

An Improved Quality Hierarchical Congestion Approximator in Near-Linear Time

Monika Henzinger

Institute of Science and Technology
Austria
Klosterneuburg, Austria
monika.henzinger@ist.ac.at

Robin Münk

Technical University of Munich
Munich, Germany
robin.muenk@tum.de

Harald Räcke

Technical University of Munich
Munich, Germany
raecke@in.tum.de

Abstract

A single-commodity congestion approximator for a graph is a compact data structure that approximately predicts the edge congestion required to route any set of single-commodity flow demands in a network. A hierarchical congestion approximator (HCA) consists of a laminar family of cuts in the graph and has numerous applications in approximating cut and flow problems in graphs, designing efficient routing schemes, and managing distributed networks.

There is a tradeoff between the running time for computing an HCA and its approximation quality. The best polynomial-time construction in an n -node graph gives an HCA with approximation quality $O(\log^{1.5} n \log \log n)$. Among near-linear time algorithms, the best previous result achieves approximation quality $O(\log^4 n)$. We improve upon the latter result by giving the first near-linear time algorithm for computing an HCA with approximation quality $O(\log^2 n \log \log n)$. Additionally, our algorithm can be implemented in the parallel setting with polylogarithmic span and near-linear work, achieving the same approximation quality. This improves upon the best previous such algorithm, which has an $O(\log^9 n)$ approximation quality. We also present a lower bound of $\Omega(\log n)$ for the approximation guarantee of hierarchical congestion approximators.

Crucial for achieving a near-linear running time is a new partitioning routine that, unlike previous such routines, manages to avoid recursing on large subgraphs. To achieve the improved approximation quality, we introduce the new concept of border routability of a cut and provide an improved sparsest cut oracle for general vertex weights.

CCS Concepts

• Theory of computation → Graph algorithms analysis.

Keywords

Congestion Approximators, Hierarchical Graph Decompositions

ACM Reference Format:

Monika Henzinger, Robin Münk, and Harald Räcke. 2026. An Improved Quality Hierarchical Congestion Approximator in Near-Linear Time. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing (STOC '26)*, June 22–26, 2026, Salt Lake City, UT, USA. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3798129.3800851>



This work is licensed under a Creative Commons Attribution 4.0 International License. STOC '26, Salt Lake City, UT, USA

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2536-4/2026/06
<https://doi.org/10.1145/3798129.3800851>

1 Introduction

A congestion approximator for a graph is a compact data structure that approximately predicts the edge congestion required to route a given set of flow demands in a network.

Typically, it consists of a collection of cuts. Each cut provides a lower bound on the congestion needed to realize a particular demand: namely, the total demand crossing the cut divided by the cut's capacity. The prediction of the approximator for a given demand vector is then the maximum of these lower bounds over all cuts in the collection.

By the classical max-flow–min-cut theorem [10], an approximator that includes all 2^n possible cuts would yield exact predictions for single-commodity flows. Surprisingly, Räcke [23] showed that already a *linear number of cuts* suffice to approximate the required congestion for all demands within a factor of $O(\log^3 n)$. Importantly, this guarantee extends to multicommodity flow demands, demonstrating that the flow-cut structure of an undirected graph can be captured efficiently using only a small collection of cuts.

Räcke's original result was non-constructive, i.e., it established the existence of a congestion approximator but did not provide an efficient way to compute it. Independent of each other, Bienkowski, Korzeniowski, and Räcke [7] and Harrelson, Hildrum, and Rao [13] provided the first polynomial-time constructions of congestion approximators. Their algorithms achieved approximation guarantees of $O(\log^4 n)$ and $O(\log^2 n \log \log n)$, respectively. Subsequently, Räcke and Shah [25] constructed a congestion approximator for single-commodity flows, obtaining an existential guarantee of $O(\log n \log \log n)$ and a polynomial-time construction with guarantee $O(\log^{1.5} n \log \log n)$.

A key feature of all the above constructions is that the collection of cuts forms a *laminar family*. We refer to a congestion approximator with this property as a *hierarchical congestion approximator*. Intuitively, this means that the cut structure of the graph is approximated by a hierarchical decomposition that can be represented by a single tree, a property that plays a central role in many applications, including oblivious routing, approximation algorithms for sparsest cut and multicommodity flow, network design problems, and routing schemes in distributed networks [5, 6, 8, 9, 15, 18].

As many of these applications require working with large graphs, the mere polynomial-time guarantees of previous constructions posed a significant obstacle. Räcke, Shah, and Täubig [26] addressed this challenge by giving an almost-linear-time algorithm that produces a hierarchical congestion approximator with an approximation guarantee of $O(\log^4 n)$ for multicommodity flows. Interestingly, this result relied on a technique of Sherman [28] to compute approximate maximum single-commodity flows in almost-linear time,

given access to a congestion approximator. The quality of this approximator directly affected the running time of his method. Thus, improving Sherman's algorithm to near-linear time would have improved the RST14 result to near-linear time, and vice versa. This circular dependency was ultimately resolved by Peng [22], who showed that both problems can be solved in near-linear time.

More recently, Li, Rao, and Wang [19] proposed a bottom-up approach to constructing congestion approximators, which contrasts structurally with the recursive, top-down techniques used in earlier work. Unlike Peng's method, their algorithm does not rely on heavy recursion, leading to smaller logarithmic factors in the running time, albeit with a weaker approximation guarantee of $O(\log^{10} n)$.

In this paper, we substantially improve the approximation guarantee for near-linear-time constructions of hierarchical congestion approximators. We show how to compute a congestion approximator for single-commodity flows with approximation guarantee $O(\log^2 n \log \log n)$ in nearly-linear time. This improves upon RST14 by almost two logarithmic factors and nearly matches the best known guarantee of $O(\log^{1.5} n \log \log n)$ for single-commodity congestion approximators, which takes time $\tilde{O}(n^2)$. Formally, we show the following theorem.

THEOREM 1. *Given an undirected graph with n vertices, there is a near-linear time algorithm that computes a hierarchical congestion approximator that w.h.p. has an approximation guarantee of $O(\log^2 n \log \log n)$ for single-commodity flows.*

Note that a single-commodity hierarchical congestion approximator is equivalent to a *tree cut sparsifier*. Formally, a tree cut sparsifier T of a graph $G = (V, E)$ with quality $q \geq 1$ is a weighted tree whose leaves correspond to the vertices of G and for any pair A, B of disjoint subsets of V , $1/q \cdot \text{mincut}_T(A, B) \geq \text{mincut}_G(A, B) \geq \text{mincut}_T(A, B)$, where, for any graph H , $\text{mincut}_H(A, B)$ denotes the value of a minimum cut separating A from B in H . The stronger notion of *tree flow sparsifier* is equivalent to a hierarchical congestion approximator for multi-commodity flows. For completeness, we give a lower bound of $\Omega(\log n)$ for the approximation quality of any hierarchical congestion approximator in the full version [14] based on a reduction from oblivious routing.

By the definition of the flow-cut gap λ for concurrent multi-commodity flows¹, a result with approximation guarantee α for single-commodity flows implies a result for multi-commodity flow with approximation guarantee $\lambda\alpha$. It is known that $\lambda = O(\log n)$ in any graph, while some graph classes have a smaller gap. This leads to the following corollary.

Corollary 1. *Given an undirected graph with n vertices, there is a near-linear time algorithm that computes a hierarchical congestion approximator that w.h.p. has approximation guarantee $O(\log^3 n \log \log n)$ for multi-commodity flows.*

¹Define for a multi-commodity flow demand d and a cut $C \subseteq V$, $d(C, V \setminus C) = \sum_{(x,y) \in C \times V \setminus C} d(x, y)$, and $\Phi(d) := \max_S d(S, V \setminus S) / \text{cap}(S, V \setminus S)$. Then the flow-cut gap of G w.r.t. d is the ratio $\text{opt}_G(d) / \Phi(d)$, where $\text{opt}_G(d)$ is the optimum congestion for routing d in G . The *flow-cut gap* λ of G is the maximum of this ratio over all demands d . If the maximum is just taken over product multicommodity flow demands, then the corresponding flow-cut gap is denoted by λ^* .

²see full version of this paper [14]

Table 1: Comparison of hierarchical congestion approximator results. A result with guarantee α for single-commodity flows implies a result for multi-commodity flow with guarantee $\lambda\alpha$, where $\lambda = O(\log n)$ is the *flow-cut gap* of the graph. For the marked results (*) this step loses only a factor of λ^* where λ^* is the flow-cut gap w.r.t. product multicommodity flows. The latter e.g. is constant for graphs excluding a fixed minor [17].

Approximation	Running Time	Type	Work
$O(\log^3 n)$	—	multi	[23]
$O(\log^4 n)$	$\text{poly}(n)$	multi	[7]
$O(\log^2 n \log \log n)$	$\text{poly}(n)$	multi	[13]
$O(\log n \log \log n)$	—	single*	[25]
$O(\log^{1.5} n \log \log n)$	$\text{poly}(n)$	single*	[25]
$O(\log^4 n)$	$O(m^{1+o(1)})$	multi, whp	[26]
$O(\log^4 n)$	$\tilde{O}(m)$	multi, whp	[22]
$O(\log^{10} n)$	$\tilde{O}(m)$	single, whp	[19]
$O(n^{o(1)})$	updates $n^{o(1)}$	multi, whp	[11]
$O(\log^2 n \log \log n)$	$\tilde{O}(m)$	single*, whp	this paper
$O(\log^9 n)$	$\tilde{O}(m)$ work $\tilde{O}(\log^c n)$ span	multi, whp	[1]
$O(\log^2 n \log \log n)$	$\tilde{O}(m)$ work $\tilde{O}(\log^c n)$ span	single*, whp	this paper
$\Omega(\log n)$	—	single	this paper ²

Note that all previous near-linear time constructions for congestion approximators that obtain a polylogarithmic guarantee only give a correct solution with high probability.

An important feature of our algorithm is that it can be easily parallelized. The details of the parallel construction can be found in the full version. The previous best parallel algorithm for congestion approximators is due to Agarwal et al. [1], who obtain an approximation guarantee of $O(\log^9 n)$ with $O(m \text{polylog } n)$ work and $O(\text{polylog } n)$ span.

THEOREM 2. *Given an undirected graph with m edges and n vertices, there is a parallel algorithm that requires $O(m \text{polylog } n)$ work, has $O(\text{polylog } n)$ span, and constructs a hierarchical congestion approximator that has approximation guarantee $O(\log^2 n \log \log n)$, w.h.p.*

Independent Work. Recently, in an independent result, Agassy, Dorfman and Kaplan published a paper on arXiv [4] that also gives a near-linear time algorithm for computing a hierarchical congestion approximator of the same quality as our algorithm. They similarly utilize a generalized sparsest cut oracle for general vertex weights, which they present in another paper that was recently published on arXiv [3]. They do however obtain their tree flow sparsifier using different techniques and offer no guarantees for the parallel setting.

Our generalized sparsest cut oracle has a running time that is linear in the number of (parallel) edges in an unweighted graph in our application. The algorithm of [3] takes time linear in the number of edges in a capacitated graph (not the sum of the edge

weights). Using this subroutine instead, we obtain the following corollary for capacitated graphs, as the other parts of our algorithm work when edges have weights.

Corollary 2. Given an undirected, capacitated graph with n vertices, there is an algorithm that takes near-linear time (in the number of edges) and computes a hierarchical congestion approximator that w.h.p. has approximation guarantee $O(\log^2 n \log \log n)$ for single-commodity flows.

1.1 Further Related Work

Apart from the work on hierarchical congestion approximators there is also the work on congestion approximators where the family of cuts does not form a laminar family. Räcke [24] introduced the concept of approximating the cut-structure of a graph not by a single tree as in a hierarchical congestion approximator but by a probability distribution over trees. These trees can be constructed in polynomial time and the collection of all cuts over all trees can be interpreted as a congestion approximator. This gives a congestion approximator with approximation guarantee of $O(\log n)$.

Madry [20] approximated the graph instead by a probability distribution over j -trees, which are trees with a small number of additional edges. His construction has a worse approximation guarantee of $O(n^{o(1)})$ but allows to sample a j -tree from the distribution in almost linear time. This gave almost linear time approximation algorithms for many applications before RST14 gave the first congestion approximator in nearly linear time. This technique is also very important in the area of dynamic graph algorithms.

For many applications approximating the graph by a probability distribution of trees as in [24] or by a distribution over j -trees is sufficient. Some applications do however require a single tree [6, 15].

The first efficient implementation of a congestion approximator for practical applications was recently achieved using a random-walk-based subroutine for finding sparse cuts [12].

Connection to Expander Decompositions. An *expander decomposition* of an undirected graph $G = (V, E)$ is a partition of its vertex set into disjoint pieces $V_1, \dots, V_k$ such that each piece induces a subgraph $G[V_i]$ with conductance at least ϕ . The goal is to minimize the number of edges that go between different pieces (we say such an edge is *cut*).

It is known that any expander decomposition may need to cut $\phi m \log n$ edges in the worst case. In the following, we say that an expander decomposition has *gap* g if it cuts at most $g\phi m$ edges. There are close connections between the problem of finding good hierarchical congestion approximators and that of computing expander decompositions with small gap.

In one direction, Goranci et al. [11] showed that an algorithm for expander decomposition can be used to compute a hierarchical congestion approximator with quality $n^{o(1)}$ in almost-linear time. This result allowed them to obtain dynamic congestion approximators with the same $n^{o(1)}$ quality.

Another relationship between near linear time constructions for congestion approximators and expander decompositions is in terms of techniques. Efficient constructions for both problems usually rely on variants of the Cut-Matching game as introduced by [16]. We generalize a Cut-Matching game that was presented by Agassy et al. [2] with the application of computing an expander

decomposition. They obtained the currently best gap of $O(\log^2 n)$ for this problem.

Due to limited space, we defer a thorough technical overview and some smaller proofs to the full version of this paper [14].

2 Basic Notation and Definitions

Graphs. We are given an undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$ and parallel edges allowed. Without loss of generality we assume throughout the paper that the graph is connected. Given an edge (u, v) , $\text{cap}(u, v)$ denotes the number of parallel edges between u and v . For $S \subseteq V$ we use $G[S]$ to denote the subgraph induced by the vertices in S . A *cut* usually refers to an edge set $E(S, V \setminus S)$ for a vertex set $S \subset V$, but, by abuse of notation, we sometimes also refer to the set S itself as a cut. For $F \subseteq E$ we use $\deg_F(v)$ to denote the sum of the capacities of the edges of F that are incident to v . If $F = E$, then we use $\deg(v)$ instead of $\deg_E(v)$.

For a set $C \subseteq V$, let the *boundary* or *border* ∂C denote the set of edges with exactly one endpoint in C and for any set $F \subseteq E$, and $v \in V$ let $\deg_F(v)$ be the capacity of the edges of F that are incident to v . For a collection of (not necessarily disjoint) subsets $C = \{C_1, \dots, C_k\}$ we use ∂C as a shorthand for $\bigcup_{C \in \mathcal{C}} \partial C$.

Partition. A *partition* $\mathcal{X}$ of a subset S of V is a family of disjoint non-empty subsets $X_1, \dots, X_k$ such that $\bigcup_k X_i = S$. Note that $\mathcal{X} = \{S\}$ means that the partition contains exactly one set, namely S , i.e., it is a trivial partition.

For a partition $\mathcal{X}$ we use $\partial \mathcal{X}$ to denote the set of edges such that for each edge exactly one endpoint belongs to one set A of $\mathcal{X}$ and the other endpoint does not belong to A . Given two partitions $\mathcal{X}$ and $\mathcal{X}'$ of two disjoint vertex sets X and X' we use $\mathcal{X} \uplus \mathcal{X}'$ to denote the set-union of the family of subsets $\mathcal{X}$ and with the family of subsets of $\mathcal{X}'$, i.e. $\mathcal{X} \uplus \mathcal{X}' = \{C \mid C \in \mathcal{X} \text{ or } C \in \mathcal{X}'\}$.

Vertex weight functions. We call a function $x : V \rightarrow \mathbb{R}$ a (*vertex*) *weight function* for the vertices of V . Let $S \subseteq V$. Then $x(S) = \sum_{v \in S} x(v)$. For example, $\deg_F(S) = \sum_{v \in S} \deg_F(v)$.

We use $x|_S : V \rightarrow \mathbb{R}$ to denote the function where $x|_S(v) = x(v)$ for $v \in S$ and $x|_S(v) = 0$ for $v \notin S$. Given two weight functions x and x' we use $x \leq x'$ when $x(v) \leq x'(v)$ for all $v \in V$ and use $|x|$ to be the function where $|x|(v) = |x(v)|$ for all $v \in V$.

Flows. A *demand* or *demand function* is a vertex weight function d such that $\sum_{v \in V} d(v) = 0$. A flow $f : V \times V \rightarrow \mathbb{R}$ satisfies $f(u, v) = -f(v, u)$ for all $u, v \in V$ and $f(u, v) = 0$ for all pairs $(u, v) \notin E$. A flow is *feasible* if $\text{cap}(u, v) \geq |f(u, v)|$ for all $(u, v) \in E$. If $f(u, v) > 0$ then we say that *flow mass is routed from u to v* . For each vertex $u \in V$ the *net-flow out of u* is $f(u) = \sum_{v \in V} f(u, v)$, i.e., the total flow mass routed from u to its neighbors in G minus the total flow mass routed from its neighbors to u .

A flow *routes a demand d* if the net-flow out of each vertex v is $d(v) \in \mathbb{R}$ (which might be negative). For $S \subseteq V$, a flow within an induced sub-graph $G[S]$ routes $d : V \rightarrow \mathbb{R}$ if the net-flow out of each vertex v is $d(v)$ and $d(v) = 0$ for all $v \notin S$.

Given two non-negative vertex weight functions $s, t : V \rightarrow \mathbb{R}$ with $\sum_v s(v) = \sum_v t(v)$, we say that a flow f is an (s, t) -flow if it routes the demand $s - t$. If $|f(u, v)| \leq c \cdot \text{cap}(u, v)$ for all $u, v \in V$, then f has *congestion c* . If $c = 1$, then f is feasible.

Congestion Approximator. Let $\alpha \geq 1$. An α -congestion approximator C for a graph $G = (V, E)$ with edge capacities cap is a collection of subsets of V such that for any demand d satisfying for all $C \in C$, $|d(C)| \leq \deg_{\partial C}(C)$, it is guaranteed that there exists a flow in G routing d with congestion α . Thus, for an α -congestion approximator C , the existence of a flow routing d can be checked simply by checking this condition for all $C \in C$. We refer to α as the *approximation guarantee* of the congestion approximator. Note that if the condition is violated for at least one set $C \in C$ then we know that no such flow with congestion 1 exists, but we do not know that no flow with congestion $\alpha > 1$ exists.

Expansion. We call $G = (V, E)$ π -expanding with quality $q > 0$, if every set $X \subseteq V$ with $\pi(X) \leq \pi(V \setminus X)$ fulfills $\text{cap}(X, V \setminus X) \geq q \cdot \pi(X)$. If this holds for $q = 1$, we just call the graph π -expanding. Note that the larger q , the stricter the requirement. For $S \subset V$, we say that $G[S]$ is π -expanding with quality q if it is $\pi|_S$ -expanding with quality q . If, however, a cut X with $\pi(X) \leq \pi(V \setminus X)$ exists such that $\text{cap}(X, V \setminus X) < q \cdot \pi(X)$, we say that X is $q \cdot \pi$ -sparse. There is an equivalent characterization of expansion using flows. The following lemma is due to a simple application of the max-flow min-cut theorem.

LEMMA 1. A graph $G = (V, E)$ is π -expanding with quality q iff every demand function d , with $|d| \leq \pi$ can be routed with congestion at most $1/q$.

Hierarchical Decomposition. A (partial) hierarchical decomposition of G is a sequence $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_L)$ of distinct partitions s.t.

- $\mathcal{P}_1$ is the partition $\{V\}$ with a single cluster, and
- for every $i > 1$, $\mathcal{P}_i$ is a refinement of $\mathcal{P}_{i-1}$, i.e., for $X \in \mathcal{P}_i$ there exists a cluster $P \in \mathcal{P}_{i-1}$ with $X \subseteq P$. We call P the *parent cluster* of X and denote it by $\text{par}_{\mathcal{P}}(X)$.

We refer to the unique cluster $V \in \mathcal{P}_1$ as the *root-cluster* of $\mathcal{P}$ and use the *height* L to denote the number of partitions in the hierarchical decomposition. We define $\text{par}_{\mathcal{P}}(V) = V$. If $\mathcal{P}_L = \{\{v\} \mid v \in V\}$ is the partition into singletons, we call $\mathcal{P}$ *complete*.

A hierarchical decomposition can be represented in a natural way by a tree structure $\mathcal{T}$ where each cluster X in a partition is represented by a node in the tree (and by abuse of notation we also call that node X) and X is a child of $\text{par}(X)$. Thus all nodes that belong to the same partition are on the same level of $\mathcal{T}$.

κ -Border Routability. While constructing the hierarchy we will use the concept of border routability. Given a set $C \subseteq V$ and a subset $U \subseteq C$, we say that a cut $E(U, C \setminus U)$ is κ -border-routable through U , if for any non-negative weight function s with $|s| \leq \deg_{E(U, C \setminus U)}$ there exists a non-negative vertex weight t with $|t| \leq \deg_{E(U, V \setminus C)}/\kappa$ and an (s, t) -flow in $G[U]$ with congestion 2.

3 Building Blocks

We next present a list of algorithmic problems and solutions that are used as subroutines by our algorithm in order to build the desired hierarchical decomposition. As these subroutines are crucial pieces for our algorithm we call them *building blocks*.

Definition 1 (Fair Cut/Flow Pair). Let $G = (V, E)$ be a graph with integral edge capacities cap and let s and t be two non-negative

vertex weight functions. For any parameter $\alpha \geq 1$, we say that a set $U \subseteq V$ and a feasible flow f is an α -fair (s, t) -cut/flow pair (U, f) if

- (1) net sources do not send too much:
all $v \in V$ with $s(v) - t(v) \geq 0$ have $0 \leq f(v) \leq s(v) - t(v)$
- (2) net targets do not absorb too much:
all $v \in V$ with $s(v) - t(v) \leq 0$ have $0 \geq f(v) \geq s(v) - t(v)$
- (3) net sources in $V \setminus U$ are nearly saturated:
all $v \in V \setminus U$ with $s(v) - t(v) \geq 0$ have $f(v) \geq (s(v) - t(v))/\alpha$
- (4) net targets in U are nearly saturated:
all $v \in U$ with $s(v) - t(v) \leq 0$ have $f(v) \leq (s(v) - t(v))/\alpha$
- (5) edges from U to $V \setminus U$ are nearly saturated:
each edge $\{u, v\} \in E(U, V \setminus U)$ with $u \in U$ and $v \in V \setminus U$ sends at least $\text{cap}(u, v)/\alpha$ flow in the direction from u to v . In particular, these edges send no flow in the reverse direction.

An α -fair (s, t) -cut U in V is a vertex set for which a flow f exists such that (U, f) is an α -fair (s, t) -cut/flow pair in V .

Note that this definition is not symmetric in the sense that the fact that (U, f) is an α -fair (s, t) -cut/flow pair in V does not imply that this also holds for $(V \setminus U, f)$. It does, however, imply that $(V \setminus U, -f)$ is an α -fair (t, s) -cut/flow pair in V . Our first building block computes an α -fair (s, t) -cut/flow pair in V .

Building Block (FAIRCUTFLOW(G, s, t, α)). Given a graph $G = (V, E)$ with integral edge capacities cap , two non-negative vertex weight functions s, t , and a parameter $\alpha > 1$, the algorithm FAIRCUTFLOW outputs an α -fair (s, t) -cut/flow pair (U, f) .

For many applications of FAIRCUTFLOW, the flow is not needed explicitly and just the cut would be sufficient alongside the promise of *existence* of a flow. However, the implementation we use (Theorem 3) computes both simultaneously, so we define it here to output both, as there is no additional overhead for providing an explicit flow. In the full version of this paper we show how to obtain the following lemma with a simple reduction to known results.

THEOREM 3. For a graph $G = (V, E)$ with $m = \deg(V)$ and $\alpha > 1$, an algorithm for FAIRCUTFLOW exists with running time $T_{\text{fc}}(m, \alpha) = \tilde{O}(m/(\alpha - 1))$.

Our next building block computes an approximately sparsest cut and gives an additional guarantee: either the cut is relatively balanced or the larger side has a certain expansion. This extra property is crucial in helping to ensure that our partitioning routine always makes sufficient progress. A similar subroutine was first used to compute expander decompositions in near-linear time [27].

Its output guarantees use parameters $q^* \geq 1$ and $\beta^* \in (0, 1/2]$ to parametrize the achieved quality of expansion and balance guarantee, respectively.

Building Block (SPARSESTCUTAPX(G, π, ϕ)). Given a graph $G = (V, E)$, an integral, non-negative vertex weight function π and a sparsity parameter $\phi \in (0, 1)$, the algorithm SPARSESTCUTAPX computes a (potentially empty) set $R \subseteq V$, with $\pi(R) \leq \pi(V \setminus R)$ such that

- (1) R is ϕ -sparse w.r.t. π , i.e., $\text{cap}(R, V \setminus R) \leq \phi \pi(R)$; and
- (2) if R is very imbalanced, i.e., $\pi(R) < \beta^* \pi(V)$, then G is $(\phi/q^*) \cdot \pi|_{V \setminus R}$ expanding with high probability.

In the full version of this paper we show our implementation of **SPARSESTCUTAPX** and prove the following theorem. The algorithm is based on a generalized adaptation of the cut-matching game [2, 16, 21, 26], allowing for weights on the vertices, as is needed here.

THEOREM 4. *Given a graph $G = (V, E)$ with $n = |V|$ and $m = \deg(V)$ and an algorithm for **FAIRCUTFLOW** that runs in time T_{fc} , we can implement **SPARSESTCUTAPX** with parameters $q^* = O(\log \pi(V))$ and $\beta^* = 1/(2 \log \pi(V))$ in running time $T_{sc} = O(\log^4(\pi(V)) \cdot (T_{fc}(m, 3/2) + \pi(V) \log(\pi(V)) + m \log n))$.*

While the theorem is more general, for all our purposes, we will have $\pi(V) = O(m)$. Hence, throughout the paper we use the resulting values $q^* = O(\log n)$ and $\beta^* = \Omega(1/\log n)$ as the quality and balance guarantee of our implementation of **SPARSESTCUTAPX**. For technical reasons, we will further assume that $q^* \geq (\log n)/125$. We also use the value $\tau^* := \min\{1/440q^*, \beta^*\}$. In our implementation, we thus have $\tau^* = \Omega(1/\log n)$. Together with our implementation of **FAIRCUTFLOW** (Theorem 3), we get the following corollary.

Corollary 3. *If $\pi(V) = O(m)$, then **SPARSESTCUTAPX** can be implemented with $q^* = O(\log n)$ and $\beta^* = \Omega(1/\log n)$ in time $O(\log^5 n \cdot (T_{fc}(m, 3/2)) + m) = \tilde{O}(m)$.*

4 Congestion Approximator

Our goal is to construct a hierarchical decomposition that fulfills certain properties, for which we first show that they naturally give a congestion approximator. More specifically, our goal is to design a hierarchical decomposition in which every cluster fulfills an expansion property w.r.t. its partitioning into sub-clusters. Formally, given a hierarchical decomposition $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_L)$ we define the function $f_{\mathcal{P}}$ for a level- i cluster X in the hierarchy as

$$f_{\mathcal{P}}(X) := \begin{cases} 1 & i = 1 \\ 3 \log \log n \log_2(2|\text{par}_{\mathcal{P}}(X)|/|X|) & \text{otherwise} \end{cases}$$

that defines an *expansion bound* for every cluster. We say that a non-leaf cluster $X \in \mathcal{P}_i$, $i < L$ is γ -well expanding for some $1 \geq \gamma > 0$ if $G[X]$ is at least $\deg_{\partial \mathcal{P}_{i+1}}$ -expanding with quality $\gamma/f_{\mathcal{P}}(X)$. Our hierarchy construction algorithm **CONSTRUCTHIERARCHY** will build a hierarchy so that every non-leaf cluster is γ -well expanding with $\gamma = \Theta(1/\log n)$.

Our goal is to find a complete hierarchical decomposition $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_L)$ with $L = O(\log n)$ such that each non-leaf cluster is γ -well expanding. The next theorem states that the set of clusters in such a decomposition is a “high-quality” congestion approximator.

THEOREM 5. *A complete hierarchical decomposition of logarithmic height in which each non-leaf cluster is γ -well-expanding gives a congestion approximator with approximation guarantee $6 \log \log n(L + \log n)/\gamma = O(\log^2 n \log \log n)$.*

PROOF. Throughout this proof we use $X_v^{(i)} \in \mathcal{P}_i$ to denote the level- i cluster that contains v within the partition $\mathcal{P}_i$, $i \geq 1$ from the hierarchy.

Suppose we are given a demand vector $d : V \rightarrow \mathbb{R}$ such that $|d(X)| \leq \deg_{\partial \mathcal{P}_i}(X)$ holds for every $X \in \mathcal{P}_i$ and $i \in \{1, \dots, L\}$. We have to show that we can route d in G with congestion at most $6 \log \log n(L + \log n)/\gamma$. This shows that $\mathcal{P}$ is a $6 \log \log n(L + \log n)/\gamma$ -congestion approximator.

We next show how to route d . The basic idea is to route d level by level. For this, we define a suitable demand d_i for each level i .

For the first level, we define $d_1(v) = 0$ for every $v \in V$. For $i > 1$ we define the *level- i demand* $d_i(v)$ of a vertex v as $d_i(v) = (\deg_{\partial \mathcal{P}_i}(v)/\deg_{\partial \mathcal{P}_i}(X_v^{(i)})) \cdot d(X_v^{(i)})$. Recall that wlog the graph is connected and, thus, $\partial \mathcal{P}_i(X) \neq 0$ for each $X \in \mathcal{P}_i$. Further observe that $d_i(X) = \sum_{v \in X} d_i(v) = \sum_{v \in X} \frac{\deg_{\partial \mathcal{P}_i}(v)}{\deg_{\partial \mathcal{P}_i}(X)} d(X) = d(X)$ for each $X \in \mathcal{P}_i$. From this we get $\sum_v d_i(v) = \sum_{X \in \mathcal{P}_i} d_i(X) = \sum_{X \in \mathcal{P}_i} d(X) = \sum_v d(v) = 0$, i.e., d_i is a proper demand vector.

Note that if the hierarchical decomposition is complete, i.e., $\mathcal{P}_L$ consists of only singleton clusters, then the definition of d_i implies that $d_L = d$. Next we assume we have for $i \in \{1, \dots, L-1\}$ a flow that routes demand $d_{i+1} - d_i$. Summing all these flows gives a flow that routes demand $d_L - d_1 = d$. Thus, we are left with giving a flow that routes the demand $d_{i+1} - d_i$.

Again, we partition the demand into several sub-demands, one for each cluster $X \in \mathcal{P}_i$. Formally, the *subflow for cluster $X \in \mathcal{P}_i$* routes demand vector $d_{i+1}|_X - d_i|_X$ inside $G[X]$. Note that $d_{i+1}|_X - d_i|_X$ is indeed a proper demand vector for $G[X]$ as $\mathcal{P}_{i+1}$ is a refinement of $\mathcal{P}_i$ and, thus, $d_{i+1}(X) = \sum_{Z \in \mathcal{P}_{i+1}, Z \cap X \neq \emptyset} d_{i+1}(Z) = \sum_{Z \in \mathcal{P}_{i+1}, Z \cap X \neq \emptyset} d(Z) = d(X) = d_i(X)$ holds for any cluster $X \in \mathcal{P}_i$ which implies $\sum_{v \in X} (d_{i+1}|_X - d_i|_X)(v) = 0$.

Let $S \subseteq X$ be any subset and assume wlog. that $\deg_{\partial \mathcal{P}_{i+1}}(S) \leq \deg_{\partial \mathcal{P}_{i+1}}(X \setminus S)$ inside $G[X]$. We show next that the total demand that has to cross S in $G[X]$ is at most $2f_{\mathcal{P}}(X) \text{cap}(S, X \setminus S)/\gamma$. By the maximum flow-minimum cut theorem [10] it follows that there exists a (single-commodity) flow that routes $d_{i+1}|_X - d_i|_X$ inside $G[X]$ with congestion at most $2f_{\mathcal{P}}(X)/\gamma$. As the sets $X \in \mathcal{P}_i$ are disjoint, it follows that $d_{i+1} - d_i$ can be routed in G with congestion at most $2f_{\mathcal{P}}(X)/\gamma$. Summing up the flows $d_{i+1} - d_i$ for $i \in \{1, \dots, L-1\}$ will then show that d can be routed in G with congestion at most $2(L-1)f_{\mathcal{P}}(X)/\gamma$.

We still have to show that the total demand that has to cross S in $G[X]$ is at most $2f_{\mathcal{P}}(X) \text{cap}(S, X \setminus S)/\gamma$. For $i > 1$ this demand is

$$\begin{aligned} \left| \sum_{v \in S} d_{i+1}(v) - \sum_{v \in S} d_i(v) \right| &\leq \sum_{v \in S} |d_{i+1}(v)| + \sum_{v \in S} |d_i(v)| \\ &= \sum_{v \in S} \frac{\deg_{\partial \mathcal{P}_{i+1}}(v)}{\deg_{\partial \mathcal{P}_{i+1}}(X_v^{(i+1)})} |d(X_v^{(i+1)})| \\ &\quad + \sum_{v \in S} \frac{\deg_{\partial \mathcal{P}_i}(v)}{\deg_{\partial \mathcal{P}_i}(X_v^{(i)})} |d(X_v^{(i)})| \\ &\leq \deg_{\partial \mathcal{P}_{i+1}}(S) + \deg_{\partial \mathcal{P}_i}(S) \\ &\leq 2 \deg_{\partial \mathcal{P}_{i+1}}(S) \\ &\leq 2f_{\mathcal{P}}(X) \text{cap}(S, X \setminus S)/\gamma, \end{aligned}$$

where the third inequality uses the fact that $\deg_{\partial \mathcal{P}_j}(Y) \geq |d(Y)|$ holds for any level j cluster Y , and the fifth inequality uses that $G[X]$ is $\deg_{\partial \mathcal{P}_{i+1}}$ -expanding with quality $\gamma/f_{\mathcal{P}}(X)$.

For $i = 1$ the only cluster is V . The demand in the flow $d_2 - d_1 = d_2$ that has to cross the cut S is

$$\begin{aligned} \left| \sum_{v \in S} d_2(v) \right| &\leq \sum_{v \in S} |d_2(v)| = \sum_{v \in S} \frac{\deg_{\partial \mathcal{P}_2}(v)}{\deg_{\partial \mathcal{P}_2}(X_v^{(2)})} |d(X_v^{(2)})| \\ &\leq \deg_{\partial \mathcal{P}_2}(S) \leq 2f_{\mathcal{P}}(V) \text{cap}(S, V \setminus S)/\gamma. \end{aligned}$$

Algorithm CONSTRUCTHIERARCHY(G):• **Construct levels 1 and 2.**

Let $\mathcal{P}$ be the (partial) hierarchical decomposition consisting only of one level, $\mathcal{P}_1 = \{V\}$, that contains only the root-cluster V . Let $\mathcal{X}$ be the partition of V into singletons and set $\phi \leftarrow 1/f_{\mathcal{P}}(V)$. Call **PARTITIONCLUSTER**($G, V, \mathcal{X}, \phi$) to obtain a set U and a partition $\mathcal{Y}$ of V . Note that then $G = G[V]$ is $\deg_{\partial\mathcal{Y}}$ -expanding with quality $\frac{1}{500}\phi/q^* = 2e\gamma/f_{\mathcal{P}}(V)$, as U must be empty (see **Observation 1**). We set $\mathcal{P}_1 = \{V\}$ and $\mathcal{P}_2 = \mathcal{Y}$. Then $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2)$ is a partial hierarchical decomposition with height $L = 2$.

• **Construct further levels.**

- (1) Assume $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_L)$ is the partial hierarchical decomposition constructed so far. If every cluster is a singleton cluster, return $\mathcal{P}$ and terminate the algorithm. Otherwise, mark every non-singleton cluster $C \in \mathcal{P}_L$ as unprocessed and initialize $\mathcal{X}_C$ as the partition of singletons for each such C .
- (2) **While** there is an unprocessed cluster $C \in \mathcal{P}_L$, process C by calling **PARTITIONCLUSTER**($G, C, \mathcal{X}_C, 1/f_{\mathcal{P}}(C)$) to obtain a set $U \subset C$ with $|U| \leq |C|/2$, and a partition $\mathcal{Y}$ of C .
 - If $U = \emptyset$, update $\mathcal{X}_C \leftarrow \mathcal{Y}$, mark C as processed, and continue with the next iteration of the while loop.
 - *Bad child event.* Otherwise, split C in $\mathcal{P}_L$ into two parts by removing C from $\mathcal{P}_L$ and inserting U and $C \setminus U$. Note that $U \in \mathcal{Y}$, so by removing U from $\mathcal{Y}$ we obtain a partition $\mathcal{Y}'$ of $C \setminus U$. Set $\mathcal{X}_{C \setminus U} \leftarrow \mathcal{Y}'$ and $\mathcal{X}_U \leftarrow \{U\}$.
 - If $\deg_{\partial\mathcal{Y}}(U) \geq \tau^*/20 \cdot \deg_{\partial\mathcal{Y}}(C)$ and $\deg_{\partial\mathcal{Y}}(C) \leq \deg_{\partial\mathcal{X}}(C) + 2\text{cap}(U, C \setminus U)$ (“either”-case of **Property 2**), mark both new clusters U and $C \setminus U$ as unprocessed.
 - Otherwise, mark U as unprocessed and $C \setminus U$ as processed.
- (3) Once all clusters of $\mathcal{P}_L$ are processed, set $\mathcal{P}_{L+1} = \bigcup_{C \in \mathcal{P}_L} \mathcal{X}_C$, add $\mathcal{P}_{L+1}$ to $\mathcal{P}$, and increment L .

This means that the subflow problem for any cluster X in the hierarchical decomposition can be solved with congestion $2f_{\mathcal{P}}(X)/\gamma$.

Now fix an edge $e = (u, v)$ and let k denote the largest level in the hierarchy such that u and v are contained in the same k -level cluster. Let X_i , $i \in \{1, \dots, k\}$ denote the i -level cluster that contains e . The congestion of e due to all flow problems is at most $\frac{2}{\gamma} \sum_{i=1}^k f_{\mathcal{P}}(X_i) \leq 6 \log \log n (1 + \sum_{i=2}^k (1 + \log |\text{par}_{\mathcal{P}}(X_i)|/|X_i|)) / \gamma = 6 \log \log n (k + \log |\text{par}_{\mathcal{P}}(X_1)| - \log |X_k|) / \gamma \leq 6 \log \log n (L + \log |V| - \log |X_k|) / \gamma \leq 6 \log \log n (L + \log n) / \gamma$. $\square$

5 Hierarchy Construction Algorithm

In this section we give an efficient algorithm for constructing a complete hierarchical decomposition $\mathcal{P} = (\mathcal{P}_1, \dots, \mathcal{P}_L)$ of logarithmic height, where each non-leaf cluster is γ -well expanding. The crucial subroutine for this top-down partitioning is **PARTITIONCLUSTER**, whose implementation is presented in **Section 6**. Here we first describe how **PARTITIONCLUSTER** is used to efficiently construct the hierarchy. It is defined via the following interface.

Subroutine (**PARTITIONCLUSTER**($G, C, \mathcal{X}, \phi$)). Given a graph $G = (V, E)$, a subset $C \subseteq V$, a partition $\mathcal{X}$ of C with $z := \max_{X \in \mathcal{X}} |X|$, and an expansion parameter ϕ with $0 < \phi \leq 1/4$, the procedure **PARTITIONCLUSTER** returns a (possibly empty) subset $U \subset C$ with $|U| \leq |C|/2$ and a new partition $\mathcal{Y}$ of C with $U \in \mathcal{Y}$ (if $U \neq \emptyset$) and $|Y| \leq \max\{z, |C|/2\}$ for all $Y \in \mathcal{Y}$. Furthermore,

- (1) the cut $E(U, C \setminus U)$ is $1/\phi$ -border routable through U with congestion 2, w.h.p.
- (2) **either** $\deg_{\partial\mathcal{Y}}(U) \geq (\tau^*/20) \cdot \deg_{\partial\mathcal{Y}}(C)$ and $\deg_{\partial\mathcal{Y}}(C) \leq \deg_{\partial\mathcal{X}}(C) + 2\text{cap}(U, C \setminus U)$
or $G[C \setminus U]$ is $\deg_{\partial\mathcal{Y}}$ -expanding with quality $\phi/(500q^*)$, w.h.p.

Consider the algorithm **CONSTRUCTHIERARCHY** at the top of the page. Starting with an initial partition that consists only of the set V , we apply **PARTITIONCLUSTER** repeatedly to the leaf clusters

of the current partial hierarchical decomposition until each leaf cluster consists of a singleton vertex. Whenever **PARTITIONCLUSTER** returns a non-empty set U , we call this a *bad child event*.

The first observation is that there is no bad child event in the first call to **PARTITIONCLUSTER**.

Observation 1. Let $\mathcal{X}$ be the partition of V into singletons. Then **PARTITIONCLUSTER**($G, V, \mathcal{X}, \phi$) returns an empty set U together with a partition $\mathcal{Y}$ s.t. G is $\deg_{\partial\mathcal{Y}}$ -expanding with quality $\frac{1}{500}\phi/q^*$.

This follows since V has no border edges, which implies that no non-trivial ϕ -border-routable set $U \subset V$ can exist. Thus U must be empty. Additionally, the output guarantees of **PARTITIONCLUSTER** imply that $G[V]$ is $\deg_{\partial\mathcal{Y}}$ -expanding with quality $\frac{1}{500}\phi/q^*$ as it is not possible that $0 = \deg_{\partial\mathcal{Y}}(U) \geq \tau^*/20 \cdot \deg_{\partial\mathcal{Y}}(C) > 0$.

The remainder of this section contains the proof correctness of the above algorithm. The main part is to prove expansion of every created cluster.

LEMMA 2. *Algorithm CONSTRUCTHIERARCHY constructs a complete hierarchical decomposition where every non-leaf cluster is γ -well-expanding with $\gamma = 1/1000eq^* = \Theta(1/\log n)$.*

PROOF. To prove the lemma we need to show that every non-leaf cluster C in the hierarchy is at least $\deg_{\partial\mathcal{P}_{i+1}}$ -expanding with quality $\gamma/f_{\mathcal{P}}(C)$, where i is the level of C . Fix a non-leaf cluster C on some level i in the final hierarchy. We first show that C is $\deg_{\partial\mathcal{P}_{i+1}}$ -expanding with quality $2e\gamma/f_{\mathcal{P}}(C)$, where e is the Euler’s constant, just after the level $i + 1$ has been constructed. However, this is not sufficient. During the construction of level $i + 2$ bad child events may occur, which means that clusters within $\mathcal{P}_{i+1}$ are further subdivided. This in turn may worsen the expansion property of the level- i cluster C , where this subdivisions happened. We will show that after the construction of level $i + 2$ is finished, C still is $\deg_{\partial\mathcal{P}_{i+1}}$ -expanding with quality $\gamma/f_{\mathcal{P}}(C)$ (i.e., the expansion is only a constant factor less).

The further construction of the hierarchy does not change $\mathcal{P}_{i+1}$ anymore, and, hence, C has the desired expansion in the end. As this holds for any C the lemma follows.

Step 1: Analysis of the expansion of level i clusters after the construction of level $i + 1$.

To construct $\mathcal{P}_{i+1}$ Algorithm **CONSTRUCTHIERARCHY** calls the subroutine **PARTITIONCLUSTER** on every non-processed clusters until no such clusters are left and then the union of the partition $\mathcal{X}_C$ of each processed cluster C forms the new partition $\mathcal{P}_{i+1}$. Therefore, to show that each cluster is $\deg_{\partial\mathcal{P}_{i+1}}$ -expanding with quality $2e\gamma/f_{\mathcal{P}}(C)$ just after level $i + 1$ has been constructed, it suffices that each processed cluster C is $\deg_{\partial\mathcal{X}_C}$ -expanding with quality $2e\gamma/f_{\mathcal{P}}(C)$ when it is marked as being processed.

To process an unprocessed cluster C , algorithm **CONSTRUCTHIERARCHY** calls **PARTITIONCLUSTER**($G, C, \mathcal{X}_C, 1/f_{\mathcal{P}}(C)$) which returns a set U and a new partition $\mathcal{Y}$. **CONSTRUCTHIERARCHY** then proceeds as follows. It makes sure that the cluster $C \setminus U$ is part of $\mathcal{P}_i$ either because $U = \emptyset$ and C already belongs to $\mathcal{P}_i$ or because $U \neq \emptyset$ and C is removed from $\mathcal{P}_i$ and $C \setminus U$ (and also U) are added to $\mathcal{P}_i$. The cluster U is marked as unprocessed. The cluster $C \setminus U$ is marked as processed if (i) either $U = \emptyset$ or (ii) the conditions of the “either”-case of **Property 2** of **PARTITIONCLUSTER** do not hold (i.e., $\deg_{\partial\mathcal{Y}}(U) < \tau^*/20 \cdot \deg_{\partial\mathcal{Y}}(C)$ or $\deg_{\partial\mathcal{Y}}(C) > \deg_{\partial\mathcal{X}}(C) + 2 \text{cap}(U, C \setminus U)$).

In both cases the partition $\mathcal{X}_{C \setminus U}$ returned by **PARTITIONCLUSTER** fulfills the first part of **Property 2** of **PARTITIONCLUSTER**, i.e., $G[C \setminus U]$ is $\deg_{\partial\mathcal{X}_{C \setminus U}}$ -expanding with quality $1/(500q^* f_{\mathcal{P}}(C))$. By setting $\gamma = 1/(1000eq^*)$ it follows that this quality equals $2e\gamma/f_{\mathcal{P}}(C)$.

The construction of level $i + 1$ only ends when all clusters are marked as processed and, thus, each cluster C in $\mathcal{P}_i$ is $\deg_{\partial\mathcal{X}_C}$ -expanding with quality $2e\gamma/f_{\mathcal{P}}(C)$. Once all clusters in $\mathcal{P}_i$ are marked as processed it follows that each cluster C in $\mathcal{P}_i$ is $\deg_{\partial\mathcal{P}_{i+1}}$ -expanding with quality $2e\gamma/f_{\mathcal{P}}(C)$ as $\deg_{\partial\mathcal{X}_C}|_C = \deg_{\partial\mathcal{P}_{i+1}}|_C$.

Step 2: Analysis of the expansion of level i clusters after the construction of level $i + 2$.

Fix a level- i cluster C after level $i + 1$ of the hierarchy has been constructed, and let $\mathcal{X}$ denote the partition of C at this time. During the construction of level $i + 2$ the partition of C may change due to bad child events. Let $\mathcal{Y}$ denote the partition of C after the construction of level $i + 2$ has finished. Observe that $\mathcal{Y}$ is a refinement of $\mathcal{X}$. We need to show that C is $\deg_{\partial\mathcal{P}_{i+1}}$ -expanding with quality $\gamma/f_{\mathcal{P}}(C)$ when the construction of level $i + 2$ is complete. This is equivalent to showing that C is $\deg_{\partial\mathcal{Y}}$ -expanding with this quality.

Let $\mathbf{d}$ be an arbitrary demand vector in C with $|\mathbf{d}| \leq \deg_{\partial\mathcal{Y}}|_C$. We show that we can route $\mathbf{d}$ with congestion at most $f_{\mathcal{P}}(C)/\gamma$ inside $G[C]$, which implies the desired expansion guarantee for C . The idea is to (a) first route $\mathbf{d}$ from $\deg_{\partial\mathcal{Y}}$ to $\deg_{\partial\mathcal{X}}$ with small congestion (using border-routability) and call the resulting flow the *to-border flow for C* and (b) then apply the fact that we know from above how to route any demand $\mathbf{d}'$ with $|\mathbf{d}'| \leq \deg_{\partial\mathcal{X}}$ with small congestion.

More formally, in order to route $\mathbf{d}$ we will (a) route a to-border $(\mathbf{d}, \mathbf{t})$ -flow for some suitable demand $\mathbf{t}$ with $|\mathbf{t}| \leq \alpha \deg_{\partial\mathcal{X}}|_C$ for some constant $\alpha \leq e$ with congestion $\text{con} \leq 1/(2\gamma)$. Then (b) we are left with routing demand $\mathbf{t}$ inside $G[C]$, but we know already that we can do that with small congestion: We are guaranteed that $G[C]$ is $2e\gamma/f_{\mathcal{P}}(C) \cdot \deg_{\partial\mathcal{X}}$ -expanding, and, thus, by **Lemma 1**, we

can route any demand $\mathbf{d}'$ with $|\mathbf{d}'| \leq \alpha \deg_{\partial\mathcal{X}}|_C$ with congestion $\alpha f_{\mathcal{P}}(C)/(2e\gamma) \leq f_{\mathcal{P}}(C)/(2\gamma)$ inside $G[C]$ for $\alpha \leq e$. The sum of the two flows then has congestion at most $f_{\mathcal{P}}(C)/\gamma$ and routes $\mathbf{d}$.

To-border flow for C . It remains to show that for any demand vector $\mathbf{d}$ with $|\mathbf{d}| \leq \deg_{\partial\mathcal{Y}}|_C$ there exists a demand vector $\mathbf{t}$ with $|\mathbf{t}| \leq \alpha \deg_{\partial\mathcal{X}}|_C$ for some suitable constant $\alpha \leq e$ such that we can route a to-border $(\mathbf{d}, \mathbf{t})$ -flow with small congestion con .

Specifically, we need to upper bound con by $4\alpha \log n$. This gives the desired bound $\text{con} \leq 4\alpha \log n \leq f_{\mathcal{P}}(C)/(2\gamma)$, as $4\alpha \log n \leq 4e \log n \leq \log \log n/(2\gamma)$. For this, recall that $q^* \geq (\log n)/125$ and $\gamma = 1/(1000eq^*) \leq \log \log n/(8e \log n)$.

We need to find a demand $\mathbf{t}$ and route the $(\mathbf{d}, \mathbf{t})$ -flow f inside C with congestion at most $4\alpha \log n$. As $\mathbf{d}$ is a demand and the flow f routes the demand $\mathbf{d} - \mathbf{t}$, it follows that $\mathbf{t}$ is a demand. Thus, it is sufficient to show that there exists a *vertex weight vector* $\mathbf{t}$ such there exists a $(\mathbf{d}, \mathbf{t})$ -flow f inside C with congestion at most $4\alpha \log n$.

To do we will show for every cluster $X \subseteq C$ with $X \in \mathcal{X}$ that there exists a vertex weight vector $\mathbf{t}|_X$ with $|\mathbf{t}|_X| \leq \alpha \deg_{\partial\mathcal{X}}|_X$ so that we can route the flow $(\mathbf{d}|_X, \mathbf{t}|_X)$ in $G[X]$ with congestion at most $4\alpha \log n$. Note that $\mathbf{d}$ equals $\sum_{X \in \mathcal{X}} \mathbf{d}|_X$. Let $\mathbf{t} := \sum_{X \in \mathcal{X}} \mathbf{t}|_X$. Note that $\mathbf{t}$ is a vertex weight vector with $|\mathbf{t}| \leq \alpha \deg_{\partial\mathcal{X}}|_C$ and the combination of the resulting the $(\mathbf{d}|_X, \mathbf{t}|_X)$ -flows over all clusters $X \in \mathcal{X}$ routes exactly the $(\mathbf{d}, \mathbf{t})$ -flow inside C .

Fix a cluster $X \in \mathcal{X}$. During the construction of level $i + 2$, X might be repeatedly partitioned due to bad child events. We represent this partitioning process by a binary tree T : the root vertex is X , the internal vertices are intermediate clusters created during the partitioning process, and the leaf vertices are the child-clusters of X in $\mathcal{Y}$, and, thus, in the final hierarchical decomposition. Thus nodes in T are always clusters that are created during the partitioning process, i.e. during the construction of level $i + 2$ in **CONSTRUCTHIERARCHY**. Only the root and the leaves of T exist when the construction of level $i + 2$ ends. At every internal node of T a flow is constructed between its two children and internal to one child. The to-border flow for C will be the *sum* of all these flows.

Every non-leaf cluster S has two children: one bad child U and $S \setminus U$ (we assume wlog. that U is the left child in the binary tree). We say an internal node has at left-depth ℓ in T , if the number of left edges on its path to the root is ℓ . The root X has thus left-depth 0. Let h denote the left height of T , i.e., the maximum left depth of any node in T .

CLAIM 1. *A node S of T at left-depth ℓ has $|S| \leq |X|/2^\ell$ and T has left height at most $\log n$.*

This can be seen since a bad child U of a parent P has $|U| \leq |P|/2$, thus the left-depth of any cluster can be at most $\log |X| \leq \log n$. Now by restricting $\mathbf{d}|_X$ to a node S of T we receive $\mathbf{d}|_S := (\mathbf{d}|_X)|_S$.

CLAIM 2. *For a node U of T that is a bad child at left depth ℓ for any $\mathbf{s}$ with $|\mathbf{s}| \leq \deg_{E(U, S \setminus U)}|_U$ there exists a $\mathbf{t}$ with $|\mathbf{t}| \leq \deg_{\partial S}|_U/(3\ell \log \log n)$ such that we can route the $(\mathbf{s}, \mathbf{t})$ -flow in $G[U]$ with congestion 2.*

PROOF. During the construction of the hierarchy we obtain the bad child U of a cluster S by calling the subroutine **PARTITIONCLUSTER**($G, S, \mathcal{X}_S, 1/f_{\mathcal{P}}(S)$) on some cluster S with $\phi = 1/f_{\mathcal{P}}(S)$ and we are guaranteed that the edge set $E(U, S \setminus U)$ is $\frac{1}{\phi}$ -border routable,

i.e., $f_P(S)$ -border routable in S , through U with congestion 2. Since $X = \text{par}_{P_{i+1}}(S)$ in the hierarchical decomposition when **PARTITIONCLUSTER** is called, it follows that $\log(|\text{par}_{P_{i+1}}(S)|/|S|) \geq \ell$. Thus, it holds that $f_P(S) \geq 3 \log \log n \log(2|X|/|S|) \geq 3\ell \log \log n$. Thus, $f_P(S)$ -border routability implies $3\ell \log \log n$ -border routability. Furthermore, the definition of border routability guarantees that for any s with $|s| \leq \deg_{E(U, S \setminus U)}|U|$ there is a t with $|t| \leq \deg_{E(U, V \setminus S)}|U|/(3\ell \log \log n) = \deg_{\partial S}|U|/(3\ell \log \log n)$ such that we can route the (s, t) -flow in $G[U]$ with congestion 2. $\square$

Now we are ready to give a construction for the vertex weight vector $t|_X$ and explain how to route the $(d|_X, t|_X)$ -flow inside cluster X with congestion at most $4\alpha \log n$. Define for a left depth $\ell \in \{0, \dots, h\}$

$$\alpha_\ell = \prod_{j=h-1}^{\ell} \left(1 + \frac{1}{(j+1) \log \log n}\right) = \left(1 + \frac{1}{(\ell+1) \log \log n}\right) \alpha_{\ell+1},$$

and $\alpha_h = 1$. Note that $\alpha_\ell > \alpha_{\ell+1} \geq 1$ for $0 \leq \ell < h$ and

$$\alpha := \alpha_0 = \prod_{j=h-1}^0 \left(1 + \frac{1}{(j+1) \log \log n}\right) \leq \exp\left(\frac{1}{\log \log n} \sum_{j=1}^h \frac{1}{j}\right) \leq e,$$

where we used $h \leq \log_2 n$.

We prove via induction over T (from leaves to the root X) the following **inductive claim**: *For any node S of T at left depth ℓ there exists a vertex weight vector $t|_S$ with $|t|_S| \leq \alpha_\ell \deg_{\partial S}|S|$ and we can route the flow $(d|_S, t|_S)$ inside S such that the congestion is 0, if S is a leaf cluster, and the congestion is α_ℓ on the edges of $E(U, S \setminus U)$, and $4\alpha_\ell(h - \ell)$ for edges in $G[U]$ and $G[S \setminus U]$, where U and $S \setminus U$ are the children of S in T with U being the bad child.*

Note that this implies that for the root X (which has left depth 0) there exists a $(d|_X, t|_X)$ -flow inside cluster X for some vertex weight vector $t|_X$ with $|t|_X| \leq \alpha \deg_{\partial X}|X|$ with congestion at most $\alpha h \leq \alpha \log n$.

For a leaf cluster S (which means $S \in \mathcal{Y}$) we have $|d|_S| \leq \deg_{\partial \mathcal{Y}}|S| = \deg_{\partial S}|S| \leq \alpha_h \deg_{\partial S}|S|$, so we set $t|_S = d|_S$, i.e., we fulfill the condition without any routing, i.e., congestion 0.

Now suppose that we have a non-leaf cluster S at left depth $\ell < h$ that has a bad child U . Further, assume that we already found vertex weight vectors $t'|_U$ and $t'|_{S \setminus U}$ and a $(d|_U, t'|_U)$ -flow with congestion $\alpha_{\ell+1}(h - \ell - 1)$ and a $(d|_{S \setminus U}, t'|_{S \setminus U})$ -flow with congestion $\alpha_\ell(h - \ell)$ so that $|t'_U| \leq \alpha_{\ell+1} \deg_{\partial U}$ and $|t'_{S \setminus U}| \leq \alpha_\ell \deg_{\partial(S \setminus U)}$ (observe that the left depth of U is $\ell + 1$).

We first route a $(t'|_{S \setminus U}, t^*)$ -flow at vertices from $S \setminus U$ that are incident to U . For this, a vertex $v \in S \setminus U$ with $t'|_{S \setminus U}(v) \geq 0$ sends $\alpha_\ell \deg_{E(U, S \setminus U)}(v)$ flow along its incident $E(U, S \setminus U)$ -edges to U (v may send less if it is running out of “supply”) and if $t'|_{S \setminus U}(v) < 0$ v receives $\alpha_\ell \deg_{E(U, S \setminus U)}(v)$ flow (v may receive less flow if its “deficit” becomes zero) across each edge of $E(U, S \setminus U)$ that is incident to v . More formally, for $v \in S \setminus U$, we set $t^*(v) = \max(0, t'|_{S \setminus U}(v) - \alpha_\ell \deg_{E(U, S \setminus U)}(v))$ if $t'|_{S \setminus U}(v) \geq 0$ and $t^*(v) = \min(0, t'|_{S \setminus U}(v) + \alpha_\ell \deg_{E(U, S \setminus U)}(v))$ otherwise. Define $t^*(v)$ for $v \in U$ correspondingly. Note that this can be done with congestion α_ℓ and that for $v \in S \setminus U$, $|t^*(v)| \leq \alpha_\ell \deg_{\partial(S \setminus U)}(v) - \alpha_\ell \deg_{E(U, S \setminus U)}(v) = \alpha_\ell \deg_{\partial S}(v)$.

After this step $t^*(v)$ at a vertex $v \in U$ fulfills

$$\begin{aligned} |t^*(v)| &\leq |t'|_U|(v) + \alpha_\ell \deg_{E(U, S \setminus U)}(v) \\ &\leq \alpha_{\ell+1} \deg_{\partial U}(v) + \alpha_\ell \deg_{E(U, S \setminus U)}(v) \\ &= \alpha_{\ell+1} \deg_{\partial S}(v) + (\alpha_\ell + \alpha_{\ell+1}) \deg_{E(U, S \setminus U)}(v). \end{aligned}$$

In a second step we want to reduce $|t^*(v)|$ for each $v \in U$ to at most $\alpha_{\ell+1} \deg_{\partial S}(v)$. Recall that U is a bad child. Thus, we use **Claim 2** that shows for $\tilde{s}$ with $|\tilde{s}| \leq (\alpha_\ell + \alpha_{\ell+1}) \deg_{E(U, S \setminus U)}$ the existence of a vertex weight $\tilde{t}$ with $|\tilde{t}| \leq (\alpha_\ell + \alpha_{\ell+1}) \deg_{\partial S}|U|/(3(\ell + 1) \log \log n) = (\alpha_\ell + \alpha_{\ell+1}) \deg_{\partial S}/(3(\ell + 1) \log \log n)$ and a $(\tilde{s}, \tilde{t})$ -flow in $G[U]$ with congestion $2(\alpha_\ell + \alpha_{\ell+1}) \leq 4\alpha_\ell$. Thus, we apply the claim with $\tilde{s}(v) := t^*(v) - \alpha_{\ell+1} \deg_{\partial S}(v)$ if $t^*(v) > \alpha_{\ell+1} \deg_{\partial S}(v)$, $\tilde{s}(v) := t^*(v) + \alpha_{\ell+1} \deg_{\partial S}(v)$ if $t^*(v) < -\alpha_{\ell+1} \deg_{\partial S}(v)$, and $\tilde{s} = 0$ otherwise. Note that $|\tilde{s}| \leq \max(0, |t^*| - \alpha_{\ell+1} \deg_{\partial S}) \leq (\alpha_\ell + \alpha_{\ell+1}) \deg_{E(U, S \setminus U)}$, i.e., it fulfills the requirements of **Claim 2**. Finally, we set $t|_S(v) = t^*(v)$ for $v \in S \setminus U$ and $t|_S(v) = t^*(v) - \tilde{s}(v) + \tilde{t}(v)$ for $v \in U$. Thus, $|t|_S|(v)$ at any vertex $v \in U$ is at most

$$\begin{aligned} |t|_S|(v) &\leq |t^*(v) - \tilde{s}(v)| + |\tilde{t}(v)| \\ &\leq \left(\alpha_{\ell+1} + \frac{\alpha_{\ell+1} + \alpha_\ell}{3(\ell + 1) \log \log n}\right) \deg_{\partial S}(v) \\ &\leq \left(\alpha_{\ell+1} + \frac{\alpha_{\ell+1}}{(\ell + 1) \log \log n}\right) \deg_{\partial S}(v) = \alpha_\ell \deg_{\partial S}(v) \end{aligned}$$

where we used $\alpha_\ell \leq 2\alpha_{\ell+1}$, and for a vertex $v \in S \setminus U$ it is at most $\alpha_\ell \deg_{\partial S}(v)$.

Finally we analyze the congestion. The edges in $E(U, S \setminus U)$ have no congestion in the recursive flows $(d|_U, t'|_U)$ -flow and a $(d|_{S \setminus U}, t'|_{S \setminus U})$ -flow and receive congestion α_ℓ in step one. By the inductive assumption the edges in $G[U]$ have congestion of $4\alpha_{\ell+1}(h - (\ell + 1))$ and receive a congestion of $4\alpha_\ell$ in step two. Thus their total congestion is $4\alpha_\ell(h - \ell)$. By the inductive assumption the edges in $G[S \setminus U]$ have congestion of $4\alpha_\ell(h - \ell)$ and receive no additional congestion in either step one or step two. Thus their congestion is still $4\alpha_\ell(h - \ell)$. It follows that we can route the flow $(d|_S, t|_S)$ inside S with congestion $4\alpha_\ell(h - \ell)$. This completes the proof of **Lemma 2** $\square$

LEMMA 3. *The complete hierarchical decomposition constructed by Algorithm **CONSTRUCTHIERARCHY** has height $L = O(\log n)$.*

PROOF. We show that any cluster C in the final hierarchy has size at most $|P|/2$, where P is the grandparent of C .

At the start of the while loop a cluster S on level L is chosen. Let P denote the parent of S in the hierarchy at this time and observe that P is also contained in the final hierarchy while S might not be due to bad child events that subdivide S .

The call to **PARTITIONCLUSTER** $(G, S, \mathcal{X}_S, 1/f_P(S))$ at the beginning of the while loop guarantees that the bad child U that is returned has size at most $|S|/2$ and all clusters in the returned partition $\mathcal{Y}$ have size at most $|S|/2$. Any generated sub-cluster in this or the following iterations on clusters that are subsets of C are a set or a subset of the partition $\mathcal{Y}$ and will therefore have size at most $|S|/2$. Such a sub-cluster C is then inserted as a child of S or as a child of a part of S that is obtained via a bad child event. In either case the size of C is at most $|S|/2 \leq |P|/2$, where P is the parent of S , and, hence, the grandparent of C . $\square$

LEMMA 4. *The time for executing the while loop is $O(\log(m)/\tau^* \cdot T_{\text{part}}(m))$, where $T_{\text{part}}(m) = \Omega(m)$ is the running time for the subroutine **PARTITIONCLUSTER**.*

PROOF. Fix a leaf cluster C at the start of the while loop. We analyze how much work the algorithm performs on sub-clusters of C until all sub-clusters of C are marked as processed.

The following claim shows that a bad child event U on some sub-cluster $S \subseteq C$ makes progress in the sense that an *unprocessed* new cluster resulting from the bad child event is substantially “smaller” than S .

CLAIM 3. *Suppose a call to **PARTITIONCLUSTER**($G, S, \mathcal{X}, \phi$) on some subcluster S returns with a bad child U . Then the resulting sub-clusters U and $S \setminus U$ fulfill*

- (1) $|U| \leq |S|/2$
- (2) $\deg_{\partial \mathcal{Y}}(S \setminus U) \leq (1 - \tau^*/80) \deg_{\partial \mathcal{X}}(S)$ if $S \setminus U$ is declared as *unprocessed* by Algorithm **CONSTRUCTHIERARCHY**, where $\mathcal{Y}$ denotes the new partition of S .

PROOF. **Property 1** is directly guaranteed by the output of **PARTITIONCLUSTER**. For **Property 2** observe that $S \setminus U$ is only declared *unprocessed* if we are in the “either”-case of **Property 2** in **PARTITIONCLUSTER**, i.e., if $\deg_{\partial \mathcal{Y}}(U) \geq \tau^*/20 \cdot \deg_{\partial \mathcal{Y}}(S)$ and $\deg_{\partial \mathcal{Y}}(S) \leq \deg_{\partial \mathcal{X}}(S) + 2 \text{cap}(U, S \setminus U)$. This gives

$$\begin{aligned} \deg_{\partial \mathcal{Y}}(S \setminus U) &= \deg_{\partial \mathcal{Y}}(S) - \deg_{\partial \mathcal{Y}}(U) \\ &\leq \deg_{\partial \mathcal{X}}(S) + 2 \text{cap}(U, S \setminus U) - \deg_{\partial \mathcal{Y}}(U) \\ &\leq \deg_{\partial \mathcal{X}}(S) + 2\phi \deg_{\partial \mathcal{Y}}(U) - \deg_{\partial \mathcal{Y}}(U) \\ &\leq \deg_{\partial \mathcal{X}}(S) + (2\phi - 1)\tau^*/20 \cdot \deg_{\partial \mathcal{Y}}(S) \\ &\leq \deg_{\partial \mathcal{X}}(S) - \tau^*/40 \cdot \deg_{\partial \mathcal{Y}}(S) \end{aligned}$$

The second inequality holds because **Property 1** of **PARTITIONCLUSTER** implies that $\text{cap}(U, S \setminus U) \leq \phi \text{cap}(U, V \setminus S) \leq \phi \deg_{\partial \mathcal{Y}}(U)$ and the final inequality uses $\phi \leq 1/4$. If now $\deg_{\partial \mathcal{Y}}(S) \leq \deg_{\partial \mathcal{X}}(S)/2$ we clearly have $\deg_{\partial \mathcal{Y}}(S \setminus U) \leq \deg_{\partial \mathcal{Y}}(S) \leq \deg_{\partial \mathcal{X}}(S)/2$. Otherwise, we obtain $\deg_{\partial \mathcal{Y}}(S \setminus U) \leq (1 - \tau^*/80) \deg_{\partial \mathcal{X}}(S)$ by plugging $\deg_{\partial \mathcal{Y}}(S) \geq \deg_{\partial \mathcal{X}}(S)/2$ into the above inequality. The claim follows. $\square$

From the above claim it directly follows that a vertex from C can be contained in a sub-cluster $S \subseteq C$ at most $\log_2 |C| + \log_{1/(1-\tau^*/80)} \deg(C) = O(\log(\deg(C))/\tau^*)$ times during the while loop. Consequently, the total work performed for calls to **PARTITIONCLUSTER** during the while loop is at most $O(\log(m)/\tau^* \cdot \sum_C T_{\text{part}}(m_C)) \leq O(\log(m)/\tau^* \cdot T_{\text{part}}(m))$, where $T_{\text{part}}(m)$ is the time required for Algorithm **PARTITIONCLUSTER** on a cluster with volume m (the inequality uses the fact that $T_{\text{part}}(m)$ grows at least linearly with the volume). The remaining cost of inserting a new cluster into the partition, or comparing the size of clusters is just linear in m , and, hence dominated by the cost for calling $T_{\text{part}}(m)$. $\square$

THEOREM 6. *Suppose we are given an undirected graph $G = (V, E)$ with volume $m = \deg(V)$, and an algorithm for **PARTITIONCLUSTER** with running time T_{part} . Then we can construct a hierarchical congestion approximator with quality $O(q^* \log n \log \log n)$ in time $O(\log^2(m)/\tau^* \cdot T_{\text{part}}(m))$. Here $q^* = O(\log n)$ and $\tau^* \leq 1/160$ are the quality- and balance-guarantee, respectively, of the **SPARSESTCUTAPX**-routine used by **PARTITIONCLUSTER**.*

PROOF. Lemma 2 shows that the hierarchy constructed by algorithm **CONSTRUCTHIERARCHY** is γ -well-expanding with parameter $\gamma = \Theta(1/\log n)$. Lemma 3 shows that our hierarchical decomposition has height $L = O(\log n)$. Hence, the quality of the congestion approximator follows by Theorem 5. The running time for constructing one level of the hierarchy is $O(\log(m)/\tau^* \cdot T_{\text{part}}(m))$ due to Lemma 4. Since the number of levels is logarithmic due to Lemma 3 the theorem follows. $\square$

Plugging in the result for our building blocks from Section 3 we get the following corollary.

COROLLARY 4. *With the implementations of the building blocks in Section 3 we obtain a running time of $O(\log^2(m)/\tau^* \cdot T_{\text{part}}(m)) = O(\log^3(m) \cdot T_{\text{part}}(m)) = O(\log^{10} \cdot T_{\text{fc}}(m, 2)) = \tilde{O}(m)$.*

6 Partitioning a Cluster

In this section we present the algorithm **PARTITIONCLUSTER** that we use to compute a new partition $\mathcal{Y}$ of a cluster C when given a partition $\mathcal{X}$ and an expansion parameter ϕ . Recall its definition at the beginning of Section 5.

The algorithm first calls **SPARSESTCUTAPX** with the vertex weight $\pi = \deg_{\partial \mathcal{X}}$ to find a cut in C that is roughly ϕ sparse w.r.t. π . This cut is then “slightly shifted” using a subroutine called **TWOWAYTRIM**, resulting in a cut that is border-routable through one side (called U). Ideally, we would like the other side of the cut to be suitably expanding such that the algorithm can terminate. This however only happens under certain conditions. If they do not hold, then we either (i) modify the original partition $\mathcal{X}$ so that $\deg_{\partial \mathcal{X}}(C)$ decreases by a multiplicative factor and repeat the algorithm from the beginning or (ii) we “trim” the current cut to obtain a border-routable set and a partition for which the balance condition in **Property 2** of **PARTITIONCLUSTER** holds.

The subroutine to “slightly shift” a cut is defined as follows.

Subroutine (**TWOWAYTRIM**(G, C, R, π, ϕ)). We are given a graph $G = (V, E)$, two subsets R, C with $R \subset C \subseteq V$, a non-negative vertex weighting π , and a parameter $\phi > 0$ such that

- (a) $\text{cap}(R, C \setminus R) \leq \phi \pi(R)$, and
- (b) $G[C]$ is $\delta \cdot \phi \pi|_{C \setminus R}$ -expanding for some $\delta \in (0, 1]$.

The procedure **TWOWAYTRIM** outputs a three-partition of C into (A, B, U) such that

- (a) $A \subseteq C \setminus R$ and $\text{cap}(A, C \setminus A) \leq 2 \text{cap}(R, C \setminus R)$
- (b) $\pi(B \cup U) \leq (11/\delta) \cdot \pi(R)$
- (c) $E(U, C \setminus U)$ is $\frac{1}{\phi}$ -border routable through U with congestion 2
- (d) $G[A \cup B]$ is $\frac{1}{25} \cdot \delta \phi (\deg_{E(B, V \setminus B)}|_{A \cup B} + \pi|_A)$ -expanding

We implement **TWOWAYTRIM** by computing fair cuts “towards both sides” of the given cut. More precisely, we use this algorithm.

Algorithm **TWOWAYTRIM**(G, C, R, π, ϕ):

- (1) Define a flow problem in $G[C \setminus R]$: The source function is $s_1 = \deg_{E(R, C \setminus R)}$, the target function is $t_1 = 1/5 \cdot \delta \phi \pi$ and run **FAIRCUTFLOW**($G[C \setminus R], s_1, t_1, \alpha_1$) with $\alpha_1 = 2$. This results in a 2-fair (s_1, t_1) -cut X_0 of $C \setminus R$. Let $X_1 = X_0 \cup R$. We set $A := C \setminus X_1$ and thus obtain the partition (X_1, A) of C .
- (2) Define a flow problem in $G[X_1]$: The source function is $s_2 = \deg_{E(X_1, A)}$, the target function is $t_2 = \frac{1}{2} \phi \deg_{E(C, V \setminus C)}$. Then

run **FAIRCUTFLOW**($G[X_1], s_2, t_2, \alpha_2$) with $\alpha_2 = 2$. This results in a 2-fair cut X_2 and we set $U := X_1 \setminus X_2$ and $B := X_2$. Notice that (A, B, U) now forms a three-partition of C , return it.

We defer the proof that this algorithm correctly implements the subroutine **TWOWAYTRIM** to the full version. The result is the following lemma.

LEMMA 5. **TWOWAYTRIM** can be implemented using two executions of **FAIRCUTFLOW** with $O(m)$ additional work.

Let us define the following notation. Suppose we are given a cluster S , a partition $\mathcal{X}$ of S , and a subset $T \subseteq S$. The operation $\mathcal{X} - T$ returns the partition $\{A \setminus T \mid A \in \mathcal{X}, A \setminus T \neq \emptyset\}$ of $S \setminus T$. When modifying a partition $\mathcal{X}$ of C we use the operation $\mathcal{X} \leftarrow (\mathcal{X} - T) \cup \{T\}$, for which we say that we *fuse the set T in $\mathcal{X}$* , i.e., we first remove T from every set in the partition and then we add T itself as set to the partition.

CLAIM 4. For any $B \subseteq C$ if $\mathcal{Y} = (\mathcal{X} - B) \cup \{B\}$, it holds that $\deg_{\partial\mathcal{Y}}(C) \leq \deg_{\partial\mathcal{X}}(C) - \deg_{\partial\mathcal{X}}(B) + 2 \text{cap}(B, C \setminus B) + \text{cap}(B, V \setminus C)$.

Thus, if $\deg_{\partial\mathcal{X}}(B)$ is large in comparison to $2 \text{cap}(B, C \setminus B) + \text{cap}(B, V \setminus C)$ (i.e., a large part of $\deg_{\partial\mathcal{X}}(B)$ is “inside” B), the fuse operation “decreases volume”, i.e., $\deg_{\partial\mathcal{Y}}(C) < \deg_{\partial\mathcal{X}}(C)$. Otherwise, the “volume increase” is limited, i.e., $\deg_{\partial\mathcal{Y}}(C) \leq \deg_{\partial\mathcal{X}}(C) + 2 \text{cap}(B, C \setminus B)$.

Next we present the algorithm **PARTITIONCLUSTER**, whose correctness we prove afterwards.

Algorithm PARTITIONCLUSTER($G, C, \mathcal{X}, \phi$)

- (1) Set $\pi \leftarrow \deg_{\partial\mathcal{X}}$ and $R \leftarrow \text{SPARSESTCUTAPX}(G[C], \pi, \phi/20)$
 - (a) If R has small π volume $[\pi(R) \leq \tau^* \cdot \pi(C)]$ then $(A, B, U) \leftarrow \text{TWOWAYTRIM}(G, C, R, \pi, \phi)$. If $|A| \geq |C|/2$, perform two fuse operations, namely set $\mathcal{Y} \leftarrow (\mathcal{X} - B) \cup \{B\}$ and $\mathcal{Y} \leftarrow (\mathcal{Y} - U) \cup \{U\}$ and return $\mathcal{Y}$ and U . Otherwise, set $T \leftarrow A$ and go to Step 2.
 - (b) If R does not have small π volume $[\pi(R) \geq \tau^* \cdot \pi(C)]$, then set T to be the set out of $\{R, C \setminus R\}$ that has the smaller cardinality and go to Step 2.
- (2) Differentiate two cases:
 - (a) If $\deg_{\partial C}(T) \leq \pi(T)/2$, then fuse T in $\mathcal{X}$, i.e., update the partition $\mathcal{X} \leftarrow (\mathcal{X} - T) \cup \{T\}$ and go to Step 1.
 - (b) Otherwise, “trim” T and return a balanced bad child T' . Formally, let $s = \deg_{E(T, C \setminus T)}|_T$ and $t = \frac{1}{2}\phi \deg_{\partial C}|_T$. We compute a 2-fair cut $X \subseteq T$ in $G[T]$ using **FAIRCUTFLOW**($G[T], s, t, 2$) and let $T' = T \setminus X$. We then return T' together with the partition that is generated by fusing T' in $\mathcal{X}$, namely $\mathcal{Y} \leftarrow (\mathcal{X} - T') \cup \{T'\}$.

Recall that if the algorithm returns a non-empty cluster U (together with the partition $\mathcal{Y}$), we call this cluster a *bad child*. We call each execution of Step 1 and Step 2 an *iteration*. Note that the procedure terminates in Case 1a and Case 2b, and only starts a new iteration if it reaches Case 2a. As we will show below, in the latter case, the remaining π volume decreases sufficiently so that this cannot happen too often.

The correctness proof now proceeds as follows: We first show in Claim 5 that the conditions for **TWOWAYTRIM** are fulfilled whp whenever it is called in Case 1a. Then we show that T fulfills a certain set of conditions at the beginning of Step 2. This is then

sufficient to show correctness, i.e., that Property 1 and Property 2 of **PARTITIONCLUSTER** are fulfilled at termination.

CLAIM 5. In Case 1a of the algorithm the conditions for executing **TWOWAYTRIM** are fulfilled for $\delta = 1/(20q^*)$ with high probability.

INVARIANT 1. At the beginning of Step 2 the set T fulfills

- (1) $T \leq |C|/2$,
- (2) $\pi(T) \geq \tau^* \pi(C)$, and
- (3) $\text{cap}(T, C \setminus T) \leq \phi/10 \cdot \pi(T)$.

PROOF. **Case 1:** First suppose the set T is due to the assignment in Case 1b. As R was returned by **SPARSESTCUTAPX**, it holds that $\pi(R) \leq \pi(C \setminus R)$. Thus, we have $\pi(C \setminus R) \geq \pi(R) \geq \tau^* \cdot \pi(C)$, which shows Property 2 for both R and $C \setminus R$, and, thus, for T . Since the cut R is $\phi/20$ -sparse we have $\pi(C \setminus R) \geq \pi(R) \geq 20/\phi \cdot \text{cap}(R, C \setminus R)$, which gives Property 3 for both R and $C \setminus R$. As T is chosen as the set of smaller cardinality out of R and $C \setminus R$ we get Property 1 for T .

Case 2: Now suppose that T is due to the assignment in Case 1a, i.e., it is the set A that results from **TWOWAYTRIM**. Property 1 directly follows from the choice of T in that case. We have

$$\pi(A) = \pi(C) - \pi(B \cup U) \geq \pi(C) - \frac{1}{\delta} \pi(R) \geq \pi(C)/2,$$

where we used Property (b) of **TWOWAYTRIM** for the first inequality and $\delta = 1/(20q^*)$ by Claim 5. In the second inequality, since $\pi(R) \leq \tau^* \pi(C) \leq 1/(440q^*) \pi(C)$, we have that $(11/\delta) \pi(R) \leq 220q^* \tau^* \pi(C) \leq \frac{220q^*}{440q^*} \pi(C) = \pi(C)/2$.

Property (a) of **TWOWAYTRIM** and the sparsity of R guaranteed by **SPARSESTCUTAPX** called with $\phi/20$ give us $\text{cap}(A, C \setminus A) \leq 2 \text{cap}(R, C \setminus R) \leq \phi/10 \cdot \pi(R) \leq \phi/10 \cdot \pi(A)$, where the last step uses $\pi(R) \leq \pi(C)/2 \leq \pi(A)$. This establishes Property 3 and concludes the proof. $\square$

INVARIANT 2. In any iteration of **PARTITIONCLUSTER**, all $X \in \mathcal{X}$ satisfy $|X| \leq \max\{z, |C|/2\}$, where z is the maximum size of a set in the initial partition.

PROOF. For the initial iteration by the definition of z . The algorithm only continues to another iteration in Case 2a if $|A| < |C|/2$. Then we obtain the new partition $\mathcal{X}'$ by setting $\mathcal{X}' = (\mathcal{X} - T) \cup \{T\}$ for some set T that satisfies the properties of Invariant 1. In particular, we have $|T| \leq |C|/2$. The inductive step thus follows since no set $X \in \mathcal{X}$ can increase in cardinality by the fuse operation and $|T| \leq |C|/2$. $\square$

LEMMA 6. If **PARTITIONCLUSTER** returns from Case 1a, then it returns a correct (imbalanced) bad child, i.e., Property 1 and Property 2 of **PARTITIONCLUSTER** are fulfilled.

PROOF. Claim 5 shows that the conditions for **TWOWAYTRIM** are fulfilled with $\delta = 1/(20q^*)$. In this case, the call to **TWOWAYTRIM** resulted in a three-partition (A, B, U) such that $|A| \geq |C|/2$. The algorithm returns U alongside the partition $\mathcal{Y}$. We thus have $|U| \leq |C|/2$, $|B| \leq |C|/2$ and $U \in \mathcal{Y}$ (if $U \neq \emptyset$) by design. From Invariant 2 we further get that all $X \in \mathcal{X}$ have size $|X| \leq z$ at the beginning of this iteration of the algorithm. Removing the nodes of $B \cup U$ from every set in $\mathcal{X}$ can only make existing sets in the partition $\mathcal{X}$ smaller and the two new sets of the resulting partition, B and U , have size at most $|C|/2$ as shown above. Hence, $|Y| \leq \max\{z, |C|/2\}$ holds for all $Y \in \mathcal{Y}$.

Property 1 of **PARTITIONCLUSTER** follows directly, as the cut $E(U, C \setminus U)$ is $1/\phi$ -border routable through U with congestion at most 2 by **Property (c)** of **TwoWayTrim**.

For **Property 2**, observe that we called **SPARSESTCUTAPX** with the sparsity parameter $\phi/20$. **Claim 5** then shows that **Req. (b)** of **TwoWayTrim** is satisfied with $\delta = 1/(20q^*)$. By **Property (d)** of **TwoWayTrim** we thus get that $G[A \cup B] = G[C \setminus U]$ is $\frac{1}{500}\phi/q^* \cdot \deg_{\partial\mathcal{Y}}$ -expanding, as desired, where we use that $\deg_{\partial\mathcal{Y}}|_{A \cup B} \leq \deg_{E(B, V \setminus B)}|_{A \cup B} + \pi|_A$.

□

To establish the correctness of **PARTITIONCLUSTER** if the algorithm returns from **Case 2b**, we use the following properties of fair cuts. The proof of a more general version of this claim can be found in the full version.

CLAIM 6. *If U is an α -fair (s, t) -cut in $G = (V, E, \text{cap})$, then*

- (1) $\text{cap}(U, V \setminus U) + t(U) \leq \alpha \cdot s(U)$, and
- (2) *there is a $(s|_{V \setminus U} + \deg_{E(U, V \setminus U)}, t')$ flow in $G[V \setminus U]$ with congestion α for some t' with $0 \leq t' \leq \alpha \cdot t|_{V \setminus U}$.*

If the algorithm returns from **Case 2b**, then the returned set U is non-empty and fulfills the “balance”-condition (the “either” case) of **Property 2** in **PARTITIONCLUSTER**.

LEMMA 7. *If **PARTITIONCLUSTER** returns from **Case 2b**, then it returns a correct (balanced) bad child, i.e., **Property 1** and **Property 2** of **PARTITIONCLUSTER** are fulfilled.*

PROOF. In this case, the algorithm outputs $U = T'$ alongside the partition $\mathcal{Y}$. We have $T' \subseteq T$ and by **Invariant 1**, $|T| \leq |C|/2$, so $|U| \leq |C|/2$. Also, we have $U \in \mathcal{Y}$ (if $U \neq \emptyset$) by construction of $\mathcal{Y}$. Analogous to the argument in the proof of **Lemma 6**, we have that all $Y \in \mathcal{Y}$ have $|Y| \leq \max\{z, |C|/2\}$ since $|T'| \leq |C|/2$.

Recall that $T' = T \setminus X$, where X is a 2-fair (s, t) -cut in $G[T]$ for $s = \deg_{E(T, C \setminus T)}$ and $t = \frac{1}{2}\phi \deg_{\partial C}|_T$. Thus, by **Claim 6** there exists a $(\deg_{E(T', X)}|_{T'} + \deg_{E(T, C \setminus T)}|_{T'}, t')$ -flow g through $G[T']$ for some non-negative $t' \leq 2t = \phi \deg_{E(T, V \setminus C)}|_{T'}$. Note that $\deg_{E(T, C \setminus T)}|_{T'} + \deg_{E(T', X)}|_{T'} = \deg_{E(T', C \setminus T')}|_{T'}$, i.e., g is a $(\deg_{E(T', C \setminus T')}|_{T'}, t')$ flow through $G[T']$. Thus, $E(T', C \setminus T')$ is $1/\phi$ -border routable through T' with congestion 2, i.e., **Property 1** holds for $U = T'$.

We prove **Property 2** of **PARTITIONCLUSTER** holds by showing the “either”-case, namely we prove that $\deg_{\partial\mathcal{Y}}(U) \geq \tau^*/20 \cdot \deg_{\partial\mathcal{Y}}(C)$ and $\deg_{\partial\mathcal{Y}}(C) \leq \deg_{\partial\mathcal{X}}(C) + 2 \text{cap}(U, C \setminus U)$. We begin with the first statement. By construction, $T' \in \mathcal{Y}$, so we have

$$\begin{aligned} \deg_{\partial\mathcal{Y}}(U) &= \deg_{\partial\mathcal{Y}}(T') \geq \text{cap}(T', V \setminus C) \\ &= \text{cap}(T, V \setminus C) - \text{cap}(T \setminus T', V \setminus C). \end{aligned}$$

Since $X = T \setminus T'$ is a 2-fair (s, t) -cut in $G[T]$ we get from **Claim 6** that $\text{cap}(T \setminus T', V \setminus C) = \text{cap}(X, V \setminus C) = \frac{2}{\phi}t(X) \leq \frac{4}{\phi}s(T) = \frac{4}{\phi}\text{cap}(T, C \setminus T)$. In addition, T satisfies **Property 3** of **Invariant 1**, which gives $\frac{4}{\phi} \cdot \text{cap}(T, C \setminus T) \leq \frac{2}{5} \cdot \pi(T)$. Thus, $\text{cap}(T \setminus T', V \setminus C) \leq \frac{2}{5} \cdot \pi(T)$.

Note that the border edges of T must have a large capacity, namely must fulfill $\text{cap}(T, V \setminus C) = \deg_{E(C, V \setminus C)}(T) > \pi(T)/2$ because the algorithm reached **Case 2b**. With the help of these inequalities we get

$$\begin{aligned} \deg_{\partial\mathcal{Y}}(T') &\geq \text{cap}(T, V \setminus C) - \text{cap}(X, V \setminus C) \\ &\geq \frac{1}{2}\pi(T) - \frac{2}{5}\pi(T) \geq \tau^*/10 \cdot \pi(C), \end{aligned}$$

where the last inequality follows from **Property 2** of **Invariant 1**. Next, observe that

$$\begin{aligned} \deg_{\partial\mathcal{Y}}(C) &\leq \pi(C) - \pi(T') + 2 \text{cap}(T', C \setminus T') + \text{cap}(T', V \setminus C) \\ &\leq \pi(C) + 2 \text{cap}(T', C \setminus T') \\ &\leq \pi(C) + 2\phi/5 \cdot \pi(T) \leq 2\pi(C), \end{aligned}$$

where the first inequality holds because $\mathcal{Y}$ is obtained by fusing T' in the current partition $\mathcal{X}$ and $\pi = \deg_{\partial\mathcal{X}}$. This fuse operation can be viewed as first removing all π -weight from T' and then adding the weight of border edges, where edges in $E(T', C \setminus T')$ are counted twice. The second step follows from **Claim 6** for the 2-fair cut $T \setminus T'$ and **Property 3** of **Invariant 1**, which gives $\text{cap}(T', C \setminus T') \leq 2s(T \setminus T') \leq 2s(T) = 2 \text{cap}(T, C \setminus T) \leq \phi/5 \cdot \pi(T)$. In total, we thus have $\deg_{\partial\mathcal{Y}}(T') \geq \tau^*/20 \cdot \deg_{\partial\mathcal{Y}}(C)$, as desired.

To finish the proof of **Property 2** we need to show the second part of the “either”-case of **Property 2**, i.e., that $\deg_{\partial\mathcal{Y}}(C) \leq \deg_{\partial\mathcal{X}}(C) + 2 \text{cap}(U, C \setminus U)$, where U is the returned bad child. Recall that $\mathcal{X}$ denotes the partition that was initially supplied to **PARTITIONCLUSTER**. **Claim 7** guarantees that $\deg_{\partial\mathcal{X}}(C)$ does not increase during **PARTITIONCLUSTER**. The only increase that may happen is due to the last step when we fuse the cluster T' in the partition $\mathcal{X}'$ of the current iteration and return the resulting partition $\mathcal{Y}$. By **Claim 4**

$$\begin{aligned} \deg_{\partial\mathcal{Y}}(C) &\leq \deg_{\mathcal{X}'}(C) - \deg_{\mathcal{X}'}(T') + 2 \text{cap}(T', C \setminus T') + \text{cap}(T', V \setminus C) \\ &\leq \deg_{\mathcal{X}'}(C) + 2 \text{cap}(T', C \setminus T') \leq \deg_{\partial\mathcal{X}}(C) + 2 \text{cap}(U, C \setminus U), \end{aligned}$$

where $\deg_{\partial\mathcal{X}'}(C) \leq \deg_{\partial\mathcal{X}}(C)$ follows from **Claim 7** and $U = T'$. This finishes the proof of the lemma. □

Observe that the algorithm terminates if it reaches **Case 1a** or **Case 2b**. Hence, an iteration of **PARTITIONCLUSTER** is only non-terminating if it reaches **Case 2a**. We first show that the $\deg_{\partial\mathcal{X}}(C)$ -weight decreases significantly in every non-terminating iteration.

CLAIM 7. *A non-terminating iteration of **PARTITIONCLUSTER** reduces the weight function $\deg_{\partial\mathcal{X}}(C)$ by a factor of $(1 - \tau^*/4)$.*

PROOF. Let $\mathcal{X}_1$ be the partition at the beginning of the non-terminating iteration and let $\mathcal{X}_2$ be the partition at the end of that iteration. In such an iteration $\mathcal{X}_1$ is only modified by the fusing operation in **Case 2a**, i.e., $\mathcal{X}_2 = (\mathcal{X}_1 - T) \cup \{T\}$. Note that this is the only non-terminating case and it happens only if $\deg_{\partial C}(T) \leq \deg_{\partial\mathcal{X}_1}(T)/2$. The new partition $\mathcal{X}_2$ has

$$\begin{aligned} \deg_{\partial\mathcal{X}_2}(C) &\leq \deg_{\partial\mathcal{X}_1}(C) - \deg_{\partial\mathcal{X}_1}(T) + \deg_{\partial\mathcal{X}_1}(T) + 2 \text{cap}(T, C \setminus T) \\ &\leq \deg_{\partial\mathcal{X}_1}(C) - \deg_{\partial\mathcal{X}_1}(T) + \deg_{\partial\mathcal{X}_1}(T)/2 + \phi/5 \cdot \deg_{\partial\mathcal{X}_1}(T) \\ &= \deg_{\partial\mathcal{X}_1}(C) - (1/2 - \phi/5) \deg_{\partial\mathcal{X}_1}(T) \\ &\leq \deg_{\partial\mathcal{X}_1}(C) - (1/4) \cdot \deg_{\partial\mathcal{X}_1}(T) \\ &\leq (1 - \tau^*/4) \deg_{\partial\mathcal{X}_1}(C), \end{aligned}$$

The first inequality follows because we can view the fuse operation as first removing all weight from T , then adding the weight of edges in ∂C that are incident to T , and finally adding the weight of edges in $E(T, C \setminus T)$. Note that the weight of the latter edges has to be added twice for the two endpoints that share an edge. The second inequality follows due to the condition $\deg_{\partial C}(T) \leq$

$\deg_{\partial X_1}(T)/2$ and **Invariant 1, Property 3**. The third inequality uses $1/2 - \phi/5 \geq 1/4 \geq \tau^*/4$ as $\tau^* \leq \beta^* \leq 1/2$ and the last uses **Invariant 1, Property 2**. $\square$

LEMMA 8. *Given a cluster C with volume $m_C = \deg(C)$, and algorithms for **SPARSESTCUTAPX**, **TWOWAYTRIM**, and **FAIRCUTFLOW**, that run in time T_{sc} , T_{tw} and T_{fc} , respectively, the algorithm for **PARTITIONCLUSTER** runs in time $O(\log(m_C)/\tau^* \cdot (T_{sc}(m_C) + T_{tw}(m_C)) + T_{fc}(m_C, 2))$.*

PROOF. In every non-terminating iteration through the while loop $\deg_{\partial X}(C)$ reduces by a factor of $(1 - \tau^*/4)$. Hence, there can be at most $\log_{1/(1-\tau^*/4)} m_C = O(\log(m_C)/\tau^*)$ such iterations. In addition to these iterations there is at most one call to **FAIRCUTFLOW**. The remaining bookkeeping cost is at most linear in m_C and thus dominated by $T_{sc}(m_C)$. $\square$

Corollary 5. With the implementation of our building blocks the running time of **PARTITIONCLUSTER** is $O(\log^7(m_C) \cdot T_{fc}(m_C, 2)) = \tilde{O}(m_C)$ for a cluster C with volume $m_C = \deg(C)$.

Acknowledgements

We thank Evangelos Kosinas for helpful discussions on this topic.

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research

and innovation programme (MoDynStruct, No. 101019564)  and the Austrian Science Fund (FWF) grant DOI 10.55776/I5982. For open access purposes, the author has applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

This project has received funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 498605858.

References

- [1] Arpit Agarwal, Sanjeev Khanna, Huan Li, Prathamesh Patil, Chen Wang, Nathan White, and Peilin Zhong. 2024. Parallel Approximate Maximum Flows in Near-Linear Work and Polylogarithmic Depth. In *Proceedings of the 35th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 3997–4061. doi:10.1137/1.9781611977912.140
- [2] Daniel Agassy, Dani Dorfman, and Haim Kaplan. 2023. Expander Decomposition with Fewer Inter-Cluster Edges Using a Spectral Cut Player. In *Proceedings of the 50th International Colloquium on Automata, Languages and Programming (ICALP)*. 9:1–9:20. doi:10.4230/LIPIcs.ICALP.2023.9
- [3] Daniel Agassy, Dani Dorfman, and Haim Kaplan. 2025. Expander Decomposition for Non-Uniform Vertex Measures. arXiv:2510.23913 [cs.DS] appeared 29 Oct 2025.
- [4] Daniel Agassy, Dani Dorfman, and Haim Kaplan. 2025. Improved Tree Sparsifiers in Near-Linear Time. arXiv:2511.06574 [cs.DS] appeared 9 Nov 2025.
- [5] Konstantin Andreev, Charles Garrod, Bruce M. Maggs, and Adam Meyerson. 2009. Simultaneous Source Location. *ACM Transactions on Algorithms* 6, 1 (2009), 16:1–16:17. doi:10.1145/1644015.1644031
- [6] Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph (Seffi) Naor, and Roy Schwartz. 2011. Min-Max Graph Partitioning and Small Set Expansion. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS)*. 17–26. doi:10.1109/FOCS.2011.79
- [7] Marcin Bienkowski, Mirosław Korzeniowski, and Harald Räcke. 2003. A Practical Algorithm for Constructing Oblivious Routing Schemes. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 24–33. doi:10.1145/777412.777418
- [8] Chandra Chekuri, Sanjeev Khanna, and Bruce Shepherd. 2004. The All-or-Nothing Multicommodity Flow Problem. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*. 156–165. doi:10.1145/1007352.1007383
- [9] Roei Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph (Seffi) Naor. 2007. Cut Problems in Graphs with a Budget Constraint. *Journal of Discrete Algorithms* 5, 2 (2007), 262–279. doi:10.1016/j.jda.2006.05.002
- [10] L. R. Ford and D. R. Fulkerson. 1956. Maximal Flow Through a Network. *Canadian Journal of Mathematics* 8 (1956), 399–404. doi:10.4153/CJM-1956-045-5
- [11] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. 2021. The Expander Hierarchy and its Applications to Dynamic Graph Algorithms. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2212–2228. doi:10.1137/1.9781611976465.132
- [12] Kathrin Hanauer, Monika Henzinger, Robin Münk, Harald Räcke, and Maximilian Vötsch. 2024. Expander Hierarchies for Normalized Cuts on Graphs. In *KDD*. ACM, 1016–1027. doi:10.1145/3637528.3671978
- [13] Chris Harrelson, Kirsten Hildrum, and Satish B. Rao. 2003. A Polynomial-time Tree Decomposition to Minimize Congestion. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 34–43. doi:10.1145/777412.777419
- [14] Monika Henzinger, Robin Münk, and Harald Räcke. 2025. An Improved Quality Hierarchical Congestion Approximator in Near-Linear Time. arXiv:2511.03716 [cs.DS]
- [15] Rohit Khandekar, Guy Kortsarz, and Vahab Mirrokni. 2012. Advantage of Overlapping Clusters for Minimizing Conductance. In *Proceedings of the 13th Latin American Symposium on Theoretical Informatics (LATIN)*. 494–505. doi:10.1007/978-3-642-29344-3_42
- [16] Rohit Khandekar, Satish Rao, and Umesh Vazirani. 2006. Graph Partitioning using Single Commodity Flows. In *Proceedings of the 38th ACM Symposium on Theory of Computing (STOC)*. 385–390. doi:10.1145/1132516.1132574
- [17] Philip N. Klein, Serge A. Plotkin, and Satish B. Rao. 1993. Excluded Minors, Network Decomposition, and Multicommodity Flow. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*. 682–690. doi:10.1145/167088.167261
- [18] Jochen Könemann, Ojas Parekh, and Danny Segev. 2011. A Unified Approach to Approximating Partial Covering Problems. *Algorithmica* 59, 4 (2011), 489–509. doi:10.1007/s00453-009-9317-0 Also in *Proc. 14th ESA*, 2006, pp. 468–479.
- [19] Jason Li, Satish Rao, and Di Wang. 2025. Congestion-Approximators from the Bottom Up. In *Proceedings of the 36th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2111–2131. doi:10.1137/1.9781611978322.68
- [20] Aleksander Madry. 2010. Fast Approximation Algorithms for Cut-based Problems in Undirected Graphs. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS)*. 245–254. doi:10.1109/FOCS.2010.30
- [21] Lorenzo Orecchia, Leonard J. Schulman, Umesh V. Vazirani, and Nisheeth K. Vishnoi. 2008. On Partitioning Graphs via Single Commodity Flows. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*. 461–470. doi:10.1145/1374376.1374442
- [22] Richard Peng. 2016. Approximate Undirected Maximum Flows in $O(m \text{ polylog } n)$ Time. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1862–1867. doi:10.1137/1.9781611974331.ch130
- [23] Harald Räcke. 2002. Minimizing Congestion in General Networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*. 43–52. doi:10.1109/SFCS.2002.1181881
- [24] Harald Räcke. 2008. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. In *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC)*. 255–264. doi:10.1145/1374376.1374415
- [25] Harald Räcke and Chintan Shah. 2014. Improved Guarantees for Tree Cut Sparsifiers. In *Proceedings of the 22nd European Symposium on Algorithms (ESA)*. 774–785. doi:10.1007/978-3-662-44777-2_64
- [26] Harald Räcke, Chintan Shah, and Hanjo Täubig. 2014. Computing Cut-Based Hierarchical Decompositions in Almost Linear Time. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 227–238. doi:10.1137/1.9781611973402.17
- [27] Thatchaphol Saranurak and Di Wang. 2019. Expander Decomposition and Pruning: Faster, Stronger, and Simpler. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2616–2635. doi:10.5555/3310435.3310597
- [28] Jonah Sherman. 2013. Nearly Maximum Flows in Nearly Linear Time. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*. 263–269. doi:10.1109/FOCS.2013.36

Received 2025-11-04; accepted 2026-02-01