Faster Weak Expander Decompositions and Approximate Max Flow

Henry Fleischmann
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

George Z. Li
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Jason Li
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We give faster algorithms for weak expander decompositions and approximate max flow on undirected graphs. First, we show that it is possible to "warm start" the cut-matching game when computing weak expander decompositions, avoiding the cost of the recursion depth. Our algorithm is also flexible enough to support weaker flow subroutines than previous algorithms.

Our second contribution is to streamline the recent non-recursive approximate max flow algorithm of Li, Rao, and Wang (SODA, 2025) and adapt their framework to use our new weak expander decomposition primitive. Consequently, we give an approximate max flow algorithm within a few logarithmic factors of the limit of expander decomposition-based approaches.

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases max flow, expander decompositions, congestion approximators, cut-matching game

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.91

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2511.02943`

Funding This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE2140739. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

In the maximum flow problem, we are given a set of vertex demands, where each vertex is required to send or receive a certain amount of flow, and the goal is to route these demands while minimizing the maximum congestion along any edge. It is one of the oldest problems in theoretical computer science [10], with connections to other famous problems including minimum cut, bipartite matching, and Gomory-Hu trees [13]. Modern algorithmic techniques have produced exciting breakthroughs in both the exact directed setting [9, 24, 6] and the approximate undirected setting [20, 32, 18, 33, 1]. These techniques include the interior point method from continuous optimization [9, 24, 6], electrical flows and Laplacian solvers [35, 20, 24, 12], expander decompositions and congestion approximators [32, 30], and dynamic data structures [6].

On the other hand, despite the rapid advancement of modern flow algorithms, progress towards *understanding* max flow, especially its underlying structural properties, has arguably lagged behind:


1. The first $(1-\epsilon)$-approximate max flow algorithm on undirected graphs runs in $O(m \log^{41} n)$ time even for constant $\epsilon > 0$ [27]. The algorithm is fairly complex, recursively alternating between multiple different problems, and it took a decade before the first non-recursive algorithm was developed [23], improving the runtime to $O(m \log^{17} n)$.[1]
2. While the exact max flow algorithms based on interior point methods are impressive, they do not shed light on the *combinatorial* structure of max flow. In response, a recent trend of studying combinatorial max flow has emerged [7, 8, 4, 3], obtaining augmenting path-style algorithms that are more faithful to traditional approaches.

This paper is dedicated to refining our understanding of approximate max flow in undirected graphs. Our starting point is the recent non-recursive algorithm for approximate max flow [23], which computes a hierarchy of so-called *weak expander decompositions*, using previously computed levels of the hierarchy to build the next level. From this hierarchy, a *congestion approximator* is extracted and used in Sherman's framework [33] to obtain the desired approximate max flow. However, [23] do not state an explicit running time, since the weak expander hierarchy construction requires calls to *fair cut/flow* [22], which introduce a large running time overhead.

The contribution of this paper is twofold:

1. First, we develop a faster weak expander decomposition algorithm by "warm starting" the cut-matching game [19] whenever a sparse cut is found. This algorithm can be implemented using $O(\log^2 n)$ calls to max flow, compared to $O(\log^3 n)$ for the standard weak expander decomposition implementation [31], and may be of independent interest.
2. Next, we streamline the framework of [23] to obtain a non-recursive approximate max flow algorithm with an improved running time of $O(m \log^9 n \log\log n)$. In particular, we are able to implement our weak expander decomposition using approximate max flow, compared to prior algorithms which require fair cut/flow or similarly strong guarantees [31, 23]. Similar to [23], these max flow calls are specialized enough to be solvable using the existing levels of the hierarchy. However, a weaker flow oracle introduces a number of technical difficulties which we discuss in the technical overview.

▶ **Theorem 1** (Informal version of Corollary 19). *Given an undirected graph with integral and polynomially-bounded edge capacities, there is an $O(m \log^9 n)$ time algorithm to construct a congestion-approximator with quality $O(\log^5 n)$. Together with Sherman's framework [33], we obtain an $(1-\epsilon)$-approximate max flow algorithm in time $O(m \log^9 n \log\log n + \epsilon^{-1} m \log^6 n)$.*

While the logarithmic exponent of 9 is too large to be practical, we remark that expander decomposition-based algorithms have historically led to similarly large constants. For example, the state of the art (strong) ϕ-expander decomposition [2] deletes $O(\phi \log^2 n)$ fraction of edges and runs in time $O(m \log^4 n/\phi + m \log^7 m)$ on capacitated graphs; to delete a constant fraction of edges, we require $\phi \approx 1/\log^2 n$ which results in $O(m \log^7 n)$ time. Even our faster implementation of weak expander decomposition runs in $O(m \log^4 n)$ time in the most ideal setting. Therefore, our max flow algorithm is within a few logarithmic factors of the limit to any expander decomposition-based approaches, and substantial future improvements will require either breakthroughs in computing expander decompositions, or bypassing expander decompositions altogether.

[1] We suppress $O(\log\log n)$ factors in these running times for simplicity.

Further related work

Our result builds on a long line of work on efficiently constructing a small collection of cuts, called congestion approximators, which approximately capture the congestion required to route any feasible demand [28, 5, 15, 29, 30, 26, 27, 14, 1, 16]. To construct an α-congestion approximator for $\alpha = \operatorname{poly}\log n$, these algorithms required (approximate) max flow as a subroutine. In a breakthrough work, [32] showed that an α-congestion approximator could also be used to compute $(1-\epsilon)$-approximate max flow. There is a chicken-and-egg problem here as congestion-approximators and approximate max-flow need each other, which [27] resolved using a costly recursion combined with ultra-sparsifiers. Recent work [23] was able to avoid this recursion by building a congestion-approximator bottom-up without requiring an approximate max-flow oracle, which is the focus of our work.

A stronger cut player?

Our expander decomposition algorithm is adapted from the cut-matching game from [19], despite the fact that cut-matching games with better parameters have since been developed. Most notably, the cut-matching game of [25] uses a stronger spectral cut player to save an $O(\log n)$ factor in their expansion guarantees. However, our warm starting analysis relies on a simple property of the potential function used in [19]: if we partition a set $S = S_1 \sqcup S_2$, then the sum of the induced potentials on S_1 and S_2 is at most the potential on S. The same property is not obviously true of the matrix-based potential function of [25].

Our work also relies essentially on the "nonstop" variant of KRV used in [30, 31] to give an efficient expander decomposition algorithm. Until the recent works of [2, 16], such a nonstop version of [25] was unknown. Given this development, we believe it may be a promising direction to adapt our warm starting analysis to this OSVV-based nonstop cut-matching game. We leave this as an open problem in our work.

2 Technical Overview

Faster weak expander decompositions

All known algorithms for computing expander decompositions in near-linear time rely on the cut-matching game. Our first main technical contribution is a faster algorithm for computing weak expander decompositions. We do this by observing that it is possible to "warm start" our recursive instances of the cut-matching game. Importantly for our application to approximate max flow, our algorithm is robust enough to support general vertex weights and to implement the matching steps using approximate max flow oracles.

We now describe the techniques in more detail. In the standard cut-matching game (on an unweighted graph), we have $T = \Theta(\log^2 n)$ rounds in total. In each round, the cut player finds two disjoint sets L_A and R_A. The matching player then tries to route a flow from L_A to R_A, implicitly defining a matching between the sets. If at any iteration the matching player fails to route the flow, the cut certifying infeasibility of the flow is a sparse cut, showing that the graph is not an expander. Otherwise, if all matching step flows are feasible, the cut player is defined so that the union of the matchings found in the T iterations is itself an expander. Combined with the fact that the matchings embed into the original graph with low congestion, this proves that G must be an expander.

In the non-stop version of the cut-matching game [30, 31], when the matching player fails to route the flow from L_A to R_A, thus finding a sparse cut S, the algorithm does not (necessarily) immediately terminate. Instead, the cut-matching game continues on $V \setminus S$. More generally, let A be the current set on which the cut-matching game is being played;

when the matching player finds a cut S, the algorithm continues on $A \setminus S$. It can be shown that after T iterations, the remaining set A is a near-expander in G, meaning that its degree vertex weighting mixes in G (but possibly not $G[A]$) with low congestion.

To convert the non-stop cut-matching game into a weak expander decomposition algorithm, [11] adds an *early termination condition*: if $\mathrm{vol}(A) \leq 99\,\mathrm{vol}(V)/100$ at any point, the non-stop cut-matching game terminates and recurses on A and $V \setminus A$. Otherwise, if the cut-matching game terminates in certifying that A is a near-expander, we recurse onto $V \setminus A$ if it is non-empty. If A is certified as a near-expander in some iteration, we then know that $\mathrm{vol}(V \setminus A) \leq \mathrm{vol}(V)/100$ so the recursive call decreases by a constant factor in size. If we reach the early termination condition, then $\mathrm{vol}(A) \leq 99\,\mathrm{vol}(V)/100$ and we have the additional guarantee that $V \setminus A$ is partitioned by sparse cuts into subsets of at most $2/3$ of the volume each. Hence, the recursive calls also decrease in size by a constant factor in this case. As a result, the recursive depth is at most $O(\log n)$, so we can compute a weak expander decomposition in $O(\log^3 n)$ iterations of the cut-matching game.

We give a new weak expander decomposition algorithm which only uses $O(\log^2 n)$ iterations of the cut-matching game. Let A again be the current set of vertices. If we find a matching successfully, then we continue the cut-matching game on A. Otherwise, we find a sparse cut $S \subseteq A$. In the previous algorithm, we would only continue on $A \setminus S$, delaying continuing on S until reaching the early termination condition or certifying that some subset of $A \setminus S$ is a near-expander in G. Our main observation is that we can continue the cut-matching game on both $A \setminus S$ and S simultaneously without a loss in the runtime. This amounts to "warm-starting" on S. Slightly more formally, we maintain a partition of V into k_t sets $\mathcal{A}_t = \{A_1, \ldots, A_{k_t}\}$ at each iteration t. At the beginning of the algorithm, we set $\mathcal{A}_0 = \{V\}$ and at each iteration t, we run the cut-matching game on each $A_i \in \mathcal{A}_t$ simultaneously. When we find some cut $S_i \subseteq A_i$, we add S_i (if nonempty) and $A_i \setminus S_i$ to $\mathcal{A}_{t+1}$. After T rounds, we will certify that each component in $\mathcal{A}_T$ is a near-expander. We remark that warm-starting crucially uses the fact that we are ultimately constructing a weak expander decomposition, not a strong one. Indeed, the matching embeddings from steps prior to restricting to a subgraph (from finding a cut) are not guaranteed to embed into our current subgraph. This is fine for certifying near-expansion but too weak a guarantee for strong expansion.

Importantly for our application to approximate max flow, this algorithm still reveals sufficient structure when implementing the matching steps with an approximate max flow oracle. To this end, we give our algorithm in two steps. The first step is to compute a weak expander decomposition where there can be a small "deleted," non-expanding portion of the input vertex weighting (Section 5.1). The non-deleted portion is certified to mix simultaneously (i.e., each component expands with respect to the non-deleted portion of the vertex weighting) and there are guaranteed to be few intercluster edges, as usual. Then, in the second step (Section 5.2), we attempt to *graft* the demand deleted in each cluster back into the cluster, as in [11]. After this step, every expanding cluster will not have any deleted demand, and nearly all demand will belong to an expanding cluster. We state an informal version of our result in Theorem 2.

▶ **Theorem 2** (Informal version of Theorem 15). *Suppose we have $G = (V, E, \mathbf{c})$ with integer edge capacities at most $\mathrm{poly}(n)$. In addition, suppose we have vertex weighting $\mathbf{d} \in \mathbb{Z}^V_{\geq 0}$, expansion parameter $\phi > 0$, and a suitable approximate max flow oracle running in time $F(n, m, \varepsilon)$. Then, there is an algorithm computing a partition $\mathcal{A} = \mathcal{A}^\circ \sqcup \mathcal{A}^\times$ of V with the following properties:*

1. *The algorithm runs in time $O(F(n, m, \varepsilon) \log^2 n + m \log^4 n)$.*
2. $\mathbf{d}(\bigcup_{A \in \mathcal{A}^\times} A) = O((\varepsilon \log^2 n + \phi \log n)\,\mathbf{d}(V))$.
3. *The total capacity of edges cut by $\mathcal{A}$ is at most $O(\phi\,\mathbf{d}(V) \log n)$.*
4. *Each $A \in \mathcal{A}^\circ$ is a $(\phi/\log^2 n, \mathbf{d})$-near-expander in G.*

Importantly for our applications, we actually obtain a stronger simultaneous mixing expansion property instead of (4), but we omit that here for simplicity (see Theorem 15). Also note that, unlike standard weak expander decompositions, our result does not exactly decompose all vertices into near-expanders (i.e., usual decompositions would get $\mathcal{A}^{\times} = \emptyset$ or the guarantee of (2) to be 0). However, this relaxation is critical for obtaining such a result and still suffices for some important applications of weak expander decompositions. Indeed, the relaxation of (2) suffices for our application to constructing congestion-approximators and approximate max flow, as we discuss next.

Faster congestion-approximators

Recall that a laminar family $\mathcal{C}$ of subsets of V forms an α-congestion-approximator if for every vertex demand, the minimum ratio over cuts $C \in \mathcal{C}$ between the capacity of the cut and the demand crossing the cut is an α-approximation to the optimal congestion of any flow routing the demand. Our goal is to construct α-congestion-approximators faster and with smaller α. To discuss our improvement over previous work, we restate the informal Theorem 2.1 from [23], which gave a novel approach for constructing congestion-approximators.

▶ **Theorem 3** (Theorem 2.1 of [23]). *Consider a capacitated graph $G = (V, E, \mathbf{c})$, and let $\alpha \geq 1$ and $\beta \geq 1$ be parameters. Suppose there exist partitions $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_L$ of V such that*

1. *$\mathcal{P}_1$ is the partition $\{\{v\} : v \in V\}$ of singleton clusters, and $\mathcal{P}_L$ is the partition $\{V\}$ with a single cluster.*
2. *For each $i \in [L-1]$, for each $C \in \mathcal{P}_{i+1}$, the intercluster edges of $\mathcal{P}_i$ internal to C along with the boundary edges of C mix in the graph G. Moreover, the mixings over all the clusters $C \in \mathcal{P}_{i+1}$ have congestion α simultaneously.*
3. *For each $i \in [L-1]$, there is a flow in G with congestion β such that each intercluster edge of $\mathcal{P}_{i+1}$ sends its capacity in flow, and each intercluster edge of $\mathcal{P}_i$ receives half its capacity in flow.*

For each $i \in [L]$, let partition $\mathcal{R}_{\geq i}$ be the common refinement of partitions $\mathcal{P}_i, \mathcal{P}_{i+1}, \ldots, \mathcal{P}_L$, i.e.,

$$\mathcal{R}_{\geq i} = \{C_i \cap \cdots \cap C_L : C_i \in \mathcal{P}_i, \ldots, C_L \in \mathcal{P}_L, C_i \cap \cdots \cap C_L \neq \emptyset\}.$$

Then, their union $\mathcal{C} = \bigcup_{i \in [L]} \mathcal{R}_{\geq i}$ is a congestion-approximator with quality $16\alpha\beta L^2$.

The partitions $\mathcal{P}_i$ described in the theorem essentially form a weak expander hierarchy, where each level is essentially a (boundary-linked) weak expander decomposition. Using the existence of routings guaranteed by (2) and (3), [23] show that a demand respecting the congestion-approximator can be iteratively routed. They then show that this weak expander hierarchy can be constructed using existing tools for constructing expander decompositions [31, 22]. By doing this, they construct a sequence of partition $\mathcal{P}_1, \ldots, \mathcal{P}_L$ satisfying properties (1), (2), and (3) with parameters $\alpha = O(\log^5 n)$ and $\beta = O(\log^3 n)$.

Our main observation is that we do not need the full power of a weak expander decomposition at each level in order to show the existence of this routing. If a small constant fraction of the vertices (measured in terms of volume in the subgraph) do not have the expander mixing property, this is still sufficient to show that the congestion-approximator routing exists. Specifically, we relax conditions (2) and (3) on the partitions $\mathcal{P}_i$ to allow for a small constant fraction of edges to not participate in the routings at level i and be instead be handled at level $i+1$. More formally, we let $\mathcal{P}_1, \ldots, \mathcal{P}_L$ be partitions of subsets $V_1, \ldots, V_L \subseteq V$. We only require the mixing properties (2) and (3) on V_{i+1} for each $i \in [L-1]$, so we should

think of $\mathcal{P}_i$ as a weak expander decomposition of V_i. To extend the partitions $\mathcal{P}_i$ of V_i to a partition $\overline{\mathcal{P}}_i$ of V, we define $\mathcal{Q}_i$ to be the induced partition from the previous level $\overline{\mathcal{P}}_{i-1}$ on $V \setminus V_i$. That is, we define

$$Q_i = \{C \cap (V \setminus V_i) : C \in \overline{\mathcal{P}}_{i-1}, C \cap (V \setminus V_i) \neq \emptyset\}.$$

Then we can define $\overline{\mathcal{P}}_i = \mathcal{P}_i \cup \mathcal{Q}_i$. Intuitively, when we only have a partition on V_i, we are giving up on routing the demand from the intercluster edges from $\overline{\mathcal{P}}_{i-1}$ in $V \setminus V_i$ and dealing with it at a higher level. In order to move it to the higher level, we include it in $\overline{\mathcal{P}}_i$ through the definition of $\mathcal{Q}_i$.

We now state a morally true version of our relaxed conditions for constructing congestion-approximators.

▶ **Theorem 4** (Informal version of Theorem 16). *Consider a capacitated graph $G = (V, E, \mathbf{c})$, and let $\alpha \geq 1$ and $\beta \geq 1$ be parameters. Let $\mathcal{P}_1, \ldots, \mathcal{P}_L$ be partitions of $V_1, \ldots, V_L$, respectively and extend these to partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_L$ as described above. Suppose the partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_L$ satisfy:*

1. *$\overline{\mathcal{P}}_1$ is the partition $\{\{v\} : v \in V\}$ of singleton clusters and $\overline{\mathcal{P}}_L$ is the partition $\{\{V\}\}$ with a single cluster.*
2. *For each $i \in [L-1]$, for each $C \in \mathcal{P}_{i+1}$, the intercluster edges of $\overline{\mathcal{P}}_i$ internal to C along with the boundary edges of C mix in the graph G. Moreover, the mixings over all the clusters $C \in \mathcal{P}_{i+1}$ have congestion α simultaneously.*
3. *For each $i \in [L-1]$, there is a flow in G with congestion β such that each intercluster edge of $\mathcal{P}_{i+1}$ sends its capacity in flow, and each intercluster edge of $\overline{\mathcal{P}}_i$ receives at most a quarter its capacity in flow.*

For each $i \in [L]$, let partition $\mathcal{R}_{\geq i}$ be the common refinement of partitions $\overline{\mathcal{P}}_i, \overline{\mathcal{P}}_{i+1}, \ldots, \overline{\mathcal{P}}_L$, i.e.,

$$\mathcal{R}_{\geq i} = \{C_i \cap \cdots \cap C_L : C_i \in \overline{\mathcal{P}}_i, \ldots, C_L \in \overline{\mathcal{P}}_L, C_i \cap \cdots \cap C_L \neq \emptyset\}.$$

Then, their union $\mathcal{C} = \bigcup_{i \in [L]} \mathcal{R}_{\geq i}$ is a congestion-approximator with quality $48\alpha\beta L^2$.

The first advantage of this relaxation is that our algorithm for constructing the partitions $\mathcal{P}_1, \ldots, \mathcal{P}_L$ is faster. In particular, we can use approximate max flow algorithms to implement the cut-matching game. This may cause some nodes to be "deleted," as described in the previous subsection, but this is okay for us since we only need expander mixing guarantees on a (large) constant fraction of the vertices for property (2). In contrast, [23] used a fair-cuts algorithm [22] to implement the same step in their paper, which incurred several additional log factors in their runtime.

The second advantage is for obtaining smaller β. In [23], they prove that the flow from $\partial\mathcal{P}_{i+1}$ to $\partial\mathcal{P}_i$ exists using the boundary-linkedness property of the expander decompositions. This approach naturally suffers from $\beta = \Omega(\log^3 n)$ because the flows guaranteed by the expander decomposition have congestion $\Omega(\log^3 n)$. Instead, our approach is to directly attempt to send flow from $\partial\mathcal{P}_{i+1}$ to $\partial\mathcal{P}_i$ at each level. Using a max-flow/min-cut algorithm, we will find a (possibly empty) cut and a flow which saturates the cut. If we simply remove the vertices which are cut out, we have that in the remainder of the graph, there is a flow from $\partial\mathcal{P}_{i+1}$ to $\partial\mathcal{P}_i$ (since the flow saturates the cut). This gives us the desired $\beta = O(1)$, and also crucially uses our relaxation of properties (2) and (3).

Finally, we note that we obtain a smaller $\alpha = O(\log^3 n)$ in our construction. This is because we construct a weak expander decomposition on each level, which we observed is sufficient for the simultaneous mixing guarantees required by property (2). This enables us to avoid the costly trimming step used in [23], and also enables our speedup using warm-starting, which no longer helps for strong expander decompositions.

The approximate max flow algorithm

We apply Sherman's framework [33] to convert a congestion-approximator into an approximate max flow algorithm. Our approach for constructing a congestion-approximator is to construct a weak expander hierarchy using the cut-matching game. In implementing the cut-matching game, we need to solve flow problems (approximately), and we do this using the previous layers of the hierarchy as a "pseudo"-congestion-approximator. Our (pseudo)-congestion-approximators have quality $\alpha\beta L^2 = O(\log^5 n)$, giving us an $O(m \log^6 n \log\log n)$ time algorithm for solving the flow problems in the cut-matching game. Our improved weak expander decomposition algorithm takes $T = O(\log^2 n)$ rounds to obtain a full weak expander decomposition. Finally, there are $L = O(\log n)$ layers of the hierarchy, totalling $O(m \log^9 n \log\log n)$ runtime.

3 Organization of the Paper

In Section 5 we give a faster algorithm for weak expander decompositions. This algorithm supports implementing its flow subroutines with weaker than usual properties, which are described in Oracle 1 and Oracle 2. In Section 6 and Section 7 we give a faster algorithm for computing a congestion-approximator from the bottom up, using the faster weak expander decomposition as a critical subroutine. To do this, we show how to efficiently implement Oracle 1 and Oracle 2. As a direct consequence of our new algorithm for constructing a congestion-approximator, we obtain the fastest known approximate max flow algorithm. We defer all proofs to the full online version of the paper.

4 Preliminaries

Functions

For two functions $f, g : X \to \mathbb{R}$ let $f \leq g$ denote that, for all $x \in X$, $f(x) \leq g(x)$. We write $\mathrm{supp}(f)$ to denote the subset of X on which f takes nonzero values. For $S \subseteq X$ and S finite, we also write $f(S)$ as shorthand for $\sum_{x \in S} f(x)$. We use $f|_S$ to mean the restriction of f to S. When clear from context, we sometimes abuse notation and use $f|_S$ to denote the function f on the same domain but set equal to 0 outside of S. We also use all of the above notation for vectors, interpreting those vectors as functions. We often bold vectors to distinguish them from scalars (e.g., write $\mathbf{d} \in \mathbb{R}^V$).

Graphs

We consider capacitated (weighted) graphs $G = (V, E, \mathbf{c})$ where $\mathbf{c} \in [1, W] \cap \mathbb{Z}$. Unless otherwise specified, we use n to denote the order of G and m to denote its size. Sometimes we write V_G (or $V(G)$), E_G (or $E(G)$), and $\mathbf{c}_G$ to clarify that they are the parameters of the graph G. For $S \subseteq V$, denote the induced subgraph of G as $G[S]$. In other words, $G[S]$ is the subgraph of G formed by retaining exactly vertices in S and edges between vertices in S.

Given a partition $\mathcal{A}$ of V, we write $\partial\mathcal{A}$ to denote the set of intercluster edges in G. When $\mathcal{A}$ is just a single cut $(S, V \setminus S)$, we sometimes write ∂S instead. We often consider $\partial\mathcal{A}$ as an edge subgraph of G. We also use the notation $\delta\mathcal{A} = \mathbf{c}(\partial\mathcal{A})$ and $\delta S = \mathbf{c}(\partial S)$ (or $\delta(\mathcal{A})$ and $\delta(S)$) as shorthand denoting the total capacity of intercluster or cut edges. For $u \in V$, we denote the (weighted) degree of u in G as $\deg_G(u) = \delta_G(\{u\})$. We will often also consider $\deg_{\partial\mathcal{A}}(u) = \delta_{\partial\mathcal{A}}(\{u\})$ which treats $\partial\mathcal{A}$ as an edge subgraph of G. Finally, throughout we consider vertex weights $\mathbf{d} \in \mathbb{Z}^V_{\geq 0}$, with the most common weight function being $\mathbf{d} = \deg_G$ or $\deg_H$ for H a subgraph of G.

Flow

A *demand* is a vector $\mathbf{b} \in \mathbb{R}^V$ whose entries sum to 0. We say a flow $f : E \to \mathbb{R}$ routes a demand $\mathbf{b}$ if for each $v \in V$ the net flow at v in f is $\mathbf{b}(v)$. We say that f has *congestion* κ if the flow through any edge in f is at most κ times its capacity. Given a flow f, a *path decomposition* of f is a collection of weighted paths in G such that, for each $(u, v) \in E$, the flow from u to v in f is the sum of weights of paths containing the edge from u to v in the path decomposition.

Expansion

Let $G = (V, E, \mathbf{c})$ be a capacitated graph, and let $\mathbf{d} \in \mathbb{R}^V_{\geq 0}$ be a vertex weighting. Let $S \subseteq V$. Then, the *conductance of S in G with respect to* $\mathbf{d}$ is

$$\Phi_{G,\mathbf{d}}(S) = \frac{\delta_G(S)}{\min(\mathbf{d}(S), \mathbf{d}(V \setminus S))}.$$

We say that a cut S is *ϕ-sparse* (in G with respect to $\mathbf{d}$) if $\Phi_{G,\mathbf{d}}(S) \leq \phi$. We say that G is a $(\phi, \mathbf{d})$-*expander* if, for all $S \subseteq V$, we have $\Phi_{G,\mathbf{d}}(S) \geq \phi$. For $A \subseteq V$, we say that A is *ϕ-nearly $\mathbf{d}$-expanding* in G (or A is a $(\phi, \mathbf{d})$-*near-expander* in G) if, for all $S \subseteq A$, we have

$$\frac{\delta_G(S)}{\min(\mathbf{d}(S), \mathbf{d}(A \setminus S))} \geq \phi.$$

Note that if A is ϕ-nearly $\mathbf{d}$-expanding in G, then the same holds for all $A' \subseteq A$, since the denominators of the relevant expressions only decrease. When $\mathbf{d} = \deg_G$ or $\mathbf{d}$ is clear from context we say G is a ϕ-expander (respectively, near-expander).

We can also define expansion with respect to flows. We say that a vertex weighting $\mathbf{d} \in \mathbb{R}^V_{\geq 0}$ *mixes in G with congestion* κ if, for all demands $\mathbf{b} \in \mathbb{R}^V$ with $|\mathbf{b}| \leq \mathbf{d}$, we have that $\mathbf{b}$ is routable in G with congestion at most κ. In fact, $\mathbf{d}$ mixes in G with congestion $1/\phi$ if and only if G is a $(\phi, \mathbf{d})$-expander. Note that while $\mathbf{d}|_A$ mixing in G with congestion κ implies that A is a $(\phi, \mathbf{d})$-near-expander in G, the converse does not hold in general. Flow-based expansion is stronger than cut-based expansion for near-expanders.

Sometimes we require an even stronger notion of expansion with respect to multi-commodity flows. We say that a collection of vertex weights $\{\mathbf{d}_i : i \in I\}$ *mixes simultaneously* in G with congestion κ if, for all tuples of demands $(\mathbf{b}_i)_{i \in I}$ with each $\mathbf{b}_i \in \mathbb{R}^V$ satisfying $|\mathbf{b}_i| \leq \mathbf{d}_i$, there exists a multicommodity flow $\mathbf{F}$ with one commodity per demand which routes all $\mathbf{b}_i$ and has total congestion κ.

Congestion-approximators

Given a graph $G = (V, E, \mathbf{c})$, a *congestion-approximator* $\mathcal{C}$ of *quality* α is a family of subsets of V such that, for any demand $\mathbf{b}$ satisfying $|\mathbf{b}(C)| \leq \delta_G(C)$ for all $C \in \mathcal{C}$, there is a flow routing demand $\mathbf{b}$ with congestion α.

5 Faster Algorithm for Weak Expander Decomposition

Our input is an undirected, capacitated graph $G = (V, E, \mathbf{c})$ of order n, size m, and with $\mathbf{c} \in \mathbb{Z} \cap [1, W]$; an expansion parameter $\phi > 0$; and a vertex weighting $\mathbf{d} : V \to \mathbb{Z}_{\geq 0}$. For intuition, it may be helpful to think of $\mathbf{d}$ as $\deg_G$. Our goal is to compute a decomposition of V, $\mathcal{A}_T$, and some $\mathbf{d}_T \leq \mathbf{d}$ such that:

1. *Decomposition into expanders:* $\{\mathbf{d}_T|_A : A \in \mathcal{A}_T\}$ mix simultaneously in G with congestion $O(1/\phi)$.
2. *Few cut edges:* The total capacity of edges cut by $\mathcal{A}_T$ is $O(\phi\,\mathbf{d}(V)\log nW)$.
3. *Limited deleted demand:* $\mathbf{d}(V) - \mathbf{d}_T(V) \le \epsilon'\,\mathbf{d}(V)$, for some small constant $\epsilon' > 0$.

In the case of implementing the matching steps of cut-matching with an exact max flow oracle or a fair cuts-based approximate max flow oracle, we can set $\epsilon' = 0$. For our application to approximate max flow, our max flow oracles are too weak to achieve such a guarantee $\epsilon' = 0$, but the guarantee of (3) still suffices. We defer stating the main result of this section, Theorem 15, so it can be stated in the context of the flow oracles it assumes.

5.1 Weak Expander Decomposition with Deleted Demand

We begin by stating some basic definitions. For $A \subseteq V$ and a vertex weighting $\mathbf{d} : A \to \mathbb{Z}_{\ge 0}$, define an *$A$-commodity flow* as a multicommodity flow where each $v \in A$ is a source of $\mathbf{d}(v)$ of its unique flow commodity.

For the purposes of our analysis, we will implicitly maintain a *flow matrix* $\mathbf{F} \in \mathbb{R}_{\ge 0}^{V\times V}$ throughout. We say a flow matrix $\mathbf{F}$ is *routable with congestion κ* if there exists a V-commodity flow f such that for each $(u,v) \in V^2$, f simultaneously routes $\mathbf{F}(u,v)$ of u's commodity to v with no edge e having more than $\kappa\,\mathbf{c}(e)$ flow passing through it.

We initialize our flow matrix as $\mathbf{F}_0 = \operatorname{diag}(\mathbf{d})$, where we view $\mathbf{d}$ as a vector in $\mathbb{Z}_{\ge 0}^V$. We also initialize our set of "deleted vertices" as the empty set; $D_0 = \emptyset$. At each step $t \ge 1$, $\mathbf{d}_t$ is equal to $\mathbf{d}$, except set equal to 0 on D_{t-1}. The algorithm then proceeds in T rounds.

Some components of $\mathcal{A}_t$ become *inactive* over the course of the algorithm. The components that are *active* are those for which we have not marked all nodes as deleted and for which we have not certified that the component is an expander. We decompose $\mathcal{A}_t = \mathcal{A}_t^\circ \sqcup \mathcal{A}_t^\times$ into the active and inactive portions, respectively. In each step, we will make progress towards certifying expansion on some $\mathcal{A}_t' \subseteq \mathcal{A}_t^\circ$. For each active component $A \in \mathcal{A}_t^\circ$, we maintain a counter x_t^A recording how many times we have made progress on this set. When the counter is high enough, we have certified expansion on A with high probability and we can set the component to be inactive. We state the algorithm more formally in Algorithm 1.

Cut Player

We implement our cut step as follows.

- Sample $\boldsymbol{r}_t \in \mathbb{R}^V$, a random unit vector.
- For each $A \in \mathcal{A}_{t-1}^\circ$:
  - For each $u \in A^\circ = \{v \in A : \mathbf{d}_{t-1}(v) > 0\}$, compute

$$p_t(u) = \left\langle \frac{\mathbf{F}_{t-1}(u)}{\mathbf{d}(u)}, \boldsymbol{r}_t \right\rangle.$$

  - Compute a partition $L_A \sqcup R_A = A^\circ$ such that:
    1. $\mathbf{d}_{t-1}(L_A) = \lceil \mathbf{d}_{t-1}(A)/8 \rceil$.
    2. $\max_{u\in L_A} p_t(u) \le \eta \le \min_{u\in R_A} p_t(u)$ or $\max_{u\in R_A} p_t(u) \le \eta \le \min_{u\in L_A} p_t(u)$.

In particular, we compute a partition of A° such that L_A is guaranteed to contain a set of vertices certifying progress towards expansion. We show the existence of such a set certifying progress via a technical lemma. This is a variant of Lemma 5.15 of [11] adapted to the undirected setting.

Algorithm 1 Weak Expander Decomposition with Deletions.

WeakDecomp$(G = (V, E, \mathbf{c}), \mathbf{d}, \phi)$
. Initialize $\mathcal{A}_0 = \{V\}$, $x_0^V = 0$, $\mathbf{d}_0 = \mathbf{d}$, and implicitly initialize $\mathbf{F}_0 = \mathrm{diag}(\mathbf{d})$
. **for** $t \in [T]$:
. . **for** $A \in \mathcal{A}_{t-1}^{\times}$: add A to $\mathcal{A}_t^{\times}$ and set $x_t^A = x_{t-1}^A$;
. . **for** $A \in \mathcal{A}_{t-1}^{\circ}$: // Cut step
. . . Find sets $L_A, R_A \subseteq A^{\circ}$, where $A^{\circ} = \{v \in A : \mathbf{d}_{t-1}(v) > 0\}$
. . Compute cuts C_A and matching $\mathbf{M}_t$ between L_A to R_A in each $A \in \mathcal{A}_{t-1}^{\circ}$ // Matching step
. . **for** $u \in V$: // Update $\mathbf{d}_t$
. . . **if** $u \in L_A \cap (A \setminus C_A) \subseteq A$ for $A \in \mathcal{A}'_{t-1} \subseteq \mathcal{A}_{t-1}^{\circ}$:
. . . . Let $\mathbf{M}_t(u) = \sum_{v \in A^{\circ}} \mathbf{M}_t(u, v)$
. . . . **if** $\mathbf{M}_t(u) < \mathbf{d}(u)/2$: set $\mathbf{d}_t(u) = 0$;
. . . **else:** $\mathbf{d}_t(u) = \mathbf{d}_{t-1}(u)$;
. . **for** $A \in \mathcal{A}_{t-1}^{\circ}$: // Form $\mathcal{A}_t$, update counters
. . . **for** $S \in \{C_A, A \setminus C_A\}$ nonempty:
. . . . **if** $A \in \mathcal{A}'_{t-1}$: set $x_t^S = x_{t-1}^A + 1$;
. . . . **else**: set $x_t^S = x_{t-1}^A$;
. . . . **if** $\mathbf{d}_t(S) \leq 15\,\mathbf{d}(S)/16$:
. . . . . **for** $u \in S$: set $\mathbf{d}_t(u) = 0$;
. . . . **if** $\mathbf{d}_t(S) = 0$ or $x_t^S > 10^5 C \log n \log nW$ or $|\operatorname{supp}(\mathbf{d}_t|_S)| = 1$: add S to $\mathcal{A}_t^{\times}$; // C sufficiently large constant
. . . . **else:** add S to $\mathcal{A}_t^{\circ}$;
. . **for** $u \in V$: // Update $\mathbf{F}_t$
. . . **if** $u \in A \in \mathcal{A}_{t-1}^{\circ}$:
. . . . Set $\mathbf{F}_t(u) = \left(1 - \frac{\mathbf{M}_t(u)}{2\,\mathbf{d}(u)}\right) \mathbf{F}_{t-1}(u) + \frac{1}{2} \sum_{w \in A^{\circ}} \frac{\mathbf{M}_t(u,w)}{\mathbf{d}(w)} \cdot \mathbf{F}_t(w)$
. . . **else:** set $\mathbf{F}_t(u) = \mathbf{F}_{t-1}(u)$;
. **return** $\mathcal{A}_T$

▶ **Lemma 5.** *Let X be a finite multi-subset of $\mathbb{R}$ with $|X| \geq 2$. There exists $\eta \in \mathbb{R}$ inducing a partition $X = L_\eta \sqcup R_\eta$ with $\max(L_\eta) \leq \eta \leq \min(R_\eta)$ or $\max(R_\eta) \leq \eta \leq \min(L_\eta)$, $|L_\eta| = \lceil |X|/8 \rceil$, and with the following additional guarantees. Define $\bar{\mu} = \frac{1}{|X|} \sum_{x \in X} x$. There exists $S \subseteq L_\eta$ such that*

1. *For each $s \in S$, we have $(s - \eta)^2 \geq \frac{1}{9} \cdot (s - \bar{\mu})^2$.*
2. *$\sum_{s \in S} (s - \bar{\mu})^2 \geq \frac{1}{36} \sum_{x \in X} (x - \bar{\mu})^2$.*

The proof is similar to the proof of Lemma 5.15 in [11]. To construct our partition $L_A \sqcup R_A$, we apply Lemma 5 to the multiset of $p_t(u)$, where each value is repeated $\mathbf{d}_t(u)$ times. Elements whose duplicates appear in both L_η and R_η have $p_t(u) = \eta$ and will not be relevant for our potential reduction analysis.

▶ **Lemma 6.** *For each $t \leq T$, we can compute the cut step partitions for all $A \in \mathcal{A}_{t-1}^{\circ}$ in total time $O(mt)$.*

Matching Player

In the t^{th} matching step, we consider the following flow problem. Let G_t be the graph with all edges between different components in $\mathcal{A}_{t-1}$ deleted and all remaining edges with capacity scaled by $2/\phi$. For $v \in L_A$ for some $A \in \mathcal{A}_{t-1}^{\circ}$, we set its source to be $\Delta(v) = \mathbf{d}_{t-1}(v)$. For $v \in R_A$, we set its sink to be $\nabla(v) = \mathbf{d}_{t-1}(v)$.

We assume access to an approximate max flow oracle with the following guarantees.

▶ **Oracle 1** (Matching Player Flow Oracle). *On such a flow instance, we find* $\mathcal{A}'_{t-1} \subseteq \mathcal{A}^\circ_{t-1}$ *with*

$$\mathbf{d}_{t-1}\left(\bigcup_{A\in\mathcal{A}'_{t-1}} A\right) \geq \frac{1}{2}\,\mathbf{d}_{t-1}\left(\bigcup_{A\in\mathcal{A}^\circ_{t-1}} A\right)$$

such that, for each $A \in \mathcal{A}'_{t-1}$*:*

1. *We find a (possibly empty) cut* $C_A \subseteq A$ *with* $\mathbf{d}_{t-1}(C_A) \leq \mathbf{d}_{t-1}(A)/2$*. In addition, we have that the total capacity of computed cuts* $(C_A, A \setminus C_A)$ *is at most*

$$\frac{\phi}{8}\sum_{A\in\mathcal{A}'_{t-1}} \mathbf{d}_{t-1}(C_A) + 2\gamma\,\mathbf{d}(V).$$

2. *We find a flow routing at least* $\Delta(A \setminus C_A) - 2\gamma\,\mathbf{d}(A)$ *source of* $\Delta|_{A\setminus C_A}$ *within* A*, for* $\gamma < 1/2$*.*

▶ Remark 7. Note that, if given access to a $(1-\gamma)$-approximate max flow oracle, we could apply the oracle on each $A \in \mathcal{A}^\circ_{t-1}$ and get the desired properties with $\mathcal{A}'_{t-1} = \mathcal{A}^\circ_{t-1}$.

Updating the flow matrix

Each application of the flow oracle induces a weighted matching in each component between the source and the sink. We update the implicit flow matrix $\mathbf{F}$ accordingly. In particular, let $\mathbf{M}_t \in V \times V$ be the symmetric matrix where, for u a source vertex, $\mathbf{M}_t(u,v)$ is the amount of flow sent from vertex u to vertex v in the flow (after computing some path decomposition using link-cut trees). Importantly, since $\mathbf{M}_t(u,v)$ is formed by a path decomposition, we can guarantee that $\mathbf{M}_t(u,v)$ has at most m nonzero entries. For convenience, for $u \in A$, define $\mathbf{M}_t(u) := \sum_{v\in A}\mathbf{M}_t(u,v)$. Also define $A^\circ := \{v \in A : \mathbf{d}_{t-1}(v) > 0\}$. We can then define $\mathbf{F}_t$ recursively from $\mathbf{F}_{t-1}$ and $\mathbf{M}_t$ as follows. For $u \in A \in \mathcal{A}^\circ_{t-1}$,

$$\mathbf{F}_t(u) = \left(1 - \frac{\mathbf{M}_t(u)}{2\,\mathbf{d}(u)}\right)\mathbf{F}_{t-1}(u) + \frac{1}{2}\sum_{w\in A^\circ}\frac{\mathbf{M}_t(u,w)}{\mathbf{d}(w)}\cdot\mathbf{F}_{t-1}(w). \tag{1}$$

For $u \in A \in \mathcal{A}^\times_{t-1}$, u is not involved in the matching step, so $\mathbf{F}_t(u) = \mathbf{F}_{t-1}(u)$.

▷ Claim 8. For all t and for all $u \in V$ with $\mathbf{d}_t(u) > 0$,

$$\sum_{w\in V}\mathbf{F}(u,w) = \mathbf{d}(u).$$

▷ Claim 9. For all $t \geq 0$, $\mathbf{F}_t$ is routable with congestion $2t/\phi$.

Convergence Analysis

To prove that we achieve the desired decomposition after $T = O(\log n \log nW)$ total steps, we consider the following potential function. For each $A \in \mathcal{A}^\circ_t$, first define the average flow vector

$$\mu^A_t = \frac{1}{\mathbf{d}_t(A)}\cdot\sum_{u\in A^\circ}\mathbf{F}_t(u).$$

As before, $A^\circ := \{u \in A : \mathbf{d}_t(u) > 0\}$. Then,

$$\psi_t(A) = \mathbf{d}_t(A) \cdot \sum_{u \in A^\circ} \mathbf{d}(u) \left\| \frac{\mathbf{F}_t(u)}{\mathbf{d}(u)} - \mu_t^A \right\|_2^2.$$

Small potential implies simultaneous mixing, as stated in the following lemma.

▶ **Lemma 10.** *Suppose that $\psi_t(A) < 1/(mW)^8$ for all $A \in \mathcal{A}_t$ with $A^\circ \neq \emptyset$. Then,*

$$\{\mathbf{d}_t \,|_A : A \in \mathcal{A}_t\}$$

mix simultaneously in G with congestion $4t/\phi$.

▶ Remark 11. Given demands $\mathbf{b}_i$ respecting $\mathbf{d}_t \,|_{A_i}$ for each $i \in [r]$, we can compute the mixing routing in $O(mt \log nW)$ time by rescaling the paths in the path decompositions computed in the prior rounds' matching steps. Like the original path decomposition computation, we can implement this using link-cut trees [34].

With some work, we can show a decrease in potential in each step. Combined with standard concentration bounds, we can deduce the following.

▶ **Corollary 12.** *Let $A \in \mathcal{A}_t$ such that $\mathbf{d}_t(A) > 0$, and recall the variable x_t^A from Algorithm 1. If $x_t^A > 10^5 C \log n \log nW$, then $\psi_t(A) < 1/(mW)^8$ with high probability in n.*

To complete the proof of the main result of this section, we also need to show that $\mathbf{d}(V) - \mathbf{d}_T(V) \leq \epsilon' \mathbf{d}(V)$. Indeed, we have the following.

▶ **Lemma 13.** *For all $t \geq 0$, we have*

$$\mathbf{d}(V) - \mathbf{d}_t(V) \leq 64t\gamma \mathbf{d}(V).$$

Putting all of this together yields our first main result.

▶ **Theorem 14** (Weak Expander Decomposition with Partial Deletions)**.** *Given $G = (V, E, \mathbf{c}), \mathbf{d} \in \mathbb{R}^V_{\geq 0}, \phi > 0$, and access to an approximate max flow oracle as in Oracle 1 with parameter $1 \geq \gamma > 0$, running in time $R(n, m, \gamma)$ per query, there is an algorithm running $T = O(\log n \log nW)$ rounds of cut-matching which computes a partition $\mathcal{A}_T$ of V and $\mathbf{d}_T \leq \mathbf{d}$ with the following properties:*

1. *For each $A \in \mathcal{A}_T$ with $\mathbf{d}_T(A) > 0$, $\mathbf{d}_T(A) \geq 15\,\mathbf{d}(A)/16$. Moreover, with high probability $\{\mathbf{d}_T \,|_A : A \in \mathcal{A}\}$ mix simultaneously in G with congestion $4T/\phi$.*
2. *The total capacity of edges cut by $\mathcal{A}_T$ is at most $O((\phi \log nW + \gamma T)\,\mathbf{d}(V))$.*
3. *$\mathbf{d}(V) - \mathbf{d}_T(V) \leq 64T\gamma\,\mathbf{d}(V)$.*
4. *The algorithm runs in time $O(T(R(n, m, \gamma) + mT))$.*

5.2 Grafting in Deleted Demand

One potential weakness of the partition from Theorem 14 is property (1), its mixing guarantee. It is not quite the case that every $A \in \mathcal{A}_T$ is either certified as a (simultaneously mixing) $(\phi, \mathbf{d})$-near-expander or entirely deleted. Instead, the expanding components might have some deleted nodes inside them still (i.e., $\mathbf{d}_T(A) < \mathbf{d}(A)$). Moreover, it might be the case that we want some stronger notion of expansion, e.g., boundary-linked expansion [14].

Fortunately, we can strengthen the decomposition with one additional *grafting* step, similarly to [11]. Let $\deg_{\partial \mathcal{A}_T} : V \to \mathbb{Z}_{\geq 0}$ be the additional vertex weighting on G corresponding to the boundary of $\mathcal{A}_T$. Let $\psi > 0$ be a parameter. Think of ψ as ϕ in the case of expander decompositions; for our application, we will set $\psi = \Omega(1)$. Consider the flow instance on a subgraph of G, generated as follows.

- Let $\mathcal{A}_T^+ = \{A \in \mathcal{A}_T : \mathbf{d}_T(A) > 0\}$.
- For each $A \in \mathcal{A}_T^+$:
  - For each $u \in A$, add $\Delta(u) = \deg_{\partial \mathcal{A}_T}(u) + \mathbf{d}(u) - \mathbf{d}_T(u)$ source.
  - For each $u \in A$ with $\mathbf{d}_T(u) = \mathbf{d}(u)$, add sink $\nabla(u) = \mathbf{d}(u)/5$.
- Remove all edges cut by $\mathcal{A}_T$ from G, and scale the capacity of remaining edges by $1/\psi$.

The intuition for this flow instance is the following. If it is feasible, then we can route all the deleted and boundary demand to non-deleted demand that is certified to mix. (We set the sinks as $\mathbf{d}(u)/5$ rather than $\mathbf{d}(u)$ purely to streamline our specific application.) As such, at the cost of a slight increment in the congestion, we certify that all the expanding components A mix simultaneously with respect to demands $(\mathbf{d} + \deg_{\partial \mathcal{A}_T}(u))|_A$, not just $\mathbf{d}_T|_A$. If the flow is not feasible, because we scaled the edges and since the source is small relative to the sink, we will find sparse cuts in most components. We want the additional guarantee that any remaining source is almost entirely routed and the new boundary can be routed as well. This condition is achievable with fair cut-based max flow algorithms (e.g., [22, 21]), and we require something analogous in the definition of our flow oracle.

▶ **Oracle 2** (Grafting Flow Oracle). *On such a flow instance, for some parameter $\gamma > 0$, we find a flow with the following properties:*

1. *For each $A \in \mathcal{A}_T^+$, if $\deg_{\partial \mathcal{A}_T}(A) \le \mathbf{d}_T(A)/8$, we find a pair $(C_A, A \setminus C_A)$ such that:*
   a. *For each $u \in A \setminus C_A$ with $\Delta(u) > 0$, we route at least $(1-\gamma)\Delta(u)$ source from u to $A \setminus C_A$.*
   b. *The flow saturates at least a $(1-\gamma)$ fraction of the capacity of each edge from C_A to $A \setminus C_A$.*
   c. *We have $\sum_A \mathbf{c}_G(E(C_A, A \setminus C_A)) \le 8\psi\, \mathbf{d}(V)$.*
2. *We have*

$$\mathbf{d}\left(\bigcup_{A \in \mathcal{A}_T^+} C_A\right) \le 30(\mathbf{d}(V) - \mathbf{d}_T(V) + \deg_{\partial \mathcal{A}_T}(V)).$$

We can now state the main result of this section, a strengthening of Theorem 14 using a grafting post-processing step.

▶ **Theorem 15** (Weak Expander Decomposition with Deletions). *Suppose we have $G = (V, E, \mathbf{c})$, $\mathbf{d} \in \mathbb{Z}_{\ge 0}^V$, $\phi > 0$, $\psi > 0$, and access to Oracle 1 with parameter $1 \ge \gamma_1 > 0$ and Oracle 2 with parameter $\gamma_2 \le 1/10$, running in time $R_1(n, m, \gamma_1)$ and $R_2(n, m, \gamma_2)$ per query, respectively. Let $T = O(\log n \log nW)$. Then, there is an algorithm computing a partition $\mathcal{A} = \mathcal{A}^\circ \sqcup \mathcal{A}^\times$ of V with the following properties:*

1. *The algorithm runs in time $O(T(R_1(n, m, \gamma_1) + mT) + R_2(n, m, \gamma_2))$.*
2. *$\mathbf{d}(\bigcup_{A \in \mathcal{A}^\times} A) = O((\gamma_1 T + \phi \log nW)\, \mathbf{d}(V))$.*
3. *The total capacity of edges cut by $\mathcal{A}$ is at most $O((\phi \log nW + \gamma_1 T + \psi)\, \mathbf{d}(V))$.*
4. *$\{(\mathbf{d} + \deg_{\partial \mathcal{A}})|_A : A \in \mathcal{A}^\circ\}$ mix simultaneously in G with congestion $T/\phi + \frac{2}{\psi}$.*
5. *There exists a flow of congestion $\frac{2}{\psi}$ such that each $u \in A \in \mathcal{A}^\circ$ sends $\deg_{\partial \mathcal{A}}(u)$ flow and each $v \in V$ receives at most $\mathbf{d}(v)/4$ flow.*

6 Sufficient Conditions for Constructing a Congestion-Approximator

In this section, we show that the following properties suffice to obtain a congestion-approximator.

▶ **Theorem 16.** *Consider a capacitated graph $G = (V, E, \mathbf{c})$, and let $\alpha \geq 1$ and $\beta \geq 1$ be parameters. Consider a sequence of partitions $\mathcal{P}_1, \ldots, \mathcal{P}_L$ of $V_1, \ldots, V_L \subseteq V$. For ease of notation, let $\overline{\mathcal{P}}_0$ denote the singleton partition. For each $i \in [L]$, define $\mathcal{Q}_i$ to be the induced partition of $\overline{\mathcal{P}}_{i-1}$ on $V \setminus V_i$ i.e.,*

$$\mathcal{Q}_i = \{C \cap (V \setminus V_i) : C \in \overline{\mathcal{P}}_{i-1}, C \cap (V \setminus V_i) \neq \emptyset\},$$

and let $\overline{\mathcal{P}}_i = \mathcal{P}_i \cup \mathcal{Q}_i$ for each $i \in [L]$. Suppose the partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_L$ satisfy:

1. *$\overline{\mathcal{P}}_1$ is a partition $\{\{v\} : v \in V\}$ of singleton clusters, and $\overline{\mathcal{P}}_L$ is a partition $\{\{V\}\}$ with a single cluster.*
2. *For each $i \in [L-1]$, the collection of vertex weightings $\{\deg_{\partial \overline{\mathcal{P}}_i \cup \partial C}|_C \in \mathbb{R}^V_{\geq 0} : C \in \overline{\mathcal{P}}_{i+1}, C \subseteq V_{i+1}\}$ mixes simultaneously in G with congestion α.*
3. *For each $i \in [L-1]$, there is a flow in G with congestion β such that each $v \in V_{i+1}$ sends $\deg_{\partial \overline{\mathcal{P}}_{i+1}}(v)$ flow and receives at most $\frac{1}{4}\deg_{\partial \overline{\mathcal{P}}_i}(v)$ flow.*

For each $i \in [L]$, let partition $\mathcal{R}_{\geq i}$ be the common refinement of partitions $\overline{\mathcal{P}}_i, \ldots, \overline{\mathcal{P}}_L$, i.e.,

$$\mathcal{R}_{\geq i} = \{C_i \cap \cdots \cap C_L : C_i \in \overline{\mathcal{P}}_i, \ldots, C_L \in \overline{\mathcal{P}}_L, C_i \cap \cdots \cap C_L \neq \emptyset\}.$$

Then, their union $\mathcal{C} = \bigcup_{i \in [L]} \mathcal{R}_{\geq i}$ is a congestion-approximator with quality $48\alpha\beta L^2$.

Note that in most natural algorithms, we have $V_L = V$ in the top layer. The rest of the section proves the theorem. As in [23], we will actually need a *pseudo-congestion-approximator* analogue of Theorem 16, where $\overline{\mathcal{P}}_L$ is not necessarily the partition $\{\{V\}\}$. The precise guarantees are given below. In particular, note that assumptions (2) and (3) remain unchanged.

▶ **Lemma 17.** *Consider a capacitated graph $G = (V, E, \mathbf{c})$ with $\mathbf{c} \in [1, W] \cap \mathbb{Z}$, and let $\alpha \geq 1$ and $\beta \geq 1$ be parameters. Consider a sequence of partitions $\mathcal{P}_1, \ldots, \mathcal{P}_L$ of $V_1, \ldots, V_L \subseteq V$. For ease of notation, let $\overline{\mathcal{P}}_0$ denote the singleton partition. For each $i \in [L]$, define $\mathcal{Q}_i$ to be the induced partition of $\overline{\mathcal{P}}_{i-1}$ on $V \setminus V_i$ i.e.,*

$$Q_i = \{C \cap (V \setminus V_i) : C \in \overline{\mathcal{P}}_{i-1}, C \cap (V \setminus V_i) \neq \emptyset\},$$

and let $\overline{\mathcal{P}}_i = \mathcal{P}_i \cup \mathcal{Q}_i$ for each $i \in [L]$. Suppose the partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_L$ satisfy:

1. *$\overline{\mathcal{P}}_1$ is the partition $\{\{v\} : v \in V\}$ of singleton clusters.*
2. *For each $i \in [L-1]$, the collection of vertex weightings $\{\deg_{\partial \overline{\mathcal{P}}_i \cup \partial C}|_C \in \mathbb{R}^V_{\geq 0} : C \in \overline{\mathcal{P}}_{i+1}, C \subseteq V_{i+1}\}$ mixes simultaneously in G with congestion α.*
3. *For each $i \in [L-1]$, there is a flow in G with congestion β such that each $v \in V_{i+1}$ sends $\deg_{\partial \overline{\mathcal{P}}_{i+1}}(v)$ flow and each $v \in V_{i+1}$ receives at most $\frac{1}{4}\deg_{\partial \overline{\mathcal{P}}_i}(v)$ flow.*

For each $i \in [L]$, let partition $\mathcal{R}_{\geq i}$ be the common refinement of partitions $\overline{\mathcal{P}}_i, \ldots, \overline{\mathcal{P}}_L$, i.e.,

$$\mathcal{R}_{\geq i} = \{C_i \cap \cdots \cap C_L : C_i \in \overline{\mathcal{P}}_i, \ldots, C_L \in \overline{\mathcal{P}}_L, C_i \cap \cdots \cap C_L \neq \emptyset\}.$$

Consider their union $\mathcal{C} = \bigcup_{i \in [L]} \mathcal{R}_{\geq i}$. For any demand $\mathbf{b} \in \mathbb{R}^V$ satisfying $|\mathbf{b}(C)| \leq \delta C$ for all $C \in \mathcal{C}$, there exists a demand $\mathbf{b}' \in \mathbb{R}^V$ satisfying $|\mathbf{b}'| \leq \deg_{\partial \overline{\mathcal{P}}_L}$ and a flow routing $\mathbf{b} - \mathbf{b}'$ with congestion $48\alpha\beta L^2$.

Instead of proving Theorem 16 directly, we prove Lemma 17, which is needed for the algorithm. The proof of Lemma 17 is technical and deferred to the full version of the paper. Note that Lemma 17 implies Theorem 16.

7 Building Our Congestion-Approximator

The partitioning algorithm starts with the partition $\overline{\mathcal{P}}_1 = \{\{v\} : v \in V\}$ of singleton clusters. The algorithm then iteratively constructs partition $\overline{\mathcal{P}}_{i+1}$ given the current partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_i$. The lemma below establishes this iterative algorithm, where we substitute L for i.

▶ **Theorem 18.** *Consider a capacitated graph $G = (V, E, \mathbf{c})$. Suppose there exists partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_L$ that satisfy the following properties:*

1. *$\overline{\mathcal{P}}_1$ is the partition $\{\{v\} : v \in V\}$ of singleton clusters.*
2. *For each $i \in [L-1]$, the collection of vertex weightings $\{\deg_{\partial\overline{\mathcal{P}}_i \cup \partial C}|_C \in \mathbb{R}^V_{\geq 0} : C \in \overline{\mathcal{P}}_{i+1}, C \subseteq V_{i+1}\}$ mixes simultaneously in G with congestion $\alpha = O(\log^3(nW))$.*
3. *For each $i \in [L-1]$, there is a flow in G with congestion $\beta = O(1)$ such that each $v \in V_{i+1}$ sends $\deg_{\partial\overline{\mathcal{P}}_{i+1}}(v)$ flow and each $v \in V_{i+1}$ receives at most $\frac{1}{4}\deg_{\partial\overline{\mathcal{P}}_i}(v)$ flow.*
4. *For each $i \in [L-1]$, the size of the boundaries are decreasing: $\delta\overline{\mathcal{P}}_{i+1} \leq \delta\overline{\mathcal{P}}_i/2$.*

Then, there is an algorithm running in $O(m \log^8(nW))$ time that constructs a partition $\overline{\mathcal{P}}_{L+1}$ such that properties (2), (3), and (4) hold for $i = L$ as well.

Note that the first three properties of Theorem 18 are the same as those in Lemma 17. As suggested in Lemma 17, the new partitions will be constructed by finding a partition $\mathcal{P}_{L+1}$ of a subset $V_{L+1} \subseteq V$ and combining it with the induced partition from $\overline{\mathcal{P}}_L$ on the remaining vertices $V \setminus V_{L+1}$. This will be done via our faster algorithm for weak expander decompositions from Section 5. Before describing our algorithm further, we first note that $L = O(\log(nW))$ iterations suffice to obtain a congestion-approximator.

▶ **Corollary 19.** *Let $G = (V, E, \mathbf{c})$ be a capacitated graph with $\mathbf{c} \in [1, W] \cap \mathbb{Z}$. There is an $O(m \log^9(nW) \log\log(nW))$ time algorithm to construct a congestion-approximator $\mathcal{C}$ of G with quality $O(\log^5(nW))$. This implies an $O(m \log^9(nW) \log\log(nW) + m \log^6(nW)/\epsilon)$ time algorithm for $(1-\epsilon)$-approximate max flow.*

In the remainder of the section, we prove Theorem 18. To do so, we apply our weak expander decomposition algorithm from Section 5, which requires us to implement Oracles 1 and 2 of Theorem 15. An approximate max flow oracle suffices to implement both of these oracles, but that is exactly the problem we are trying to solve. To resolve this, [23] observed that the pseudo-congestion-approximator is a real congestion-approximator on some modified graph. They then use the real congestion-approximator on the modified graph to obtain the required flow oracles via additional post-processing, at the loss of additional log factors. Instead, we directly show that the pseudo-congestion-approximator made up of the partitions $\overline{\mathcal{P}}_1, \ldots, \overline{\mathcal{P}}_L$ suffices for solving the approximate max flow instances required by the cut-matching game, with no log factor loss. This suffices for achieving properties (2) and (3) of Theorem 18 for the next iteration of our partitioning algorithm, as stated in Section 5.

7.1 Cut-Matching via a Pseudo-Congestion-Approximator

We apply our weak expander decomposition algorithm from Section 5. The main difficulty here is developing an efficient algorithm for the matching player (i.e., implementing Oracle 1), which requires solving a max flow problem (approximately). Previous work [33] shows how to convert a congestion-approximator into an approximate max flow algorithm. In this section, we build on [33] to show that our pseudo-congestion-approximator is sufficient to approximately solve our specific max flow instance arising in the cut-matching game.

We first recall the flow instance which we need to solve for the matching player. We start with a demand $\mathbf{d} = \deg_{\partial \overline{P}_L}$ and the starting partition $\mathcal{A}_0^\circ = \{V\}$. At each iteration t of the cut-matching game, we will maintain a collection $\mathcal{A}_t^\circ$ of disjoint subsets of V. From the cut player, we obtain sets $L_A \sqcup R_A = A$ for each $A \in \mathcal{A}_t^\circ$. We wish to solve the following flow problem guaranteeing the properties of Oracle 1. Let G_t be the graph of G with all edges between components in $\mathcal{A}_t^\circ$ removed and all edge capacities scaled up by a factor of $2/\phi$. Add a source vertex s and a sink vertex t. For $A \in \mathcal{A}_t^\circ$, we do the following: for each $u \in L_A$, we add an edge (s, u) with capacity $\mathbf{d}_{t-1}(u)$, and, for each $v \in R_A$, we add an edge (v, t) with capacity $\mathbf{d}_{t-1}(v)$.

To solve this flow problem, We use the following instantiation of Sherman's algorithm [33] as stated in [21], but with the running time speedup from [17] (see Section 5 of their arXiv version), which partially routes the demand, leaving a small amount of residual demand.

▶ **Lemma 20** (Almost-Route). *Consider a graph $G = (V, E, \mathbf{c})$, two vertices $s, t \in V$, parameters $\epsilon, \tau > 0$, and a laminar family of vertex subsets $\mathcal{C}$. There is an $O(m \log(n)/\epsilon)$ time algorithm that computes either*

1. *An (s,t)-cut in G of value less than τ, or*
2. *A flow f in G routing a demand $\mathbf{d}$ such that the residual demand $\tilde{\mathbf{d}} = \tau(\mathbf{1}_s - \mathbf{1}_t) - \mathbf{d}$ satisfies*
   $|\tilde{\mathbf{d}}(C)| \leq \epsilon \cdot \delta C$ for all $C \in \mathcal{C}$.

In the setting where $\mathcal{C}$ is a congestion-approximator, the residual demand can be routed via the congestion-approximator so that the flow satisfies the input demand. We show that, in our case, a pseudo-congestion-approximator also suffices to route the residual demand.

▶ **Lemma 21.** *Given a laminar family of sets $\mathcal{C}$ defined as in Lemma 17 for $G = (V, E, \mathbf{c})$, there is an algorithm running in $O(m \log^6(nW) \log\log(nW))$ time which satisfies the guarantees of Oracle 1.*

With this, we have an implementation of Oracle 1 in $O(m \log^6(nW) \log\log nW)$ time, which we can already use to apply Theorem 14. Next, we will also implement Oracle 2, so we can apply Theorem 15.

7.2 Grafting Deleted Nodes

In this section, we will use the pseudo-congestion-approximator to solve another approximate max flow problem needed to construct the weak expander decomposition, Oracle 2 from Section 5. We are using Oracle 2 to guarantee boundary-linkedness in our weak expander decomposition and obtain the full guarantees given in Theorem 15.

Now, we restate the flow problem which we want to solve. Recall that in our setting, $\mathbf{d}_T(u)$ is either 0 or $\mathbf{d}(u)$ for all $u \in V$, and $\psi = \Theta(1)$.

- Let $\mathcal{A}_T^+ = \{C \in \mathcal{A}_T : \mathbf{d}_T(C) > 0\}$ and $V^+ = \bigcup_{C \in \mathcal{A}_T^+} C$.
- For each $C \in \mathcal{A}_T^+$:
  - For each $u \in C$, add $\Delta(u) = \deg_{\partial C}(u) + \mathbf{d}(u) - \mathbf{d}_T(u)$ source.
  - For each $u \in C$ with $\mathbf{d}_T(u) = \mathbf{d}(u)$, add sink $\nabla(u) = \mathbf{d}(u)/5$.
- Remove all edges cut by $\mathcal{A}_T^+$ from G, and scale the capacity of remaining edges by $1/\psi$.

Note that this flow problem on G is the combination of independent flow problems in each component of $\mathcal{A}_T^+$. We wish to find a fair flow-cut pair for this flow problem. In order to solve this, we first state our subroutine for one-sided fair cuts, which we prove in the full version of the paper.

▶ **Theorem 22.** *Consider a graph $G = (V, E, \mathbf{c})$ with $\mathbf{c} \in \mathbb{Z} \cap [1, W]$, a vertex subset $U \subseteq V$, and a vertex $t \in U$. Let $\mathcal{C}$ be a laminar family of vertex subsets of $V \setminus \{t\}$ of total size z, such that any demand vector $\mathbf{b} \in \mathbb{R}^V$ satisfying $|\mathbf{b}(C)| \leq \delta C$ can be routed in G with congestion q in time T. There is an algorithm in time $O(z \log nW + \epsilon^{-1} qm \log^3 nW + T \log nW)$ that computes a set $A \subseteq U$ containing t and a flow f such that*

1. *$\delta A \leq 4\delta U$.*
2. *Each edge $(u, v) \in \partial A$ with $u \notin A$, $v \in A$ sends at least $(1 - \epsilon)$ fraction capacity of flow into A.*
3. *Each vertex $v \in A \setminus \{t\}$ carries a net flow of zero.*

In order to simulate the independent flow problems on each component of $\mathcal{A}_T^+$, we need to define a slightly augmented graph G^{flow} starting from $G = (V, E, \mathbf{c})$, on which we will define our flow problems. Start with the graph G. For each $C \in \mathcal{A}_T^+$ and each edge $e = (u, v)$ on the boundary ∂C, we add a new "split node" x_e, remove the edge (u, v), and add two edges, (u, x_e) and (x_e, v), each with capacity $\mathbf{c}(e)$. We also add a new node t, and, for each $C \in \mathcal{A}_T^+$, add an edge connecting every node $u \in C$ satisfying $\mathbf{d}_T(u) = \mathbf{d}(u)$ to t. This node t will represent the sink in the flow problem, and we specify the rest of the capacities next.

We set the capacities of edges (u, t) to be $\mathbf{d}(u)/5$, simulating the sink in the original flow problem. Then for each vertex $u \in V^+$ with $\mathbf{d}_T(u) = 0$, add a "leaf node" from u denoted $\tilde{u}$ and an edge $(\tilde{u}, u)$ with capacity $\mathbf{d}(u) - \mathbf{d}_T(u) = \mathbf{d}(u)$. We ensure that $u \in U$ and $\tilde{u} \notin U$, which intuitively simulates a source of $\mathbf{d}(u)$ at each such $u \in V^+$. In particular, note that if, for example, $u \in A$, then $(1 - \epsilon)\,\mathbf{c}(e)$ flow is sent along $(\tilde{u}, u)$ from (2) of Theorem 22. Furthermore, for each edge $e = (u, v) \in \partial C$ for $C \in \mathcal{A}_T^+$, we ensure that $u, v \in U$ and $x_e \notin U$; since the capacity of (u, x_e) and (x_e, v) are both $\mathbf{c}(e)$ and these are the only edges incident to split node x_e, this intuitively simulates having source of $\mathbf{c}(e)$ on u and v, with the additional property that no flow can be sent between components.

For technical reasons, we also connect each leaf node $\tilde{u}$ to t with an edge $(\tilde{u}, t)$ of capacity $(\mathbf{d}(u) - \mathbf{d}_T(u))/5 = \mathbf{d}(u)/5$. For similar technical reasons, we also add an edge connecting each vertex $u \notin V^+$ to t with an edge of capacity of $\mathbf{d}(u)/5$. All flow using these edges will be removed, so our solution is a valid flow for Oracle 2, but we need these edges for routing some residual demands.

Finally, to apply Theorem 22, we define $U = V \cup \{t\}$ (not including the additional leaf nodes or split nodes). Importantly, the total capacity of the boundary is still not too much larger than the total source.

▷ Claim 23. We have $\delta_{G^{\text{flow}}} U \leq 6\Delta(V)/5$.

Let $\mathcal{C}$ be defined as in Lemma 17, so that $\mathcal{C}$ is a pseudo-congestion-approximator. We augment $\mathcal{C}$ to a congestion-approximator $\mathcal{C}^{\text{flow}}$ in G^{flow} as follows. For each $C \in \mathcal{C}$, define $\widetilde{C} = C \cup \{\tilde{u} : u \in V^+ \cap C\} \cup \{x_e : e = (u, v), u, v \in C\}$ as the set with the copy $\tilde{u}$ added (if it exists) for each $u \in C$, and split node x_e added if both endpoints are in C. Now define

$$\mathcal{C}^{\text{flow}} = \{\widetilde{C} : C \in \mathcal{C}\} \cup \{\{\tilde{u}\} : u \in V^+\} \cup \{\{x_e\} : \text{split node } x_e\}.$$

We show that $\mathcal{C}^{\text{flow}}$ has the required properties to apply Theorem 22.

▶ **Lemma 24.** *$\mathcal{C}^{flow}$ is a laminar family of vertex subsets excluding t of total size $z = O(nL)$ such that any demand vector $\mathbf{b}$ on G^{flow} satisfying $|\mathbf{b}(C)| \leq \delta_{G^{\text{flow}}} C$ for each $C \in \mathcal{C}^{\text{flow}}$ can be routed with congestion $O(\alpha\beta L^2)$ in time $O(m \log^4(nW))$.*

▶ **Lemma 25.** *Assuming the conditions of Theorem 18, there is an algorithm implementing the guarantees of Oracle 2 in time $O(m \log^8(nW))$.*

7.3 Proof of Theorem 18

In this previous two subsections, we have shown how to implement Oracles 1 and 2, so we can apply Theorem 15. We restate it here for convenience:

▶ **Theorem 15** (Weak Expander Decomposition with Deletions). *Suppose we have $G = (V, E, \mathbf{c})$, $\mathbf{d} \in \mathbb{Z}^V_{\geq 0}$, $\phi > 0$, $\psi > 0$, and access to Oracle 1 with parameter $1 \geq \gamma_1 > 0$ and Oracle 2 with parameter $\gamma_2 \leq 1/10$, running in time $R_1(n, m, \gamma_1)$ and $R_2(n, m, \gamma_2)$ per query, respectively. Let $T = O(\log n \log nW)$. Then, there is an algorithm computing a partition $\mathcal{A} = \mathcal{A}^\circ \sqcup \mathcal{A}^\times$ of V with the following properties:*

1. *The algorithm runs in time $O(T(R_1(n, m, \gamma_1) + mT) + R_2(n, m, \gamma_2))$.*
2. *$\mathbf{d}(\bigcup_{A \in \mathcal{A}^\times} A) = O((\gamma_1 T + \phi \log nW)\, \mathbf{d}(V))$.*
3. *The total capacity of edges cut by $\mathcal{A}$ is at most $O((\phi \log nW + \gamma_1 T + \psi)\, \mathbf{d}(V))$.*
4. *$\{(\mathbf{d} + \deg_{\partial \mathcal{A}})|_A : A \in \mathcal{A}^\circ\}$ mix simultaneously in G with congestion $T/\phi + \frac{2}{\psi}$.*
5. *There exists a flow of congestion $\frac{2}{\psi}$ such that each $u \in A \in \mathcal{A}^\circ$ sends $\deg_{\partial \mathcal{A}}(u)$ flow and each $v \in V$ receives at most $\mathbf{d}(v)/4$ flow.*

Now, we prove Theorem 18. Choose $\gamma_1 = T/10^{10}$, $\gamma_2 = 1/10^{10}$, $\phi = \log(nW)/10^{10}$, $\psi = 1/10^{10}$, and define $\mathbf{d} = \deg_{\partial \overline{\mathcal{P}}_L}$. Assuming the conditions given in Theorem 18, we have given an algorithm implementing Oracle 1 in time $R_1(m, n, \gamma_1) = O(m \log^6(nW) \log\log(nW))$ and Oracle 2 in time $R_2(m, n, \gamma_2) = O(m \log^8(nW))$ (see Lemmas 21 and 25). Hence, we can apply Theorem 15, to obtain a partition $\mathcal{A} = \mathcal{A}^\circ \sqcup \mathcal{A}^\times$ of V with properties (1)–(5) in time $O(m \log^8(nW) \log\log(nW))$.

We define $\mathcal{P}_{L+1} = \mathcal{A}^\circ$, $V_{L+1} = \bigcup_{A \in \mathcal{A}^\circ} A$, and extend $\mathcal{P}_{L+1}$ from a partition of V_{L+1} to a partition of V by defining $\mathcal{Q}_{L+1} = \{C \cap (V \setminus V_{L+1}) : C \in \overline{\mathcal{P}}_L\}$ and $\overline{\mathcal{P}}_{L+1} = \mathcal{P}_{L+1} \sqcup \mathcal{Q}_{L+1}$. We now verify properties (2), (3), and (4) in Theorem 18. Property (2) follows by property (4) from Theorem 15. Property (3) follows from property (5) from Theorem 15. Finally, consider property (4). The capacity of edges cut by $\mathcal{P}_{L+1}$ is upper bounded by the capacity of edges cut by $\mathcal{A}$ and the capacity of edges cut by $\mathcal{Q}_{L+1}$ and not already cut by $\mathcal{P}_{L+1}$ is upper bounded by $\deg_{\partial \overline{\mathcal{P}}_L}(\bigcup_{A \in \mathcal{A}^\times} A) = \mathbf{d}(\bigcup_{A \in \mathcal{A}^\times} A)$. By properties (2) and (3) from Theorem 15, the claimed bound follows.

References

1 Arpit Agarwal, Sanjeev Khanna, Huan Li, Prathamesh Patil, Chen Wang, Nathan White, and Peilin Zhong. Parallel approximate maximum flows in near-linear work and polylogarithmic depth. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3997–4061. SIAM, 2024. doi:10.1137/1.9781611977912.140.

2 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 9:1–9:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.9.

3 Aaron Bernstein, Joakim Blikstad, Jason Li, Thatchaphol Saranurak, and Ta-Wei Tu. Combinatorial maximum flow via weighted push-relabel on shortcut graphs. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 464–485, 2025. doi:10.1109/FOCS63196.2025.00026.

4 Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, and Ta-Wei Tu. Maximum flow by augmenting paths in $n^{2+o(1)}$ time. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2056–2077. IEEE, 2024. doi:10.1109/FOCS61266.2024.00123.

5 Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2023)*, pages 24–33, 2003. doi:10.1145/777412.777418.

6 Li Chen, Rasmus Kyng, Yang Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *Journal of the ACM*, 72(3):1–103, 2025. doi:10.1145/3728631.

7 Julia Chuzhoy and Sanjeev Khanna. A faster combinatorial algorithm for maximum bipartite matching. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2185–2235. SIAM, 2024. doi:10.1137/1.9781611977912.79.

8 Julia Chuzhoy and Sanjeev Khanna. Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 83–94, 2024. doi:10.1145/3618260.3649725.

9 Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 451–460, 2008. doi:10.1145/1374376.1374441.

10 George B Dantzig. Application of the simplex method to a transportation problem. *Activity analysis and production and allocation*, 1951.

11 Henry Fleischmann, George Z Li, and Jason Li. Improved directed expander decompositions. *arXiv preprint arXiv:2507.09729*, 2025. doi:10.48550/arXiv.2507.09729.

12 Yu Gao, Yang Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than Goldberg–Rao. *SIAM Journal on Computing*, pages FOCS21–85–FOCS21–156, 2021. doi:10.1137/22M1476666.

13 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

14 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. doi:10.1137/1.9781611976465.132.

15 Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2023)*, pages 34–43, 2003. doi:10.1145/777412.777419.

16 Monika Henzinger, Robin Münk, and Harald Räcke. An improved quality hierarchical congestion approximator in near-linear time. *arXiv preprint arXiv:2511.03716*, 2025. doi:10.48550/arXiv.2511.03716.

17 Arun Jambulapati and Kevin Tian. Revisiting area convexity: Faster box-simplex games and spectrahedral generalizations. *NeurIPS*, 36:57583–57596, 2023.

18 Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014. doi:10.1137/1.9781611973402.16.

19 Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4):19:1–19:15, 2009. doi:10.1145/1538902.1538903.

20 Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 755–764, 2013. doi:10.1145/2488608.2488704.

21 Jason Li and Owen Li. A simple and fast algorithm for fair cuts. In *Integer Programming and Combinatorial Optimization - 26th International Conference Proceedings (IPCO 2025)*, volume 15620 of *Lecture Notes in Computer Science*, pages 400–411, 2025. doi:10.1007/978-3-031-93112-3_29.

22 Jason Li, Danupon Nanongkai, Debmalya Panigrahi, and Thatchaphol Saranurak. Near-linear time approximations for cut problems via fair cuts. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 240–275, 2023. doi:10.1137/1.9781611977554.CH10.

23 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, 2025. doi:10.1137/1.9781611978322.68.

24 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016. doi:10.1109/FOCS.2016.70.

25 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 ACM Symposium on Theory of Computing (STOC 2008)*, pages 461–470, 2008. doi:10.1145/1374376.1374442.

26 Richard Peng. A note on cut-approximators and approximating undirected max flows. *CoRR*, 2014. arXiv:1411.7631.

27 Richard Peng. Approximate undirected maximum flows in $O(m \operatorname{polylog}(n))$ time. In *Proceedings of the 27th annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1862–1867. SIAM, 2016. doi:10.1137/1.9781611974331.CH130.

28 Harald Räcke. Minimizing congestion in general networks. In *43rd Symposium on Foundations of Computer Science, FOCS 2002, Vancouver, BC, Canada, November 16-19, 2002, Proceedings*, pages 43–52. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181881.

29 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 255–264, 2008. doi:10.1145/1374376.1374415.

30 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 2014)*, pages 227–238. SIAM, 2014. doi:10.1137/1.9781611973402.17.

31 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2635. SIAM, 2019. doi:10.1137/1.9781611975482.162.

32 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. doi:10.1109/FOCS.2013.36.

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

34 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

35 Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. doi:10.1137/090771430.