3b)$ **then**
    Let $(\tilde{A}, \tilde{S} \setminus \tilde{A})$ be the returned cut.
    $A \leftarrow \tilde{A} \cap S$.
    return $\{A\} \cup$ Refine$(G, S \setminus A, \sigma)$.
  end if
end function

First, to show that the running time is $\tilde{O}(m)$, we note that whenever we recur on a sub-cluster $R \subseteq S$, it holds that either:

- $|R| \leq \frac{3|S|}{4}$ (when we have $\nu(\tilde{R}) \leq \frac{3\nu(\tilde{S})}{4}$), or
- $\mu_S(\tilde{S} \setminus \tilde{R}) = \Omega\left(\frac{\mu_S(\tilde{S})}{\log n}\right)$ and $\Phi^{\mu_S}_{G_{\tilde{S}}}(\tilde{R}, \tilde{S} \setminus \tilde{R}) = O(\phi_S \log n)$. We show that in this case $\mu_R(\tilde{R}) = O(\mu_S(\tilde{S}) \cdot (1 - \frac{1}{\log n}))$. It holds that the new weighting μ_R satisfies $\mu_R(\tilde{R}) \leq \mu_S(\tilde{R}) + \text{cap}(\tilde{R}, \tilde{S} \setminus \tilde{R})$ (see Figure 7). We have that $\Phi^{\mu_S}_{G_{\tilde{S}}}(\tilde{R}, \tilde{S} \setminus \tilde{R}) \leq c \cdot \phi_S \log n$. We choose the constant c_ϕ in $\phi_S = \frac{c_\phi}{\log n \cdot f(S)}$ such that $\Phi^{\mu_S}_{G_{\tilde{S}}}(\tilde{R}, \tilde{S} \setminus \tilde{R}) \leq \frac{1}{2f(S)} \leq \frac{1}{2}$, which in particular means that $\text{cap}_{G_{\tilde{S}}}(\tilde{R}, \tilde{S} \setminus \tilde{R}) \leq \frac{1}{2}\mu_S(\tilde{S} \setminus \tilde{R})$. Thus, we have $\mu_R(\tilde{R}) \leq \mu_S(\tilde{R}) + \frac{1}{2}\mu_S(\tilde{S} \setminus \tilde{R}) = \mu_S(\tilde{S}) - \frac{1}{2}\mu_S(\tilde{S} \setminus \tilde{R}) = O(\mu_S(\tilde{S}) \cdot (1 - \frac{1}{\log n}))$.
- We have case (2c), for the sub-cluster $R = A_1 \setminus A_2$. In this case it may be that the cut $(\tilde{R}, \tilde{S} \setminus \tilde{R})$ is not sparse, if there are many edges in $E(\tilde{A}_1 \setminus \tilde{A}_2, \tilde{S} \setminus \tilde{A}_1)$. However, note that it does hold

[21] In the following, we assume that the cuts are *well-behaved*, in the sense that if (v, x_b) is an edge in $G_{\tilde{S}}$ with $x_b \in X_{B_S}$, then v and x_b are in the same side of the cut that Corollary 6.5 returns (*i.e.*, x_b is not an isolated vertex in the other side of the cut). If this does not hold, we may simply move x_b to the side where v resides. The presented arguments still hold, sometimes with slightly different constants.



(a) (b)

Figure 7: Visualizing the quantities $\mu_R(R'), \mu_S(R')$, and $\text{cap}(R', S' \setminus R')$. The quantity $\mu_R(R')$ corresponds to the blue and the green split nodes, while $\mu_S(R')$ corresponds only to the blue split nodes. Finally $\text{cap}_{G'[S']}(R', S' \setminus R') = \text{cap}_{G[S]}(R, S \setminus R)$ corresponds to the green nodes. Figure (b) corresponds to case $(2c)$ of Corollary 6.5, and Figure (a) corresponds to all other cases.

that $\mu_S(\tilde{R}) \leq \frac{3}{4}\mu_S(\tilde{S})$ and additionally $\Phi^{\mu_S}_{G_{\tilde{S}}}(\tilde{S} \setminus \tilde{A}_1, \tilde{A}_1), \Phi^{\mu_S}_{G_{\tilde{S}}}(\tilde{A}_1 \setminus \tilde{A}_2, \tilde{A}_2) = O(\phi_S \log n)$. We choose the constant c_ϕ in $\phi_S = \frac{c_\phi}{\log n \cdot f(S)}$ such that both sparsities are at most $\frac{1}{16 f(S)} \leq \frac{1}{16}$, giving that $\text{cap}(\tilde{R}, \tilde{S} \setminus \tilde{R}) \leq \text{cap}(\tilde{S} \setminus \tilde{A}_1, \tilde{A}_1) + \text{cap}(\tilde{A}_1 \setminus \tilde{A}_2, \tilde{A}_2) \leq 2 \cdot \frac{1}{16}\mu_S(\tilde{S}) = \frac{1}{8}\mu_S(\tilde{S})$. Therefore, $\mu_R(\tilde{R}) \leq \mu_S(\tilde{R}) + \text{cap}(\tilde{R}, \tilde{S} \setminus \tilde{R}) \leq \frac{3}{4}\mu_S(\tilde{S}) + \frac{1}{8}\mu_S(\tilde{S}) = \frac{7}{8}\mu_S(\tilde{S})$.

Therefore, the recursion depth is poly-logarithmic, so the running time is $\tilde{O}(m)$.

We now show that the first result of Theorem 6.2 holds when we stop recurring on a sub-cluster. Assume that when running in some sub-cluster $R \subseteq S$, Corollary 6.5 terminated with case (1). In this case, $G_{\tilde{R}}$ is $\Omega(\phi_R)$-expander with respect to μ_R, which means that X_{B_R} $\Omega(\phi_R)$-expands in $G_{\tilde{R}}$. Therefore, by Remark 3.6, $G_{\tilde{R}}$ $\Omega(\phi_R)$-respects the all-to-all flow problem on the vertices of X_{B_R}, which establishes the first result. Now, assume that Corollary 6.5 terminated with case (3b). This means that we have $A \subseteq R$ on which we do not recur, which satisfies that $G_{\tilde{A}}$ is an $\Omega(\phi_R)$-expander with respect to μ_R. Equivalently, $\tilde{A} \cap X_{B_R}$ $\Omega(\phi_R)$-expands in $G_{\tilde{A}}$. Denote by $C = E(A, R \setminus A)$ the set of cut edges that were cut in this step. We have that we can route a flow in $G_{\tilde{A}}$ with congestion $O(1)$, such that each edge $e \in C$ sends one unit of flow, and each $v \in \tilde{A}$ gets at most $O(\phi_R \cdot \mu_R(v)) = O(\mu_R(v))$ units. [22] This means that only boundary nodes in $\tilde{A} \cap X_{B_R}$ can receive flow, and each receives at most $O(1)$ units. Let X_C denote the new boundary edges in $G_{\tilde{A}}$, that correspond to the cut edges of C. Using Lemma 4.3, we get that $(\tilde{A} \cap X_{B_R}) \cup X_C = X_{B_A}$ $\Omega(\phi_R)$-expands in $G_{\tilde{A}}$. This means (Remark 3.6) that $G_{\tilde{A}}$ $\Omega(\phi_R)$-respects the all-to-all flow problem on the vertices of X_{B_A}. Finally, note that because $|A| \leq |R|$, we have $\phi_R = \Omega(\phi_A)$.

Next, we show the second result of Theorem 6.2. The argument follows the ideas of [HHR03, RS14]. Note that inter-cluster edges are only created as cut edges in cases (2a), (2b), (2c), (3a) and (3b) of Corollary 6.5.

We view the cuts formed by cutting S as a binary tree[23], denoted the refinement tree T^{ref}_S of S (see Figure 8). The root of the binary tree corresponds to S itself and every vertex corresponds to a subset $D \subseteq S$. The children of D in the binary tree are the subsets we recur on during the application of Corollary 6.5 on $G_{\tilde{D}}$. Note that in case (2c) we have that D splits into $A_1, D \setminus A_1$, and A_1 further splits into $A_2, A_1 \setminus A_1$. These are two levels of T^{ref}_S. We choose to set the left child of each node to be the side in which we can route flow from the cut edges:

[22] Recall from Section 3.3 the definition of routing a flow from the cut edges.

[23] This tree is for analysis purposes and should not be confused with the outer hierarchical decomposition.

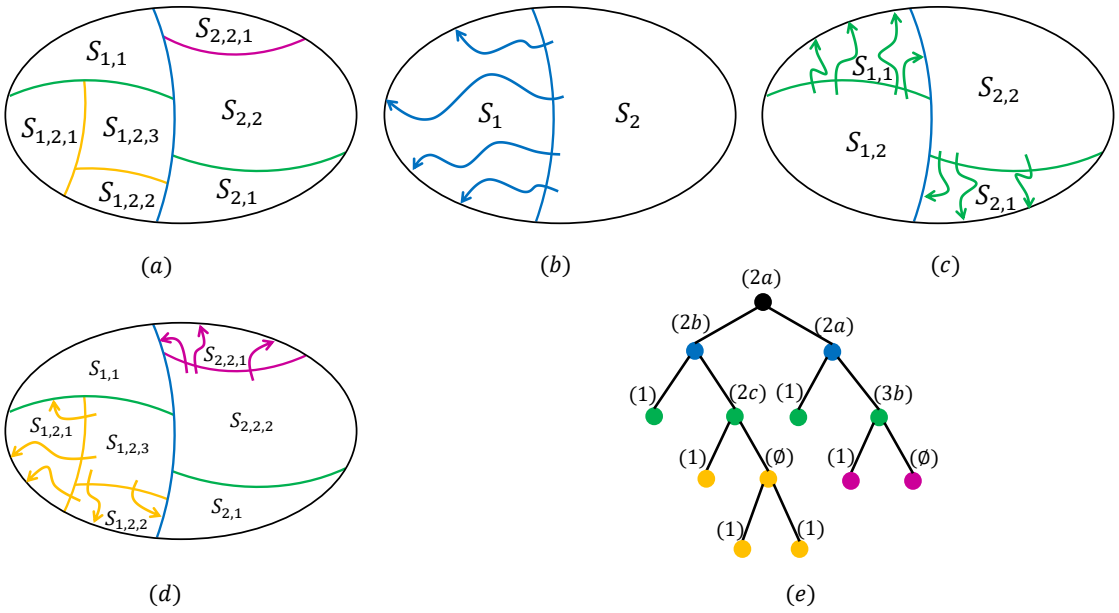


Figure 8: Figure (a) depicts several cuts computed by the refinement phase of a cluster S. Figures $(b) - (d)$ illustrate the flow routings from the cuts to the boundary of the smaller side, as guaranteed by Corollary 6.5. Figure (e) shows the binary tree that corresponds to these cuts: each node represents a sub-cluster obtained during the cutting process. The label on each node indicates the case of Corollary 6.5 that was applied. A label of $(\emptyset)$ means that Corollary 6.5 was not applied, and the node's state is instead inferred from its parent. The left child always corresponds to a side of the cut which is smaller (in terms of number of vertices) than the parent, by a constant factor.

1. In cases (2a), (2b) and (3b), $D \setminus A$ is the left child of D.
2. In case (2c), $D \setminus A_1$ is the left child of D and A_2 is the left child of A_1. See Figure 8(e).
3. In case (3a), A is the left child of D.

Note that in all cases the left child $L \subseteq D$ satisfies $|L| \leq \frac{3}{4}|D|$, and additionally, we can route in $G_{\tilde{L}}$ a flow with congestion $O(1)$ such that each cut edge of $E(L, D \setminus L)$ sends one unit of flow and each vertex $v \in \tilde{L}$ receives at most $O(\phi_D \log n \cdot \mu_D(v))$ units.[24] That is, each boundary split node of $X_{B_D} \cap \tilde{L}$ gets at most $O(\phi_D \log n)$ units. Let $c_0 \geq 1$ be such that in all cases each boundary split node in $X_{B_D} \cap \tilde{L}$ gets at most $c_0 \phi_D \log n$ units. In the definition of f in Theorem 6.2, we will choose $c_f := \frac{4c_0}{\log(\frac{4}{3})} \geq 1$.

To describe the routing in $G'[S']$, where each inter-cluster split node sends one unit of flow to the boundary of S, we proceed as follows.[25] We shift our focus from $G_{\tilde{S}}$ back to $G'[S']$, effectively adding a split node in the middle of each inner edge. We identify flows from cut edges (in $G_{\tilde{S}}$) with flows that route mass from the split nodes (in $G'[S']$) corresponding to the cuts' edges. We start with one unit of mass placed on each inter-cluster split node. The mass is then routed according to the binary tree: beginning from the leaves and proceeding upward iteratively. At each binary cut $(L, D \setminus L)$, we use the flows described in Corollary 6.5 to route the mass currently residing on the cut edges (*i.e.*, split nodes of cut edges) to the boundary split nodes of L, *i.e.*, to $X_{B_D} \cap \tilde{L}$. We scale the flow according to the current load on the cut edges. Next, we analyze the load on each boundary split node as a result of these routings.

We define the *left depth* of a node $D \subseteq S$ (we identify the node with the corresponding subset of S) in the binary tree to be the number of left branches on the path from the root of the tree (corresponding to S) to the node D, plus one. That is, the left depth of S is 1. We denote by $\Phi(j)$ the maximum load on a boundary edge of a cluster of left depth j, after we process this sub-cluster (*i.e.*, after we route from the cut edges $E(L, D \setminus L)$ induced at this cluster to its boundary edges $E(L, V \setminus D) \subseteq E(D, V \setminus D)$).

Note that the left depth of a node is at most $\log_{4/3} n \leq 3 \log n$. Therefore, any node of depth $3 \log n$ must be a leaf node, so $\Phi(3 \log n) = 1$ (no routings are done at the leaf level). Following the analysis of [HHR03], we now prove by induction on $j = 3 \log n, \ldots, 1$ the following claim:

Claim 6.6 (See [HHR03, Lemma 4])**.**

$$\Phi(j) \leq \prod_{l=j}^{3 \log n} \left(1 + \frac{1}{l \log \log n}\right)$$

Proof. Assume the claim is true for all $l \geq j+1$. Consider a node D at left depth j in the binary tree. Let $L \subseteq D$ be its left child. After processing D, the load on each boundary edge of $D \setminus L$ didn't change, while the load on each boundary edge of L may have increased. Indeed, a boundary edge of L has load of at most $\Phi(j+1)$ immediately after processing the sub-cluster L. This load then increases when processing the sub-cluster D, since we additionally route the load on the cut edges $E(L, D \setminus L)$ to it.

[24] In case (2c), this is true for D directly, while for A_1 it's true with ϕ_D and μ_D instead of ϕ_{A_1} and μ_{A_1}. However, because $|A_1| = \Theta(|D|)$, we have $\phi_{A_1} = \Theta(\phi_D)$. Additionally, we have $\mu_D \leq \mu_{A_1}$.

[25] Observe that, unlike in other parts of the paper where we reason about demand states, here we explicitly construct and route a flow.

During the processing of D, each cut edge in $E(L, D \setminus L)$ (*i.e.*, split node of a cut edge) has load at most $\Phi(j) + \Phi(j+1) \leq 2\Phi(j)$, because $\Phi(j+1)$ can come from the processing of L and $\Phi(j)$ from $D \setminus L$ (the inequality is because $\Phi(j) \geq \Phi(j+1)$).

Therefore, the flow from $X_{E(L,D\setminus L)}$ to the boundary split nodes of L is scaled by at most $2\Phi(j)$. Therefore, each boundary split node of $X_{E(L,V\setminus D)}$ gets at most $2\Phi(j) \cdot c_0 \phi_D \log n$ units of flow. Therefore, we have:

$$\Phi(j) \leq \Phi(j+1) + 2c_0\phi_D \log n \cdot \Phi(j) \leq \Phi(j+1) + \frac{2c_0 \log n}{f(D) \cdot \log n}\Phi(j) = \Phi(j+1) + \frac{2c_0}{f(D)}\Phi(j).$$

Rearranging, we get:

$$\Phi(j) \leq \Phi(j+1) \cdot \left(\frac{1}{1 - \frac{2c_0}{f(D)}}\right) \leq \Phi(j+1) \cdot \left(1 + \frac{4c_0}{f(D)}\right),$$

where the last inequality holds due to the inequality $1/(1-x) \leq 1 + 2x$ for $x \in [0, \frac{1}{2}]$, and indeed $0 < \frac{2c_0}{f(D)} \leq \frac{1}{2}$ for large enough n. Because D has left depth j, we must have $|D| \leq \left(\frac{3}{4}\right)^{j-1} \cdot |S|$. Because $|S| \leq \frac{2}{3}\sigma \leq \frac{3}{4}\sigma$, we have $|D| \leq \left(\frac{3}{4}\right)^{j} \cdot \sigma$. Therefore,

$$\begin{aligned} f(D) &= c_f \log\left(\frac{\sigma}{|D|}\right) \log\log n \geq c_f \cdot j \log\left(\frac{4}{3}\right) \log\log n = \frac{4c_0}{\log\left(\frac{4}{3}\right)} \cdot j \log\left(\frac{4}{3}\right) \log\log n \\ &= 4c_0 \cdot j \log\log n. \end{aligned}$$

Plugging in, we have

$$\begin{aligned} \Phi(j) &\leq \Phi(j+1) \cdot \left(1 + \frac{4c_0}{4c_0 \cdot j \log\log n}\right) = \Phi(j+1) \cdot \left(1 + \frac{1}{j \log\log n}\right) \\ &\leq \prod_{l=j+1}^{3\log n} \left(1 + \frac{1}{l \log\log n}\right) \cdot \left(1 + \frac{1}{j \log\log n}\right) = \prod_{l=j}^{3\log n} \left(1 + \frac{1}{l \log\log n}\right), \end{aligned}$$

which proves the claim. □

We therefore have that the maximum load on an edge (*i.e.*, a split node) in the routing scheme is $\Phi(1) \leq \prod_{l=1}^{3\log n}\left(1 + \frac{1}{l \log\log n}\right)$. Using Claim A.1 we have that $\Phi(1) = O(1)$. This means that the load on each edge is always constant, and the congestions of the scaled flows are also constant. An edge is congested by the flows when it is in the left side of the cut (*i.e.*, $G_{\tilde{L}}$), which can happen at most $O(\log n)$ times. Therefore, the total congestion is $O(\log n)$, which completes the proof of Theorem 6.2. □

References

[ADK23] Daniel Agassy, Dani Dorfman, and Haim Kaplan. Expander decomposition with fewer inter-cluster edges using a spectral cut player. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, pages 9–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023.

[ADK25] Daniel Agassy, Dani Dorfman, and Haim Kaplan. Expander decomposition for non-uniform vertex measures. *arXiv preprint arXiv:2510.23913*, 2025.

[AGG+09] Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce Maggs, and Adam Meyerson. Simultaneous source location. *ACM Transactions on Algorithms (TALG)*, 6(1):1–17, 2009.

[AR98] Yonatan Aumann and Yuval Rabani. An o (log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[BFK+14] Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph (Seffi) Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, 43(2):872–904, 2014.

[BKR03] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 24–33, 2003.

[BL97] Yair Bartal and Stefano Leonardi. On-line routing in all-optical networks. In *International Colloquium on Automata, Languages, and Programming*, pages 516–526. Springer, 1997.

[Che70] Jeff Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in analysis*, 625(195-199):110, 1970.

[CKS04] Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 156–165, 2004.

[CKS05] Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 183–192, 2005.

[EKLN07] Roee Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph Seffi Naor. Cut problems in graphs with a budget constraint. *Journal of Discrete Algorithms*, 5(2):262–279, 2007.

[GHZ21] Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1208–1220, 2021.

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings*

of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 2212–2228. SIAM, 2021.

[HHR03] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 34–43, 2003.

[HRG22] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1325–1338, 2022.

[KKM14] Rohit Khandekar, Guy Kortsarz, and Vahab Mirrokni. On the advantage of overlapping clusters for minimizing conductance. *Algorithmica*, 69(4):844–863, 2014.

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

[LL25] Jason Li and Owen Li. A simple and fast algorithm for fair cuts. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 400–411. Springer, 2025.

[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, 2023.

[LR99] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999.

[LRW25] Jason Li, Satish Rao, and Di Wang. Congestion-approximators from the bottom up. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2111–2131. SIAM, 2025.

[MadHVW97] Bruce M Maggs, F Meyer auf der Heide, Berthold Vöcking, and Matthias Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 284–293. IEEE, 1997.

[OSVV08] Lorenzo Orecchia, Leonard J. Schulman, Umesh V. Vazirani, and Nisheeth K. Vishnoi. On partitioning graphs via single commodity flows. In *Proceedings of the 40th Annual Symposium on Theory of Computing (STOC)*, pages 461–470, 2008.

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

[Räc02] Harald Räcke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.

[Räc08] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 255–264, 2008.

[Rao99] Satish Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proceedings of the fifteenth annual symposium on Computational geometry*, pages 300–306, 1999.

[RS14] Harald Räcke and Chintan Shah. Improved guarantees for tree cut sparsifiers. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22nd Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 774–785. Springer, 2014.

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 227–238, 2014.

[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.

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2635, 2019.

[VDBCK+24] Jan Van Den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-linear time algorithms for decremental graphs: Min-cost flow and more via duality. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2010–2032. IEEE, 2024.

A Omitted Proofs

A.1 Omitted Proofs from Section 3

Proof of Fact 3.19. Denote $P' = P - P^{\uparrow Q}$. We begin by proving that P' is a valid demand state. Let $k \in \mathcal{K}$. Observe that:

$$\sum_{u \in V} P'(u,k) = -\sum_{u \in V}\sum_{v \in V} \frac{P(u,k)}{\|P(u)\|_1} D_Q(u,v) + \sum_{u \in V}\sum_{v \in V} \frac{P(v,k)}{\|P(v)\|_1} D_Q(v,u) = 0.$$

We now prove that $\text{dem}_{P'}(B,W) \leq \text{dem}_Q(B,W)$, for every cut (B,W). Fix a cut (B,W), we get that:

$$\begin{aligned}
\text{dem}_{P'}(B,W) &= \sum_{k\in\mathcal{K}} \left| \sum_{u\in B} P'(u,k) \right| \\
&= \sum_{k\in\mathcal{K}} \left| \sum_{u\in B}\sum_{v\in V} \frac{P(v,k)}{\|P(v)\|_1} D_Q(v,u) - \sum_{u\in B}\sum_{v\in V} \frac{P(u,k)}{\|P(u)\|_1} D_Q(u,v) \right| \\
&= \sum_{k\in\mathcal{K}} \left| \sum_{u\in B}\sum_{v\in B} \frac{P(v,k)}{\|P(v)\|_1} D_Q(v,u) + \sum_{u\in B}\sum_{v\in W} \frac{P(v,k)}{\|P(v)\|_1} D_Q(v,u) \right. \\
&\qquad \left. - \sum_{u\in B}\sum_{v\in B} \frac{P(u,k)}{\|P(u)\|_1} D_Q(u,v) - \sum_{u\in B}\sum_{v\in W} \frac{P(u,k)}{\|P(u)\|_1} D_Q(u,v) \right| \\
&= \sum_{k\in\mathcal{K}} \left| \sum_{u\in B}\sum_{v\in W} \frac{P(v,k)}{\|P(u)\|_1} D_Q(v,u) - \sum_{u\in B}\sum_{v\in W} \frac{P(u,k)}{\|P(u)\|_1} D_Q(u,v) \right| \\
&\overset{(1)}{\leq} \sum_{u\in B}\sum_{v\in W}\sum_{k\in\mathcal{K}} \frac{|P(v,k)|}{\|P(v)\|_1} D_Q(v,u) + \frac{|P(u,k)|}{\|P(u)\|_1} D_Q(u,v) \\
&= \sum_{u\in B}\sum_{v\in W}\sum_{k\in\mathcal{K}} D_Q(v,u) + D_Q(u,v) = \text{dem}_Q(B,W),
\end{aligned}$$

where Inequality (1) is due to triangle inequality and change of order of summation. □

A.2 Omitted Proofs from Section 5

Proof of Claim 5.8. Recall Figure 3. Intuitively, when we spread the demands of $P(v)$ on the neighboring split nodes of v, we do not change the demand across the (B',W') cut, except in the case that this spreading moved mass from one side of the cut to the other. In other words, this happens only if the edge (x_e,v) crosses the cut. This may occur only if $e=(u,v)\in E(B,W)$, and $v\in W$. In this case, the amount of mass that crosses from v to x_e is at most 1. Summing across all edges in $E(B,W)$, we only change the demand across the cut by at most $\text{cap}(B,W)$.

The formal proof is as follows. Recall that $\left\|P_{\{v\}'}(x_e)\right\|_1 \leq 1$ for every $v\in V$ and $e\in E$ adjacent

to v. We get that

$$\begin{aligned}
\mathrm{dem}_{P_{\mathcal{Q}_0}}(B', W') &= \sum_{k\in\mathcal{K}} \left| \sum_{u\in B'} P_{\mathcal{Q}_0}(u,k) \right| = \sum_{k\in\mathcal{K}} \left| \sum_{u\in B'} \sum_{v\in V} P_{\{v\}'}(u,k) \right| \\
&\overset{(1)}{\geq} \sum_{k\in\mathcal{K}} \left| \sum_{v\in B} \sum_{u\in B'} P_{\{v\}'}(u,k) \right| - \sum_{k\in\mathcal{K}} \left| \sum_{v\in W} \sum_{u\in B'} P_{\{v\}'}(u,k) \right| \\
&\overset{(2)}{\geq} \sum_{k\in\mathcal{K}} \left| \sum_{v\in B} P(v,k) \right| - \sum_{k\in\mathcal{K}} \sum_{v\in W} \sum_{u\in B'} \left| P_{\{v\}'}(u,k) \right| \\
&= \mathrm{dem}_P(B,W) - \sum_{v\in W} \sum_{k\in\mathcal{K}} \sum_{u\in B'\cap\{v\}'} \left| P_{\{v\}'}(u,k) \right| \\
&= \mathrm{dem}_P(B,W) - \sum_{v\in W} \sum_{u\in B'\cap\{v\}'} \sum_{k\in\mathcal{K}} \left| P_{\{v\}'}(u,k) \right| \\
&\overset{(3)}{\geq} \mathrm{dem}_P(B,W) - \sum_{v\in W} \sum_{u\in B'\cap\{v\}'} 1 \\
&\overset{(4)}{=} \mathrm{dem}_P(B,W) - \mathrm{cap}_G(B,W),
\end{aligned}$$

where Inequality (1) holds by swapping the order of summation and by triangle inequality; Inequality (2) holds by the way we defined $P_{\{v\}'}$ (spreading out the demands of $P(v)$ on the adjacent split nodes), as well as the way we defined B' (that is, B' contains all the split nodes defined by the cut (B,W) in G) and triangle inequality; Inequality (3) holds by the mentioned observation that $\left\|P_{\{v\}'}(x_e)\right\|_1 \leq 1$, and Equality (4) holds by the way we defined B'.

□

A.3 Algebraic Tools

Claim A.1 (See [HHR03, Lemma 10])**.** *For $k \geq 3$ and $c > 0$,*

$$\prod_{\ell=1}^{k} \left(1 + \frac{c}{\ell}\right) \leq k^{2c}.$$

Proof. We have

$$\prod_{\ell=1}^{k} \left(1 + \frac{c}{\ell}\right) = \exp\left(\sum_{\ell=1}^{k} \ln\left(1 + \frac{c}{\ell}\right)\right) \leq \exp\left(\sum_{\ell=1}^{k} \frac{c}{\ell}\right) = \exp\big(cH_k\big),$$

where we used $\ln(1+x) \leq x$ for $x > -1$, and $H_k = \sum_{\ell=1}^{k} \frac{1}{\ell}$ denotes the k-th harmonic number. Since $H_k \leq 2\ln k$,

$$\exp(cH_k) \leq \exp\big(c \cdot 2\ln k\big) = k^{2c}.$$

□