Combinatorial Maximum Flow via Weighted Push-Relabel on Shortcut Graphs

Aaron Bernstein* Joakim Blikstad† Jason Li‡ Thatchaphol Saranurak§
Ta-Wei Tu¶

Abstract

We give a combinatorial algorithm for computing exact maximum flows in directed graphs with n vertices and edge capacities from $\{1, \ldots, U\}$ in $\tilde{O}(n^2 \log U)$ time, which is near-optimal on dense graphs. This shaves an $n^{o(1)}$ factor from the recent result of [Bernstein–Blikstad–Saranurak–Tu FOCS'24] and, more importantly, greatly simplifies their algorithm. We believe that ours is by a significant margin the simplest of all algorithms that go beyond $\tilde{O}(m\sqrt{n})$ time in general graphs. To highlight this relative simplicity, we provide a full implementation of the algorithm in C++.

The only randomized component of our work is the cut-matching game. Via existing tools, we show how to derandomize it for *vertex-capacitated* max flow and obtain a deterministic $\tilde{O}(n^2)$ time algorithm. This marks the first deterministic near-linear time algorithm for this problem (or even for the special case of bipartite matching) in any density regime.

*New York University, bernstei@gmail.com. Supported by Sloan Fellowship, Google Research Fellowship, NSF Grant 1942010, and Charles S. Baylis endowment at NYU.

†CWI Amsterdam, joakim@cwi.nl.

‡Carnegie Mellon University, jmli@cs.cmu.edu.

§University of Michigan, thsa@umich.edu. Supported by NSF Grant CCF-2238138. Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT, Sofia University "St. Kliment Ohridski" as part of the Bulgarian National Roadmap for Research Infrastructure.

¶Stanford University, taweitu@stanford.edu. Supported by a Stanford School of Engineering Fellowship, a Microsoft Research Faculty Fellowship, and NSF CAREER Award CCF-1844855.

Contents

1 Introduction

Fast algorithms for computing maximum flows have been pivotal in algorithmic research, inspiring various paradigms such as graph sparsification, dynamic data structures, and the application of continuous optimization to combinatorial problems. These algorithms find numerous applications in problems like bipartite matching, minimum cuts, and Gomory–Hu trees [GH61, LP20, LNP+21, CLN+21, CHLP23, ALPS23]. Below, we summarize the development of fast maximum flow algorithms since Dantzig's introduction of the problem [Dan51]. The input for this problem consists of a directed graph with n vertices and m edges. For simplicity, we assume in this introduction that edge capacities range from $\{1, \ldots, \text{poly}(n)\}$.

Augmenting Paths: The Impact of Clarity. Ford and Fulkerson [FF56] introduced the *augmenting path* framework, where algorithms iteratively identify an augmenting path in the residual graph and augment the flow along this path by a value equal to the bottleneck of the augmenting path. Using this framework, algorithms with a time complexity of $O(m \cdot \min\{m^{1/2}, n^{2/3}\})$ were discovered since the 70s [Kar73, ET75, GR98]; this time bound remained the best record for the next 40 years.

Despite the lack of progress on time complexity, the clarity of this framework has enabled impactful developments that extend far beyond the maximum flow problem. For example, the blocking flow [Kar73, Din70, GR98] has become an important object for parallel, distributed, and local algorithms [Vis92, Coh95, AKR07, OZ14, CS20, HHS23]. The push-relabel method [GT88] generalizes to solve numerous problems, including approximate low-degree spanning trees [CRRT09], global minimum edge/vertex cuts [HO94, HRG00], parametric flow [GGT89, GMQT12], submodular function minimization [Iwa03, FI03], submodular flow [FM12], and packing/covering matroid [Qua24]. Push-relabel algorithms have also been adapted to local, dynamic, and parallel settings [HRW17, SW19, CMGS25].

Interior Point Methods and Dynamic Graph Data Structures. In the 2000s, Daitch and Spielman [DS08] introduced a novel framework for computing maximum flow via *interior point methods* (IPMs) by reducing the problem to solving $\tilde{O}(\sqrt{m})$ instances of electrical flow, which can be solved in near-linear time [ST04].[1] This sparked further advances in IPMs [Mad13, LS14, Mad16, LS20b, KLS20] that minimize the number of iterations required to solve electrical flow or related problems. Consequently, an $\tilde{O}(m\sqrt{n})$ time maximum flow algorithm [LS14] and, for unit-capacity graphs, an $m^{4/3+o(1)}$ time algorithm [KLS20] were developed, breaking the previous time record set by the augmenting-path approaches.

Since 2020, the emphasis has shifted from minimizing the number of iterations in IPMs to reducing the cost per iteration through *dynamic graph data structures*. A series of impressive works [vdBLN+20, vdBLL+21, GLP21, vdBGJ+22] culminated in almost-optimal $m^{1+o(1)}$ time algorithms for maximum flow and min-cost flow [CKL+22, vdBCP+23], along with their extensions to dynamic settings [CKL+24, vdBCK+24].

[1] In this paper, we use $\tilde{O}(\cdot)$ to hide poly-logarithmic factors in n. As standard in modern graph algorithms, we use *near-linear* time to refer to algorithms with a running time of $\tilde{O}(m)$, and *almost-linear* time for those running in $m^{1+o(1)}$ time.

In Pursuit of Clarity. These breakthrough algorithms with almost-optimal time bounds are, however, still not simple in two different aspects. Conceptually, the IPM framework updates the flow solutions without a clear combinatorial interpretation. And, technically, the required dynamic data structures are still highly involved.[2] The current state of the art motivated a new search for algorithms that are more "combinatorial", as a proxy for clarity.

With this motivation, Chuzhoy and Khanna [CK24a, CK24b] presented an algorithm for maximum bipartite matching, a special case of maximum flow, that runs in $n^{2+o(1)}$ time without using IPM. Their algorithm is conceptually clearer; it repeatedly increases the flow value along paths listed by dynamic shortest-path data structures. These data structures, however, are still complex.

Recently, Bernstein, Blikstad, Saranurak, and Tu [BBST24] introduced an augmenting-path-based maximum flow algorithm with a running time of $n^{2+o(1)}$, improving upon the $O(m \cdot \min\{m^{1/2}, n^{2/3}\})$ bound within the augmenting path framework. They bypassed heavy dynamic data structures and replaced them with a simple variant of push-relabel called *weighted push-relabel*. The weighted push-relabel algorithm itself is clear, combinatorial, and implementable. The authors of [BBST24] showed that given edge weights derived from an *expander hierarchy,* weighted push-relabel is guaranteed to efficiently return a flow with good quality. Unfortunately, their algorithm for constructing the expander hierarchy is very complex. Furthermore, their analysis of why the returned flow has good quality, given the expander hierarchy, is also quite intricate. Thus, the goal of obtaining much simpler and clearer algorithms remains highly desirable.

Our Result. We propose a new combinatorial maximum flow algorithm that is significantly simpler and also strictly speeds up all recent combinatorial algorithms [CK24a, CK24b, BBST24], improving the running time from $n^{2+o(1)}$ to $O(n^2 \cdot \text{polylog}(n))$.[3]

Theorem 1.1. *There is a randomized algorithm that, given an n-vertex directed graph G with edge capacities from $\{1, \ldots, U\}$ and $s, t \in V(G)$, finds a maximum (s,t)-flow in G with high probability in $O(n^2 \log^{19} n \log U)$ time.*

It is worth noting though that while our $\tilde{O}(n^2)$ time algorithm improves on the almost-linear time algorithm of [CKL+22] on dense graphs, there also exists an IPM-based $\tilde{O}(m + n^{1.5})$ time algorithm of [vdBLL+21] that already runs in near-linear time on dense graphs.

Overall, we believe that our algorithm is by a significant margin the simplest among all exact maximum flow algorithms for general graphs that go beyond the $\tilde{O}(m\sqrt{n})$ bound of [LS14], which include all the recent progress using IPM plus dynamic graph data structures.[4] Our result is also relatively self-contained. The main outside tool that we need is weighted push-relabel from [BBST24], which is a simple extension of classic push-relabel; a fully self-contained description and analysis is given in [BBST24, Section 4]. Beyond that we use:

[2] There has been exciting progress in simplicity among IPM-based algorithms. The involved dynamic min-ratio cycle data structure in [CKL+22, vdBCP+23, CKL+24] was recently replaced by a simpler dynamic min-ratio cut data structure [vdBCK+24] based on dynamic tree cut sparsifiers [GRST21].

[3] While we work out an explicit number of $O(\log n)$ factors in the $\tilde{O}(\cdot)$ notation, we made no significant effort in minimizing it. There are places (e.g., the expander decomposition algorithm) where more efficient algorithms should exist following standard techniques, yet we chose slower ones that we believe are simpler for presentation, analysis, and implementation.

[4] We are only comparing ourselves to other algorithms for general graphs (directed, capacitated). There is a separate line of work for directed graphs *with small capacities*, culminating in $m^{4/3+o(1)}$ time algorithms [Mad16, LS20b, LS20a, Kat20]; these works use the IPM framework but do not require dynamic data structures. Finally, for the special case of *undirected, uncapacitated, simple* graphs, there is a much simpler $\tilde{O}(n^2)$ time algorithm of Karger and Levine [KL15].

- Standard graph algorithms such as topological sort and finding strongly connected components.
- Classic tools for capacitated flow such as link-cut trees and fractional-to-integral flow rounding.
- The heaviest machinery that we use is the cut-matching game for expanders. We specifically use the "non-stop" cut-matching game. Loosely speaking, whereas the standard cut-matching game either certifies that the graph is an expander or finds a sparse cut, the non-stop version either certifies almost-expansion or returns a *balanced* sparse cut. This has already become a relatively standard tool for undirected expanders [RST14, SW19, LRW25]. The algorithm for directed expanders is similar but needs a more refined analysis; we use as a black box the result of [FLL25].

Implementation. As evidence of its relative simplicity, we present the full implementation of Theorem 1.1 in C++ [BT25]. To our knowledge, this is the first complete implementation of any maximum flow algorithm that is provably faster than $O(m\sqrt{n})$ time in any density regime. Previously, [Kav24] gave a partial implementation of the almost-linear time algorithm of [CKL+22], but significant parts are missing. The lack of implementation of the recent theoretically efficient maximum flow algorithms is likely due to the sheer complexity of the algorithm descriptions that results from their reliance on dynamic graph data structures.

We emphasize that our implementation is not at all practical, particularly because of the $\log^{19} n$ factor in our running time. Our goal is merely to show that the entire algorithm can be captured from start to finish using code of reasonable length.

Derandomization. The only randomized component of Theorem 1.1 is the KRV-style cut-matching game from [FLL25] for computing expander decomposition. There exists a deterministic variant of the cut-matching game [CGL+20, BPS20], but it incurs an $n^{o(1)}$ overhead in the running time and the quality of expansion, while also being more complicated. Using [CGL+20, BPS20], it should be straightforward to make our algorithm deterministic, but with running time of $n^{2+o(1)} \log U$ instead of $\tilde{O}(n^2 \log U)$.

In fact, for the special case of *vertex-capacitated* graphs (instead of edge-capacitated), our algorithm naturally gives a *deterministic* $\tilde{O}(n^2)$ time algorithm when combined with the appropriate implementation of the cut-matching game. This is the first near-linear time deterministic algorithm for the problem on dense graphs. Even for the special case of bipartite matching, no near-linear time *deterministic* algorithms were previously known in any density regime. In particular, the $\tilde{O}(m + n^{1.5})$ time algorithm by [vdBLN+20, vdBLL+21] is randomized.

Theorem 1.2. *There is a deterministic algorithm that, given an n-vertex directed graph G with vertex capacities from $\{1, \ldots, U\}$ and $s, t \in V(G)$, finds a maximum (s,t)-flow in G in $O(n^2 \log^{46} n \log U)$ time.*

Techniques. The main technical difficulty faced by [BBST24] in their bottom-up expander hierarchy construction is enforcing the hierarchical structure in between levels. That is, the next level $i+1$ need to be a coarsening of the previous level i, in the sense that each cluster on level $i+1$ is the (disjoint) union of some clusters on level i. Unfortunately, directly building a coarsening at level $i+1$ (without changing levels i and below) is infeasible, in that the multiplicative losses build up significantly over the levels of the hierarchy. Instead, [BBST24] need to re-cluster levels i

and below in order to build level $i+1$ with minimal losses. The final algorithm continuously and dynamically breaks down and repairs previous levels while building the next level, a process which is highly technical and suffers an extra $n^{o(1)}$ factor.

Our starting point is a recent bottom-up *weak* expander hierarchy construction for undirected graphs [LRW25].[5] Their simple conceptual insight is that if one settles for weak expanders, then the back-and-forth rebuilding between levels is completely unnecessary. Instead, we can first build the next level $i+1$ without regard to the structure of the previous i levels, so that the coarsening property is not ensured. To enforce coarsening, simply *refine* each previous level with respect to level $i+1$. That is, each cluster on a previous level is further partitioned into all possible non-empty intersections between that cluster and some cluster on level $i+1$. It turns out that unlike (strong) expander decomposition, weak expander decompositions are closed under refinements: if each cluster in a weak expander decomposition is partitioned arbitrarily, then the new clustering is still a weak expander decomposition.

However, the original algorithm of [BBST24] commits to strong expander hierarchy for a good reason: their weighted push-relabel framework breaks down in subtle ways for weak expander hierarchy. Surprisingly, we show that adding *shortcuts* to expanders, as similarly done on undirected graphs [HHL+24, LRW25], resolves this issue. More precisely, we run the weighted push-relabel flow algorithm of [BBST24], where the edge weights are induced by an expander hierarchy, on the original input graph *union with shortcuts*, where the shortcut edges are stars on top of each expander in the hierarchy. Thus, our algorithm follows the augmenting-path framework, except that the paths can use the shortcut edges.

This idea leads to major simplification in both the algorithm and analysis of [BBST24], as will be explained in detail in Section 3. At a high level, the shortcut enables reasoning about the flow of edges at different levels of the hierarchy independently, whereas [BBST24] needed an intricate analysis to handle the interactions of edges across levels.

2 Preliminaries

Let $\mathbb{N} \stackrel{\text{def}}{=} \{1, 2 \dots\}$ be the set of positive integers and $1/\mathbb{N} \stackrel{\text{def}}{=} \{x : 1/x \in \mathbb{N}\}$. We use boldface $\boldsymbol{a}$ to denote a vector, where $\boldsymbol{a} \le \boldsymbol{b}$ means $\boldsymbol{a}$ is entry-wise upper-bounded by $\boldsymbol{b}$. For a vector $\boldsymbol{w}$ we may write $\boldsymbol{w}(S) \stackrel{\text{def}}{=} \sum_{e \in S} \boldsymbol{w}(e)$, and $\boldsymbol{w}|_S$ denote the restriction of $\boldsymbol{w}$ to S. Let $\mathbf{1}_S$ denote the vector that has one on entries in S and zero elsewhere.

2.1 Basic Definitions

Graphs. Without stated otherwise, in this paper we consider directed edge-capacitated graphs of the form $G = (V, E, \boldsymbol{c}_G)$, where $\boldsymbol{c}_G \in \mathbb{R}^E_{\ge 0}$. When the context is clear, we may drop $\boldsymbol{c}_G$ from the definition of the capacitated graph and implicitly use the subscript G to indicate its underlying graph. By default, we will assume that the capacities are positive integers. Via standard capacity scaling techniques (e.g., [BBST24, Appendix B]), we will also assume that the capacities are bounded by n^2, i.e., $\|\boldsymbol{c}_G\|_\infty \le n^2$, by incurring an $O(\log U)$ overhead in the final running time.

For a directed edge $e = (u, v)$, let $\text{tail}(e) \stackrel{\text{def}}{=} u$ and $\text{head}(e) \stackrel{\text{def}}{=} v$. Let $E_G(A, B)$ for disjoint $A, B \subseteq V$ denote the set of edges whose tails are in A and heads in B.

[5] A similar strategy was implemented by Bartal [Bar96] to construct a low-diameter decomposition hierarchy.

Incident Edges, Degree, Volume, and Cuts. Consider an $F \subseteq E$. Let $\delta_F(v)$ be the set of edges in F incident to v, whose sum of capacities we denote by $\deg_F(v) \stackrel{\text{def}}{=} \sum_{e \in \delta_F(v)} \boldsymbol{c}_G(e)$. Let $\mathrm{vol}_F(S)$ for $S \subseteq V$ be $\mathrm{vol}_F(S) \stackrel{\text{def}}{=} \sum_{v \in S} \deg_F(v)$. Let $\deg_G(v) \stackrel{\text{def}}{=} \deg_E(v)$ and $\mathrm{vol}_G(S) \stackrel{\text{def}}{=} \mathrm{vol}_E(S)$. We may use $\mathrm{vol}(v)$ and $\deg(v)$ interchangeably for a vertex v, and we may also use them as vectors on V, in which case we will use the notation $\mathbf{deg}, \mathbf{vol}$ (with appropriate subscripts). For $S \subseteq V$, let $\overline{S} \stackrel{\text{def}}{=} V \setminus S$.

Flows. A *(single-commodity) flow* $\boldsymbol{f} \in \mathbb{R}^E_{\geq 0}$ assigns non-negative value to each edge. The *congestion* of a flow $\boldsymbol{f}$ in G is $\mathrm{cong}_G(\boldsymbol{f}) \stackrel{\text{def}}{=} \max_{e \in E} \boldsymbol{f}(e)/\boldsymbol{c}_G(e)$. Let $\boldsymbol{f}^{\text{out}}(u) = \sum_{(u,v) \in E} \boldsymbol{f}((u,v)) - \sum_{(v,u) \in E} \boldsymbol{f}((v,u))$ denote the net outgoing flow at vertex $u \in V$.

A *(single-commodity) vertex-demand* is a pair $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with *source* and *sink* vector $\boldsymbol{\Delta}, \boldsymbol{\nabla} \in \mathbb{R}^V_{\geq 0}$. A *flow instance* is a tuple $\mathcal{I} = (G, \boldsymbol{\Delta}, \boldsymbol{\nabla})$, and we often consider *diffusion instances* where $\|\boldsymbol{\Delta}\|_1 \leq \|\boldsymbol{\nabla}\|_1$. Let $\boldsymbol{\Delta}_{\boldsymbol{f}}(u) \stackrel{\text{def}}{=} \max\{0, \boldsymbol{\Delta}(u) - \boldsymbol{\nabla}(u) - \boldsymbol{f}^{\text{out}}(u)\}$ and $\boldsymbol{\nabla}_{\boldsymbol{f}}(u) \stackrel{\text{def}}{=} \max\{0, \boldsymbol{\nabla}(u) - \boldsymbol{\Delta}(u) + \boldsymbol{f}^{\text{out}}(u)\}$ be the *residual source* and *sink* demands at u. A flow $\boldsymbol{f}$ *partially routes* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ if $\boldsymbol{\Delta}_{\boldsymbol{f}} \leq \boldsymbol{\Delta}$ and $\boldsymbol{\nabla}_{\boldsymbol{f}} \leq \boldsymbol{\nabla}$, and the *value* of such an $\boldsymbol{f}$ is $|\boldsymbol{f}| \stackrel{\text{def}}{=} \sum_{u \in V} (\boldsymbol{\Delta}(u) - \boldsymbol{\Delta}_{\boldsymbol{f}}(u))$, i.e., the amount of source (equivalently, sink) demand routed. The flow $\boldsymbol{f}$ *(completely) routes* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ if $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1$ (equivalently $\boldsymbol{\Delta}_{\boldsymbol{f}} = \mathbf{0}$) and $\boldsymbol{\nabla}_{\boldsymbol{f}} \geq \mathbf{0}$. The demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is *routable* (with congestion κ) if there exists a flow (with congestion κ) routing it. The flow $\boldsymbol{f}$ is *feasible* for $\mathcal{I}$ if $\boldsymbol{\Delta}_{\boldsymbol{f}} \leq \boldsymbol{\Delta}$, $\boldsymbol{\nabla}_{\boldsymbol{f}} \leq \boldsymbol{\nabla}$, and $\mathrm{cong}_G(\boldsymbol{f}) \leq 1$.

Given a flow $\boldsymbol{f}$, the *residual graph* $G_{\boldsymbol{f}}$ contains for each $e = (u,v) \in E$ a *forward* edge $\overrightarrow{e} = (u,v)$ with capacity $\boldsymbol{c}_{\boldsymbol{f}}(\overrightarrow{e}) \stackrel{\text{def}}{=} \boldsymbol{c}_G(e) - \boldsymbol{f}(e)$ if $\boldsymbol{c}_{\boldsymbol{f}}(\overrightarrow{e}) > 0$ and a *backward* edge $\overleftarrow{e} = (v,u)$ with capacity $\boldsymbol{c}_{\boldsymbol{f}}(\overleftarrow{e}) \stackrel{\text{def}}{=} \boldsymbol{f}(e)$ if $\boldsymbol{c}_{\boldsymbol{f}}(\overleftarrow{e}) > 0$. An *augmenting path* is a path in $G_{\boldsymbol{f}}$ consisting of edges with positive residual capacities from an unsaturated source s (i.e., $\boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0$) to an unsaturated sink t (i.e., $\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$).

A flow $\boldsymbol{f}$ is $1/z$*-integral* where $z \in \mathbb{N}$ if each $\boldsymbol{f}(e)$ is an integral multiple of z, i.e., $\boldsymbol{f} \in (1/z) \cdot \mathbb{N}^E$. Similarly, a demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is $1/z$*-integral* if $\boldsymbol{\Delta}, \boldsymbol{\nabla} \in (1/z) \cdot \mathbb{N}^E$.

Lemma 2.1 (Flow rounding [KP15])**.** *Given an integral flow $\boldsymbol{f}$ in an m-edge graph G routing an integral demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with $\boldsymbol{\Delta}, \boldsymbol{\nabla} \in z \cdot \mathbb{N}^V$ where $z \in \mathbb{N}$, there is a deterministic $O(m \log m)$ time algorithm that computes an integral $\boldsymbol{f}'$ with $\boldsymbol{f}' \leq \lceil \boldsymbol{f}/z \rceil$ routing $(\boldsymbol{\Delta}/z, \boldsymbol{\nabla}/z)$.*

2.2 Weighted Push-Relabel

We will use the *weighted push-relabel* algorithm from [BBST24] as a black box (see [BBST24, Section 4]). Recall that the classic push-relabel algorithm up to height h finds a flow f such that the residual graph has no augmenting paths of length h; the running time is $\tilde{O}(mh)$ because an edge is processed every time it moves up a layer. The weighted push-relabel algorithm takes as additional input a weight function $\boldsymbol{w}$ and finds a flow such that every augmenting path P has $\boldsymbol{w}(P) \leq 3h$. The key advantage of using weights is that an edge of weight $\boldsymbol{w}(e)$ only has to be processed every $\boldsymbol{w}(e)$ layers, so each edge contributes only $h/\boldsymbol{w}(e)$ to the running time.

Theorem 2.2 (Weighted Push-Relabel [BBST24])**.** *Suppose we have a maximum flow instance $\mathcal{I} = (G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla})$ consisting of an n-vertex m-edge directed graph $G = (V, E)$, integral edge capacities $\boldsymbol{c} \in \mathbb{N}^E$, and integral demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$. Additionally, suppose we have a weight function $\boldsymbol{w} \in \mathbb{N}^E$ and height parameter $h \in \mathbb{N}$. Then there is an algorithm* WeightedPushRelabel$(G, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla}, \boldsymbol{w}, h)$ *that in $O\left(\left(m + n + \sum_{e \in E} \frac{h}{\boldsymbol{w}(e)}\right) \log n\right)$ time finds a feasible integral flow $\boldsymbol{f}$ such that*

(i) *the $\boldsymbol{w}$-distance in the residual graph $G_{\boldsymbol{f}}$ between any unsaturated source s ($\boldsymbol{\Delta}_{\boldsymbol{f}}(s) > 0$) and any unsaturated sink t ($\boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$) is at least $\operatorname{dist}^{\boldsymbol{w}}_{G_{\boldsymbol{f}}}(s,t) > 3h$, and*

(ii) *the average $\boldsymbol{w}$-length of the flow is $\frac{\boldsymbol{w}(\boldsymbol{f})}{|\boldsymbol{f}|} \le 9h$.*

(iii) *$\boldsymbol{f}$ is a $\frac{1}{6}$-approximation of $\boldsymbol{f}^*_{\boldsymbol{w},h}$—the optimal (not necessarily integral) flow with average $\boldsymbol{w}$-length $\frac{\boldsymbol{w}(\boldsymbol{f}^\star_{\boldsymbol{w},h})}{|\boldsymbol{f}^\star_{\boldsymbol{w},h}|} \le h$.*

2.3 Crucial Concepts: Hierarchies, Expanders, and Shortcuts

Levels and Hierarchy. Let $\mathsf{level} : E \to \{1, \dots, L\}$ be a *level function* of G. Define $E_i \stackrel{\text{def}}{=} \{e \mid \mathsf{level}(e) = i\}$ as a set of *level-i edges*. Let $E_{\le i}, E_{\ge i}, E_{<i}$ and $E_{>i}$ be defined in a natural way.

Definition 2.3. Given a level function level of G, a *level-i component* of G is a strongly connected component in $G \setminus E_{>i}$. Let $\mathcal{C}_i$ contain all level-i components of G. A *hierarchy* of G induced by level is $\mathcal{H} = \{\mathcal{C}_0, \dots, \mathcal{C}_L\}$.

Note that the vertex sets of components in the hierarchy $\mathcal{C}$ over all levels form a laminar family where $\mathcal{C}_0 = \{\{v\}\}_{v \in V}$ consists of singletons and $\mathcal{C}_L$ consists of SCCs of G. When we refer to a hierarchy $\mathcal{H}$, we implicitly assume that the level function inducing $\mathcal{H}$ is given to us too. Next, we define a vertex ordering that respects a hierarchy.

Definition 2.4. A vertex ordering $\boldsymbol{\tau}_{\mathcal{H}} \in [n]^V$ *respects* a hierarchy $\mathcal{H}$ of G if the following holds.

1. For every component $C \in \mathcal{H}$, the image $\boldsymbol{\tau}(C)$ of C is contiguous in $[n]$.
2. If u are v are in different level-i components but u can reach v in $G \setminus E_{>i}$, then $\boldsymbol{\tau}_{\mathcal{H}}(u) < \boldsymbol{\tau}_{\mathcal{H}}(v)$.

Note that such an ordering can be easily computed in $O(mL)$ time (see [BBST24, Lemma 5.3]).

(Component-Constrained) Expansion. Rather than the usual definition of expanders in terms of cuts, we define them in terms of flow. Intuitively, a set of terminal edges $F \subseteq E$ is expanding if any unit demand on F can be routed in G with low congestion.

Formally, we say that a demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with $\boldsymbol{\Delta}(V) \le \boldsymbol{\nabla}(V)$ *respects* $\boldsymbol{d} \in \mathbb{R}^V_{\ge 0}$ if $\boldsymbol{\Delta}(v), \boldsymbol{\nabla}(v) \le \boldsymbol{d}(v)$ for all vertices $v \in V$. Let $\mathcal{C}$ be a collection of disjoint vertex sets. We say that $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is *$\mathcal{C}$-component-constrained* if $\boldsymbol{\Delta}(S) \le \boldsymbol{\nabla}(S)$ for all sets $S \in \mathcal{C}$.[6]

Definition 2.5. An edge set F is *$(\boldsymbol{w}, h)$-length ϕ-expanding* in G where $\boldsymbol{w} \in \mathbb{R}^{E(G)}_{\ge 0}$ are edge weights if every $\mathbf{vol}_F$-respecting demand is routable in G via a $\boldsymbol{f}$ with congestion at most $1/\phi$ and average length $\frac{\boldsymbol{w}(\boldsymbol{f})}{|\boldsymbol{f}|} \le h$.[7] The edge set is *$h$-hop ϕ-expanding* if it is *$(\mathbf{1}, h)$-length ϕ-expanding*, and simply *ϕ-expanding* if the hop/length constraint is dropped. An edge set F is *$\mathcal{C}$-component-constrained ϕ-expanding* if the same holds for $\mathbf{vol}_F$-respecting $\mathcal{C}$-component-constrained demand.

[6] Ideally we want to route sources in S to sinks in S, and a demand being $\mathcal{C}$-component-constrained means that this is not vacuously made impossible by simply not having enough sink capacity in S.

[7] Equivalently, $\boldsymbol{f}$ can be decomposed into a collection of flow paths such that the (weighted) average length of these flow paths is bounded by h. Note that this is not the same as the *length-constrained expansion* recently proposed in the literature (e.g., [HRG22, HHS23, HHL+24]). For comparison, this notion if the average-length version of the *router* defined in [HHL+24], which is a simpler-to-define object than *length-constrained expanders* in [HHL+24] and literature.

Intuitively, if F is $\mathcal{C}$-component-constrained expanding, then any $\mathbf{vol}_F$-respecting demand between vertices inside the same component S of $\mathcal{C}$ can be routed with low congestion. Note that the flow routing the demand inside each component S can go out of S, but the total congestion for *simultaneously* routing the demand of all components $S \in \mathcal{C}$ must be low.

Weak Expander Hierarchies. We use a weaker version of the expander hierarchy defined in [BBST24]. We start with a brief intuition. If the entire edge set E is ϕ-expanding in G then the hierarchy only has one level. If not, we want a smaller set of edges E_2 such that all the components of $G \setminus E_1$ are (weakly) ϕ-expanding in G. We then want an even smaller set E_3 such that inside each component of $G \setminus E_3$, the edges of E_2 are ϕ-expanding. This is formalized as follows.

Definition 2.6 (Weak Expander Hierarchy)**.** A hierarchy $\mathcal{H}$ of G is *ϕ-expanding* if, for every level $i \in \{1, \ldots, L\}$, the level-i edge set E_i is $[G \setminus E_{>i}]$-component-constrained ϕ-expanding in G, where $[G \setminus E_{>i}]$ refers to the set of SCCs in $G \setminus E_{>i}$.

In contrast to Definition 2.6, the strong hierarchy of [BBST24] instead required that each level-i component C is itself an expander. That is, $E_i \cap C$ is ϕ-expanding in C.[8]

Shortcut of Hierarchy. The key difference between our algorithm and that of [BBST24] is the use of shortcuts. These are new edges that we add to the graph to speed up flow computations; see Section 3.2 for more intuition.

Definition 2.7. (Star Edges) For any edge set $F \subseteq E$, the *star* A_F *on* F is a star whose root is a new Steiner vertex r and leaves are the tail of all edges in F.[9] The edges between the root and leaves are bidirectional. We say that A_F has *capacity scale* $\psi \in 1/\mathbb{N}$ if the edges between the root r and the endpoint of $e \in F$ have capacity $\psi \cdot \boldsymbol{c}(e)$. If there are multiple edges in F that have the same tail we avoid parallel edges in A_F by simply combining those into single edges by summing up the capacities.

Definition 2.8 (Shortcut of Hierarchy)**.** Given a hierarchy $\mathcal{H}$ of G, the *shortcut* A induced by $\mathcal{H}$ is a collection of stars defined as follows. For each level-i component $C \in \mathcal{C}_i$, let A_C be the star on $E_i \cap C$. Let $A_i \stackrel{\text{def}}{=} \bigcup_{C \in \mathcal{C}_i} A_C$ denote a *level-i shortcut* and $A \stackrel{\text{def}}{=} \bigcup_{1 \le i \le L} A_i$. We say that A has capacity scale ψ if every star A_C has capacity scale ψ. A *shortcut graph* $G_A \stackrel{\text{def}}{=} G \cup A$ is obtained by augmenting the shortcut A to G.

Definition 2.9. Given a shortcut A induced by a hierarchy $\mathcal{H}$ of some subgraph $G' \subseteq G$ (with $V(G') = V(G)$), define the *weight function* $\boldsymbol{w}$ of G_A as follows.

- For each original edge $e = (u, v) \in E$, set $\boldsymbol{w}_{\mathcal{H}}(e) \stackrel{\text{def}}{=} |\boldsymbol{\tau}_{\mathcal{H}}(u) - \boldsymbol{\tau}_{\mathcal{H}}(v)|$ where $\boldsymbol{\tau}_{\mathcal{H}}$ is a vertex ordering induced by $\mathcal{H}$ (recall Definition 2.4).
- For each shortcut edge $e \in A_C$ in component C, set $\boldsymbol{w}_{\mathcal{H}}(e) \stackrel{\text{def}}{=} |V(C)|$.

[8] The hierarchy of [BBST24] also included a set of "DAG edges" D. We do not need this terminology, but loosely speaking, the DAG edges in our hierarchy correspond to the edges in E_i that cross the SCCs of $G \setminus E_{>i}$.

[9] The choice of tail or head of edges is arbitrary.

Note that to define the weight function $\boldsymbol{w}_{\mathcal{H}}$, the hierarchy $\mathcal{H}$ does not need to include all edges in G. In this case, the weight of each $e \in G \setminus G'$ is still well-defined since $\mathcal{H}$ still induces an ordering $\boldsymbol{\tau}_{\mathcal{H}}$ of $V(G)$. Also observe that any level-i edge $e = (u, v)$ that moves *backward* in the vertex order $\boldsymbol{\tau}_{\mathcal{H}}$ (i.e., $\boldsymbol{\tau}_{\mathcal{H}}(u) > \boldsymbol{\tau}_{\mathcal{H}}(v)$ is contained in some level-i component.

Observation 2.10. *For any $(u, v) \in E_i$ with $\boldsymbol{\tau}_{\mathcal{H}}(u) > \boldsymbol{\tau}_{\mathcal{H}}(v)$ there exists a $C \in \mathcal{C}_i$ such that $u, v \in C$.*

3 Technical Overview

For simplicity of presentation, we assume during this overview that the input graph is unit-capacitated, in which case the congestion of $\boldsymbol{f}$ is simply $\text{cong}(\boldsymbol{f}) \stackrel{\text{def}}{=} \max_{e \in E} \boldsymbol{f}(e)$. Note that it suffices to design a constant-approximate flow algorithm for directed graphs, as the exact algorithm then follows by repeating the approximate algorithm $O(\log n)$ times on the residual graph.

3.1 High-Level Framework Borrowed from [BBST24]

We first describe the high-level framework of [BBST24] that our algorithm also follows.

Weight Function. Using Theorem 2.2 (weighted push-relabel from [BBST24]), computing an approximate flow in $\tilde{O}(n^2)$ time reduces to computing a "good" weight function $\boldsymbol{w}$ that satisfies: (1) there exists an approximate max flow $\boldsymbol{f}$ for which all flow paths P have $\boldsymbol{w}(P) = \tilde{O}(n)$ (this allows us to set $h = \tilde{O}(n)$ in Theorem 2.2) and (2) $\sum_e 1/\boldsymbol{w}(e) = \tilde{O}(n)$.

Note that if G is a DAG with topological order $\boldsymbol{\tau}$, then $\boldsymbol{w}(u, v) \stackrel{\text{def}}{=} |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$ is a good weight function: *all* paths have $\boldsymbol{w}(P) \leq n$ and $\sum_e 1/\boldsymbol{w}(e) = \tilde{O}(n)$ because the weights incident to any vertex v form a harmonic sum. Applying Theorem 2.2 yields a remarkably simple approximate max-flow algorithm for DAGs, as noted by [BBST24].

For general graphs, we follow [BBST24] in defining $\boldsymbol{w}$ based on an expander hierarchy $\mathcal{H}$: let $\boldsymbol{\tau}$ be any vertex ordering that respects $\mathcal{H}$ (see Definition 2.4) and set $\boldsymbol{w}(u, v) \stackrel{\text{def}}{=} |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$. Observe that we again have $\sum_e 1/\boldsymbol{w}(e) = \tilde{O}(n)$ for the same reason as in a DAG.

The Three Key Steps. At a high level, our algorithm follows three basic steps of [BBST24].

1. Prove that the weigh function $\boldsymbol{w}$ defined by an expander hierarchy $\mathcal{H}$ is a good weight function.
2. Show how to efficiently construct an $\mathcal{H}$ using an expander decomposition subroutine.
3. Show how to compute expander decomposition with respect to edge set E_i using weighted push-relabel with weight function defined by the partial hierarchy $E_1, \ldots, E_i$ computed so far.

Below, we describe in Section 3.2 how we introduce *shortcuts* into the framework of [BBST24] and then explain how they significantly simplify (and improve) the three steps above in Sections 3.3.1 to 3.3.3, respectively.

3.2 Our Key Contribution: Shortcutting Expanders

Much of the technical difficulty of [BBST24] comes from the need to keep track of the interactions between levels of the hierarchy. We show that it is possible to avoid much of this interaction by adding a set of *new* edges (and a new Steiner vertex) to the graph, denoted *shortcuts*, which simplify the graph topology and speed up flow computations.

The starting point is the following observation: as long as we only add shortcuts to edge sets that are already expanding in G, they will not significantly change the flow structure of the graph.

Observation 3.1. *Let F be a set of terminal edges that is ϕ-expanding in G. Then, adding a star A_F on F with capacity scale $\psi = \phi/\text{polylog}(n)$ (Definition 2.7) has minimal impact on the flow structure of G: formally, for any feasible flow $\boldsymbol{f}'$ in $G \cup A_F$, there exists a flow $\boldsymbol{f}$ in G with congestion $1 + 1/\text{polylog}(n)$.*

Proof. In $\boldsymbol{f}'$ the total flow on edges in A_F is at most $\psi|A_F| = 2\psi|F|$. If we instead try to route this flow inside G properly, the resulting flow instance has at most $\psi \deg_F(v)$ supply/demand on any vertex v. Since F is expanding in G, this flow can be routed with a congestion of $\tilde{O}(\psi/\phi) = 1/\text{polylog}(n)$. □

Shortcutting an Expander Hierarchy. In line with the above observation, we will add shortcuts to the expanders at every level of our hierarchy $\mathcal{H}$. Formally, letting $\mathcal{H}$ be a ϕ-expander hierarchy (Definition 2.6), for every level i and every SCC C in $G \setminus E_{>i}$, we add a star on $G[C] \cap E_i$ with capacity scale $\psi = \phi/\text{polylog}(n)$. For the rest of the overview, we refer to A as the total set of shortcuts and define $G_A \stackrel{\text{def}}{=} G \cup A$. Note that the final flow we compute will be in $G \cup A$; we discuss in Section 3.4 how to transform this into a flow on G.

3.3 Improvements and Simplifications from Using Shortcuts

The addition of shortcuts ends up significantly simplifying many seemingly unrelated parts of the algorithm of [BBST24], as well as improving the running time from $n^{2+o(1)}$ to $\tilde{O}(n^2)$. At a high level, shortcuts on $G[C] \cap E_i$ provide a trivial way to send a moderate amount of flow between those edges without needing to interact with other levels of the hierarchy. In this section, we outline how we use shortcuts to solve the three key steps described in Section 3.1.

3.3.1 Weak Expansion and Shortcuts

The authors of [BBST24] show that given a *strong* ϕ-expander hierarchy $\mathcal{H}$ (without shortcuts), the corresponding weight function $\boldsymbol{w}_{\mathcal{H}}(u, v) \stackrel{\text{def}}{=} |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$ is a good one—that is, there exists an approximate max flow $\boldsymbol{f}$ such that all flow paths P have $\boldsymbol{w}(P) = \tilde{O}(n/\phi)$. Unfortunately, this claim is false if $\mathcal{H}$ is a weak expander hierarchy.

We briefly outline the proof of [BBST24] to see why a strong hierarchy is needed. They show that to bound $\boldsymbol{w}_{\mathcal{H}}(P)$ for a path P in a flow $\boldsymbol{f}$—and hence to prove that $\boldsymbol{w}_{\mathcal{H}}$ is a good weight function—it suffices to prove the following *short-flow property*: for any level i component C (Definition 2.3), $\boldsymbol{f}$ uses at most $\tilde{O}(1/\phi)$ edges in $C \cap E_i$. They prove the short-flow property by exploiting the fact that since $C \cap E_i$ is expanding in C (because $\mathcal{H}$ is an expander hierarchy), the aggregate flow through C can be rerouted to use few edges in E_i. This rerouting needs to be done carefully to avoid blowing up congestion on lower levels. More importantly, the argument crucially requires the

fact that rerouting a level i component does not add any flow on $E_{>i}$, since otherwise those higher levels might once again violate the short-flow property. This is true in a strong expander hierarchy because C is an expander in $G \setminus E_{>i}$. In a weak expander hierarchy, where the rerouting might use any edge in G, the whole argument breaks down.

We now sketch the proof that adding shortcuts leads to a good weight function even for a weak hierarchy. The main advantage of this, as we will see in the next section, is that constructing a weak hierarchy is much easier. Our weight function $\boldsymbol{w}_{\mathcal{H}}(u,v) \stackrel{\text{def}}{=} |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$ is the same as in [BBST24] for edges in G; for every level i-component C, we set the star shortcuts on C to have weight $|C|$.

The argument that $\boldsymbol{w}$ satisfies the short-flow property is quite straightforward: unlike in [BBST24], we can handle each flow path P separately rather than carefully rerouting aggregate instances. Consider any flow path P in the current flow and any level i component C; for simplicity assume P has one unit of flow. Let $e_1, ..., e_q$ be the edges of P in $C \cap E_i$. If $q < 2/\psi = \tilde{O}(1/\phi)$ then P already satisfies the short-flow property. Otherwise, let $E_{\text{first}} \stackrel{\text{def}}{=} \{e_1, ..., e_{1/\psi}\}$ and $E_{\text{last}} \stackrel{\text{def}}{=} \{e_{q-1/\psi+1}, ..., e_q\}$. We "leak" flow from E_{first} to E_{last} through the star edges in the natural way: reduce the flow on all edges in $P[e_1, e_q]$ by ψ and instead add ψ flow on the star edge from e_1 to r and r to e_q; then reduce the flow on all edges in $P[e_2, e_{q-1}]$ by ψ and add ψ flow on the star edge from e_2 to r and r to e_{q-1}. Repeating this for $1/\psi$ steps, the only edges with non-zero flow on $P \cap C \cap E_i$ will be the $2/\psi = \tilde{O}(1/\phi)$ edges of E_{first} and E_{last}. It is not hard to check the above procedure obeys the capacities of star edges. Note that the flow of all other edges only decreases, so there is no risk of violating the short-flow property for other levels; this is why a weak expander hierarchy suffices once we add shortcuts.[10]

3.3.2 Bottom-Up Construction of Weak Expander Hierarchy

As mentioned in the three key steps of Section 3.1, both our algorithm and that of [BBST24] construct the hierarchy via the following expander decomposition subroutine: given a partial hierarchy $E_1, ..., E_i$, either certify that E_i is expanding (in which case we have a complete hierarchy) or compute a set of edges E_{cut} such that $E_i \setminus E_{\text{cut}}$ is expanding in $G \setminus E_{\text{cut}}$.[11] In this section, we assume black-box access to such a subroutine and show that computing a strong expander hierarchy is still extremely challenging, while computing a weak one is trivial. The main advantage of shortcuts is thus that they allow us settle for a weak hierarchy.

Say that we have already computed a partial hierarchy $E_1, ..., E_i$, where E_i might not be expanding in G. It is tempting to run the expander decomposition subroutine, and simply set $E_{i+1} \leftarrow E_{\text{cut}}$. If we were in the lucky case where $E_{i+1} \subseteq E_i$, then this would indeed work (we would then set $E_i \leftarrow E_i \setminus E_{\text{cut}}$). Unfortunately, one can construct examples where the expander decomposition *must* pick E_{cut} that uses edges from $E_{<i}$, which causes serious complications. Consider some level $j < i$ and some level-j SCC C inside which E_j is expanding. Say that E_{cut} contains some edges in C. If we set $E_{i+1} \leftarrow E_{\text{cut}}$, the edges of E_{cut} are added to $E_{>j}$, so C might split into SCCs C_1, C_2. But even though E_j was expanding in C, there is no guarantee that is it expanding in C_1 or C_2: routing a flow-instance in C_1 might require using edges from C_2, and vice versa.

[10] In fact, observe that our argument does not use the expansion properties of $\mathcal{H}$ at all. The only reason we need expansion is to guarantee that adding shortcuts does not significantly alter the flow structure of the graph (Observation 3.1)

[11] The *strong* expander decomposition of [BBST24] requires $E_i \setminus E_{\text{cut}}$ to be expanding in $G \setminus E_{\text{cut}}$, while our *weak* one, which is simpler to compute, only needs it to be expanding in G.

To summarize, when constructing a strong expander hierarchy in a bottom-up fashion, fixing expansion properties at level i can actually ruin the properties at level $j < i$. As a result, the algorithm of [BBST24] could not proceed in a direct bottom-up fashion, and instead needed to repeatedly move up and down between levels to fix issues caused in the previous step. Ensuring a small number of total moves required significant algorithmic adjustment and was the most technically involved aspect of the entire paper (see [BBST24, Section 7]). It was also the reason behind the $n^{o(1)}$ factor in their running time: the total number of moves was exponential in the number of layers in the hierarchy, so they could only afford a hierarchy with $o(\log n)$ layers.

By contrast, if we are willing to settle for a weak expander hierarchy, then the natural bottom-up construction easily goes through: we simply set $E_{i+1} \leftarrow E_{\text{cut}}$ and remove all edges in $E_{\leq i} \cap E_{\text{cut}}$ from their respective sets. No further changes to the lower levels are needed and we can proceed to finding the next set E_{i+2}. In particular, consider the component C from the previous paragraph: although E_j might not be expanding in $G[C_1]$ or $G[C_2]$ individually, we know by definition of expander hierarchy that any demand respecting $G[C] \cap E_j$ can be routed in the whole graph G, and hence the same goes for any demands respecting $G[C_1] \cap E_j$ and $G[C_2] \cap E_j$.

3.3.3 Computing a Single Level of the Hierarchy

We now briefly discuss our approach to the expander decomposition subroutine from the previous section. Our high-level approach mimics that of [BBST24]. By standard cut-matching games, computing an expander decomposition boils down to repeatedly solving the following: we are given some supply/demand on E_j and want to either find a large flow or a small cut. We do so by running weighted push-relabel (Theorem 2.2) with the weight function $\boldsymbol{w}$ defined by the lower levels of the hierarchy $E_1, \ldots, E_{j-1}$. If the flow $\boldsymbol{f}$ computed by weighted push-relabel is large, then we are done. If the flow $\boldsymbol{f}$ is small, we need to return a small cut.

As in regular push-relabel, we compute a small cut by exploiting the property that the residual graph with respect to flow $\boldsymbol{f}$ contains no augmenting paths of small $\boldsymbol{w}$-weight, so the vertices can be partitioned into many layers L_k. We then argue that one of these layers has few edges crossing it in the residual graph and hence yields a small cut. We analyze each hierarchy-level i separately and argue that most layers L_k have few crossing edges from E_i. Observe that by definition of expander hierarchy, all edges in E_i are in some SCC C of $G \setminus E_{>i}$. We prove two key properties:

1. The vertices of each $E_i \cap C$ have small diameter—that is, for any $(u, u'), (v, v') \in E_i \cap C$, $\text{dist}_{\boldsymbol{w}}(u, v)$ is small—and hence occupy only a small number of layers L_k.

2. Observe that push-relabel augments down the computed flow $\boldsymbol{f}$ and so reverses a small number of paths going through C. We thus also need to prove that there exists a small "pruned set" $P_C \subseteq E_i$ such that the remaining vertices in each $(E_i \cap C) \setminus P_C$ have a small diameter even after the path reversals.[12]

Our proofs of the two properties above diverge significantly from those of [BBST24]. In [BBST24], although it is easy to argue that there exists a (u, v)-path with few edges in E_i (because E_i is expanding in C), bounding the weight of lower-level edges requires a much more careful analysis. The path reversals of Property 2 pose an even bigger challenge. Intuitively, since C is an expander, one can prove the existence of P_C by using expander pruning. But even though the number of reversed flow paths is small, these paths can have many edges, so standard expander pruning will

[12] Note that we only only need to prove the *existence* of P_C.

not give sufficiently strong bounds. Thus [BBST24] needed to develop an entirely new expander pruning subroutine that can handle path reversals and is also sensitive to distances under $\boldsymbol{w}_{\mathcal{H}}$.

By contrast, the presence of shortcuts leads to a much simpler analysis. Recall that every component C now comes with a star root r_C and star edges incident to every edge $e \in E_i \cap C$. The diameter of $E_i \cap C$ is thus trivially small because one can use the path $(u, r_C) \circ (r_C, v)$. To handle path reversals, we observe that any simple path will use only two star edges, so only a small number of star edges will be saturated by the flow paths. As such, except for a small set P_C all other edges in $E_i \cap C$ will still be incident to an (unsaturated) bidirectional star edge in the residual graph and thus still have small diameter. This proof sketch again highlights the way in which star edges allow us to largely avoid analyzing the interaction between different levels of the hierarchy.

3.4 Unfolding the Shortcuts

While greatly simplifying most of the algorithm, shortcuts do also add a new element of complexity, as the algorithm computes a flow $\boldsymbol{f}'$ in G_A instead of the original graph G. By an extension of Observation 3.1, we can show that there must exist a flow $\boldsymbol{f}$ of almost the same value, but we still need to compute this flow. Fortunately, this only needs to be done at the very end of the algorithm, by which point we have computed a lot of useful information that allows us to compute $\boldsymbol{f}$ using straightforward techniques. Loosely speaking, given an SCC of $G \setminus E_{>i}$, the flow on the shortcut edges of the star on $G[C] \cap E_i$ defines a flow instance respecting E_i in C. Since each component C is a (weak) expander with respect to E_i, all of these flow instances can be routed in G with low congestion. This routing is easy to obtain, again, via the weighted push-relabel algorithm. This leverages the fact that we have already proven when computing the expander hierarchy that these demands can be routed not just with low congestion, but also with short paths. Iterating this for all levels in the hierarchy top-down, we map the flow in G_A back to a flow in G with small congestion blow-up. Finally, we run a standard flow rounding to convert it into the desired feasible approximate flow.

4 Weighted Push-Relabel on Shortcut Graphs

Below, in Corollary 4.2, we show that by running the weighted push-relabel algorithm of [BBST24, Algorithm 1] we can in $\tilde{O}(n^2)$ time find a constant-approximate maximum flow (and approximate minimum cut) on the shortcut graph G_A with capacity scale $\frac{1}{\text{polylog}(n)}$. To this end, we prove a more general version, Lemma 4.1, that will later prove useful in the construction of the expander hierarchy. In this version, we have an additional edge set F not part of the hierarchy.

This should be compared to [BBST24, Theorem 6.1]. However, note that our proof is significantly simpler due to the addition of shortcuts. In fact, in contrast to [BBST24, Theorem 6.1] where the given hierarchy is required to be expanding, such a condition is not even needed for our Lemma 4.1 to work. Instead, Lemma 4.1 gives us a flow in G_A not G, and the expansion of the hierarchy is used only when we are "unfolding" the flow back to G, which happens after running the weighted push-relabel algorithm.

Lemma 4.1 (Weighted Push-Relabel on Shortcut Graphs)**.** *Let $F \subseteq E$ and $\mathcal{H}$ be the hierarchy of $G \setminus F$ with $L = O(\log n)$ levels. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$, and $G_A \stackrel{\text{def}}{=} G \cup A$ be the shortcut graph. Given a diffusion instance $\mathcal{I} = (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with ψ-integral demand and parameter $\kappa \in \mathbb{N}$, there is a deterministic algorithm that runs on G_A in*

$O(n^2 \log^3 n \cdot (\kappa + 1/\psi))$ time and finds a ψ-integral flow $\boldsymbol{f}$ in G_A with congestion κ and average weight $\frac{\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f})}{|\boldsymbol{f}|} < O(n \log n \cdot (\kappa + 1/\psi))$.

Additionally, if $|\boldsymbol{f}| < \|\boldsymbol{\Delta}\|_1$, then the algorithm also returns a cut $\emptyset \neq S \subsetneq V(G_A)$ with $\boldsymbol{\nabla}_{\boldsymbol{f}}(S) = 0$ and $\boldsymbol{\Delta}_{\boldsymbol{f}}(\overline{S}) = 0$ of size

$$\boldsymbol{c}(E_{G_A}(S, \overline{S})) \leq \frac{41|\boldsymbol{f}| + \min\{\mathbf{vol}_F(S), \mathbf{vol}_F(\bar{S})\}}{\kappa}.$$

Recall from Definition 2.9 that even though the hierarchy $\mathcal{H}$ is only on $G \setminus F$, the weights $\boldsymbol{w}_{\mathcal{H}}(e)$ for $e \in F$ are still well-defined from the vertex ordering induced by $\mathcal{H}$. Before proving the more general version Lemma 4.1, we show how it directly implies Corollary 4.2 in the case when $F = \emptyset$ and $\kappa = 1$.

Corollary 4.2. *Let $\mathcal{H}$ be a hierarchy of G. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$, and $G_A = G \cup A$ be the shortcut graph. There is a deterministic algorithm that, given G_A and $s, t \in V(G)$, in $O(n^2 \log^3 n/\psi)$ time finds an $O(1)$-approximate maximum (s,t)-flow $\boldsymbol{f}$ and an $O(1)$-approximate minimum (s,t)-cut S in G_A.*

Proof. This follows from Lemma 4.1 when $\kappa = 1$, $F = \emptyset$, and $\mathcal{I} = (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is a flow instance for routing a (s,t)-flow. More precisely, $\boldsymbol{\Delta}(s) = n^3 U$ where U is a maximum capacity of G and $\boldsymbol{\Delta}(v) = 0$ for all $v \neq s$. Similarly, $\boldsymbol{\nabla}(t) = n^3 U$ and $\boldsymbol{\nabla}(v) = 0$ for all $v \neq t$. Since this demand cannot be fully routed, Lemma 4.1 must return a flow $\boldsymbol{f}$ with congestion κ and a cut S, such that $\boldsymbol{c}(E_{G_A}(S, \overline{S})) \leq 41|\boldsymbol{f}|$. Moreover, $s \in S$ and $t \notin S$, since the residual demands satisfy $\boldsymbol{\nabla}_{\boldsymbol{f}}(S) = 0 = \boldsymbol{\Delta}_{\boldsymbol{f}}(\overline{S})$ and $\boldsymbol{\Delta}_{\boldsymbol{f}}(s), \boldsymbol{\nabla}_{\boldsymbol{f}}(t) > 0$. Hence $\boldsymbol{f}$ together with the cut $(S, \overline{S})$ must be 41-approximate maximum flow respectively 41-approximate minimum cut in G_A. The running time follows from Lemma 4.1. □

In the remainder of this section, we prove Lemma 4.1.

Weighted Push-Relabel of [BBST24]. Our approach to prove Lemma 4.1 is similar to the corresponding sparse-cut algorithm of [BBST24, Theorem 6.1, Algorithm 2]. Because we run on the shortcut graph G_A, instead of the original graph G with an directed expander hierarchy as in [BBST24], our proofs here will be significantly simpler,[13] and have less overhead in the running time.

The Algorithm. Let $M \stackrel{\text{def}}{=} \sum_{e \in F} \boldsymbol{c}(e)$ be the total capacity in F. We set

$$h \stackrel{\text{def}}{=} \left\lceil n \cdot \left(\frac{6L}{\psi} + 100\kappa \log M \right) \right\rceil \tag{1}$$

as the target height, and scale up all the capacities (but not the demands) by a factor of κ. We then run the weighted push-relabel algorithm (Theorem 2.2) with weights $\boldsymbol{w}_{\mathcal{H}}$ on G_A to compute the flow $\boldsymbol{f}$, which by Theorem 2.2(ii) satisfies $\frac{\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f})}{|\boldsymbol{f}|} \leq 9h$ as required.[14] In case we did not route

[13] This section replaces most of the analysis in [BBST24, Section 5 and 6]. We still rely on the rather short weighted push-relabel algorithm [BBST24, Section 4].

[14] Since Theorem 2.2 requires the capacities to be integer, and the short-cut edges have capacity scale ψ, we scale up all capacities and demands before calling the weighted push-relabel by $\frac{1}{\psi} \in \mathbb{N}$, and then scale down the returned flow. The scaled down flow will have congestion κ with respect to the original capacities $\boldsymbol{c}$, and might no longer be integral.

all of the demand we also need to output a cut. Similar to [BBST24], we modify the edge lengths slightly to $\boldsymbol{w_f}$ by setting many of the edge weights in the residual graph to 0. In particular, we set $\boldsymbol{w_f}(\overrightarrow{e}) = 0$ for any edge $e = (u, v) \in E \setminus F$ (notably this excludes edges in F, short-cut edges in A, and reversed edges formed in the residual graph) that moves forward in the vertex order associated with $\mathcal{H}$ (i.e., $\boldsymbol{\tau}_{\mathcal{H}}(u) < \boldsymbol{\tau}_{\mathcal{H}}(v)$) . In the residual graph (ignoring edges with zero residual capacity), we calculate the $\boldsymbol{w_f}$-distance for each vertex from its closest unsaturated source, and let S_i be the corresponding distance layers. The main part of the analysis is to show that one of these distance-layer cuts $(S_{\le i}, \overline{S_{\le i}})$ must satisfy the output-guarantee of Lemma 4.1.

Algorithm 1: SPARSECUT$(\mathcal{I} = (G, F, \mathcal{H}, \boldsymbol{c}, \boldsymbol{\Delta}, \boldsymbol{\nabla}), \kappa, F, \mathcal{H})$

Let $h \stackrel{\text{def}}{=} \left\lceil n \cdot \left(\frac{6L}{\psi} + 100\kappa \log M\right)\right\rceil$, and $\boldsymbol{c}^{\kappa} \stackrel{\text{def}}{=} \kappa \cdot \boldsymbol{c}$.

Run WEIGHTEDPUSHRELABEL$(G_A, \frac{1}{\psi} \cdot \boldsymbol{c}^{\kappa}, \frac{1}{\psi} \cdot \boldsymbol{\Delta}, \frac{1}{\psi} \cdot \boldsymbol{\nabla}, \boldsymbol{w}, h)$ (Theorem 2.2) to get a flow $\boldsymbol{f}'$.

Set $\boldsymbol{f} \stackrel{\text{def}}{=} \psi \cdot \boldsymbol{f}'$. `// f is ψ-integral`

if $|\boldsymbol{f}| = \|\boldsymbol{\Delta}\|_1$ **then return** $\boldsymbol{f}$

else

  Let $\boldsymbol{w_f}$ be $\boldsymbol{w}$ extended to $(G_A)_{\boldsymbol{f}}$, except set $\boldsymbol{w_f}(\overrightarrow{e}) \stackrel{\text{def}}{=} 0$ for all edges $e \in E(G) \setminus F$ moving forward in the vertex order $\boldsymbol{\tau}_{\mathcal{H}}$ (i.e., $e = (u, v)$ such that $\boldsymbol{\tau}_{\mathcal{H}}(u) < \boldsymbol{\tau}_{\mathcal{H}}(v)$).

  Let $S_0 = \{s \in V : \boldsymbol{\Delta_f}(s) > 0\}$.

  Compute $\boldsymbol{w_f}$-distance levels $S_i \stackrel{\text{def}}{=} \left\{v \in V : \text{dist}^{\boldsymbol{w_f}}_{G_{\boldsymbol{f}}}(S_0, v) = i\right\}$ in the residual graph $G_{\boldsymbol{f}}$.

  return $\boldsymbol{f}$ and the cut $(S_{\le i}, \overline{S_{\le i}})$ minimizing $\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(E_{G_{\boldsymbol{f}}}(S_{\le i}, \overline{S_{\le i}})) - \min\{\mathbf{vol}_F(S_{\le i}), \mathbf{vol}_F(\overline{S_{\le i}})\}$.

Running Time. Note that G_A has at most $n + nL$ vertices and $m + nL$ edges, since there are L layers in the hierarchy $\mathcal{H}$, and in each layer there is at most one added star edge per vertex. The weight function $\boldsymbol{w}_{\mathcal{H}}$ can be computed in $O(mL)$ time, and the distance layers can be computed in $O((m + nL) \log n)$ time by Dijkstra's shortest-paths algorithm. By Theorem 2.2, the running time of the WEIGHTEDPUSHRELABEL() call is:

$$O\left(\left(m + n + \sum_{e \in E(G_A)} \frac{h}{\boldsymbol{w}_{\mathcal{H}}(e)}\right) \log n\right) = O\left(h \log n \sum_{e \in E(G_A)} \frac{1}{\boldsymbol{w}_{\mathcal{H}}(e)}\right)$$
$$= O\left(h \log n \left(\sum_{e \in A} \frac{1}{\boldsymbol{w}_{\mathcal{H}}(e)} + \sum_{e \in E(G)} \frac{1}{\boldsymbol{w}_{\mathcal{H}}(e)}\right)\right)$$

Since each edge $e \in A$ has $\boldsymbol{w}(e) \ge 1$, we have that $\sum_{e \in A} \frac{1}{\boldsymbol{w}(e)} = O(nL)$, as there are at most nL star edges. The term with $E(G)$ can be bounded similarly as [BBST24, Claim 5.5] by harmonic sums, since their weights are induced by a permutation $\boldsymbol{\tau}_{\mathcal{H}}$:

$$\sum_{e \in E(G)} \frac{1}{\boldsymbol{w}(e)} \le \sum_{\substack{\boldsymbol{\tau}_{\mathcal{H}}(u), \boldsymbol{\tau}_{\mathcal{H}}(v) \in [n] \\ \boldsymbol{\tau}_{\mathcal{H}}(u) \ne \boldsymbol{\tau}_{\mathcal{H}}(v)}} \frac{1}{|\boldsymbol{\tau}_{\mathcal{H}}(u) - \boldsymbol{\tau}_{\mathcal{H}}(v)|} = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j - i} = O(n \log n).$$

We thus see that the total running time of the algorithm is $O((m+nL)\log n + hn(L+\log n)\log n)$. Plugging in the value of h from (1), the running time can be bounded by

$$O\left(n^2 \cdot \left(\frac{L}{\psi} + \kappa \log M\right) \cdot (L + \log n) \cdot \log n\right).$$

Since $L = O(\log n)$ and $M \leq n^4$, the running time becomes $O(n^2 \log^3 n \cdot (\kappa + 1/\psi))$.

Finding a Sparse Cut. In case $|\boldsymbol{f}| = ||\boldsymbol{\Delta}||_1$, we are done, so let us focus on the case of $|\boldsymbol{f}| < ||\boldsymbol{\Delta}||_1$, where we need to show that the returned cut satisfies the output guarantee of Lemma 4.1.

Theorem 2.2 guarantees that the $\boldsymbol{w}_{\mathcal{H}}$-length of any remaining augmenting path is at least $3h$ in the residual graph. The following lemma, which can be proved in exactly the same way as [BBST24, Lemma 6.3], certifies that even after setting some edge weights to 0 to obtain $\boldsymbol{w}_{\boldsymbol{f}}$, the $\boldsymbol{w}_{\boldsymbol{f}}$-length of any augmenting path is still at least h.

Lemma 4.3. *If* $\mathrm{dist}^{\boldsymbol{w}_{\mathcal{H}}}_{(G_A)_{\boldsymbol{f}}}(S_0, v) > 3h$, *then* $\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{(G_A)_{\boldsymbol{f}}}(S_0, v) > h$.

Proof Sketch. On any path P, all edges (u, v) going forward in the vertex order $\boldsymbol{\tau}_{\mathcal{H}}$ (recall new weight is $\boldsymbol{w}_{\boldsymbol{f}}(u, v) = 0$; old weight was $\boldsymbol{w}_{\mathcal{H}}(u, v) = |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$) can be charged to increasing the location in the vertex order, whose total increase is at most n plus the amount we move backwards in the vertex order (and all edges (u, v) moving backwards have weight $\boldsymbol{w}_{\boldsymbol{f}}(u, v) = \boldsymbol{w}_{\mathcal{H}}(u, v) \geq |\boldsymbol{\tau}(u) - \boldsymbol{\tau}(v)|$). Hence $\boldsymbol{w}_{\mathcal{H}}(P) \geq \boldsymbol{w}_{\boldsymbol{f}}(P) \geq \frac{\boldsymbol{w}_{\mathcal{H}}(P) - n}{2}$, and the lemma follows since $h > n$. □

By the output guarantee of Theorem 2.2 together with Lemma 4.3, we thus know that among distance layers $S_0, S_1, \ldots S_h$ there is no unsaturated sink. That is, we know that $\boldsymbol{\Delta}_{\boldsymbol{f}}(\overline{S_0}) = 0$ and for all $0 \leq i \leq h$ that $\boldsymbol{\nabla}_{\boldsymbol{f}}(S_i) = 0$.

Definition 4.4 (Good Cuts)**.** For $0 \leq i < h$, the distance layer cut $(S_{\leq i}, \overline{S_{\leq i}})$ is *good* if the capacity of this cut, in the residual graph,[15] disregarding edges in F, is at most $40|\boldsymbol{f}|$, i.e., if

$$\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(E_{(G_A)_{\boldsymbol{f}}}(S_{\leq i}, \overline{S_{\leq i}}) \setminus F) \leq 40|\boldsymbol{f}|.$$

We will prove that there are at least $h/4$ good cuts. If $F = \emptyset$, this means that the flow $\boldsymbol{f}$ together with such a good cut are both constant-approximate. When $F \neq \emptyset$, assuming we have at least $h/4$ good cuts, one can instead finish the proof with a standard ball growing argument.

We begin by showing that (large) components at a certain level at the hierarchy remain mostly intact and close together in the residual graph. The following lemma, which has a very short proof due to the construction of the shortcuts, is to be contrasted with the corresponding [BBST24, Lemma 6.5] that required a much longer analysis.

Lemma 4.5. *Consider a level ℓ of the hierarchy $\mathcal{H}$ and a level-ℓ component C. Let $E_C \stackrel{\text{def}}{=} E_\ell \cap C$, and let r be the Steiner vertex of the star A_{E_C}. There exists a subset $P_C \subseteq E_C$ such that*

(1) for each $e \in E_C \setminus P_C$, we have $\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{(G_A)_{\boldsymbol{f}}}(r, \mathrm{tail}(e)) \leq |C|$ and $\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{(G_A)_{\boldsymbol{f}}}(\mathrm{tail}(e), r) \leq |C|$; and

(2) $\boldsymbol{c}^{\kappa}(P_C) \leq \frac{2|\boldsymbol{f}|}{\psi}$.

[15] Recall that we set $\boldsymbol{c}^{\kappa} = \kappa \cdot \boldsymbol{c}$. and that in the residual graph $(G_A)_{\boldsymbol{f}}$ each edge $e = (u, v)$ has a corresponding forward edge $\overrightarrow{e} = (u, v)$ with capacity $\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(\overrightarrow{e}) \stackrel{\text{def}}{=} \boldsymbol{c}^{\kappa}(e) - \boldsymbol{f}(e)$ and a backward edge $\overleftarrow{e} = (v, u)$ with capacity $\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(\overleftarrow{e}) \stackrel{\text{def}}{=} \boldsymbol{f}(e)$.

Proof. Recall that in G_A there is a bidirectional star A_{E_C} added between the tails of edges in E_C to r, where all these star edges have a ψ-fraction of the capacity of the corresponding edges in E_C. Let $P' \subseteq A_{E_C}$ be the subset of the (directed) edges of this star that are saturated by the flow $\boldsymbol{f}$, i.e., edges e where $\boldsymbol{f}(e) = \boldsymbol{c}^\kappa(e)$. The total capacity of P' is at most $2|\boldsymbol{f}|$, since every flow path passes through the star at most once.[16] Let P_C be all edges in E_C whose tail is incident to an edge in P'. In particular, $\boldsymbol{c}^\kappa(P_C) \leq \frac{2|\boldsymbol{f}|}{\psi}$. Moreover, all tails of edges $e \in E_C \setminus P_C$ are of distance at most $|C|$ from the star center r in the residual graph, in both directions, since the corresponding star edges are not saturated and have $\boldsymbol{w}$-length $|C|$ (hence $\boldsymbol{w}_{\boldsymbol{f}}$-length $|C|$). □

We are now ready to show that at least a constant fraction of our distance layer cuts are good.

Lemma 4.6. *There are at least $h/4$ good cuts.*

Proof. Consider one of our candidate level cuts $(S_{\leq i}, \overline{S_{\leq i}})$ where $0 \leq i < h$, and what edges can cross it in the residual graph $(G_A)_{\boldsymbol{f}}$.

1. Backward edges $\overleftarrow{e}$. These have residual capacities $\boldsymbol{c}^\kappa_{\boldsymbol{f}}(\overleftarrow{e}) = \boldsymbol{f}(e)$. The contribution of these (across all good level cuts) can be bounded by $\sum_{\overleftarrow{e} \in \overleftarrow{E}} \boldsymbol{c}^\kappa_{\boldsymbol{f}}(\overleftarrow{e})\boldsymbol{w}_{\boldsymbol{f}}(e) = \sum_{e \in E} \boldsymbol{f}(e)\boldsymbol{w}_{\boldsymbol{f}}(e) = \boldsymbol{w}_{\boldsymbol{f}}(\boldsymbol{f}) \leq 9h \cdot |\boldsymbol{f}|$ by Theorem 2.2(ii) and that $\boldsymbol{w}_{\boldsymbol{f}} \leq \boldsymbol{w}$.

2. Forward edges $\overrightarrow{e}$, where $e = (u, v)$ is a level-ℓ edge with $\boldsymbol{\tau}_{\mathcal{H}}(u) > \boldsymbol{\tau}_{\mathcal{H}}(v)$ that is contained in a level-ℓ component C of $\mathcal{H}$, and the corresponding star edges (which exist by Observation 2.10). We will use Lemma 4.5 to show that most level cuts only have few of these edges.

The remaining edges are either in F (which we disregard for in the definition of good cuts), or are edges e where we set $\boldsymbol{w}_{\boldsymbol{f}}(e) = 0$, as they move forward in $\boldsymbol{\tau}_{\mathcal{H}}$ (and hence cannot cross our distance layer cuts).

For each level ℓ of $\mathcal{H}$, and level-ℓ component C, let P_C be the pruned set in Lemma 4.5, and let $E'_C \stackrel{\text{def}}{=} (E_\ell \cap C) \setminus P_C$. Let $A_{E'} \subseteq A_{(E_\ell \cap C)}$ be the shortcut edges (in the star of the level-ℓ component C) that are not incident to P_C. Call a cut $(S_{\leq i}, \overline{S_{\leq i}})$ *bad* if any edge $e \in E'_C \cup A_{E'_C}$ crosses it, for any level ℓ and level-ℓ component C. First we argue that there are at most $h/2$ bad cuts. In particular, we argue that some level-ℓ component C can only be responsible for $3|C|$ bad cuts (where $|C|$ counts the number of vertices in this component). Indeed by Lemma 4.5, each (tail) of an edge in E'_C is of distance at most $|C|$ from the Steiner vertex r at the center of the shortcut star on $E_\ell \cap C$. Let i_r be the layer such that $S_{i_r} \ni r$. The earliest layer any edge in E'_C can start in is in layer $i_r - |C|$, and the latest layer any edge in E'_C can end is in layer $i_r + |C| + |C|$ (the tail of the edge can be at layer $i_r + |C|$, and then the length of this edge can be another $|C|$ layers). Thus all edges in E'_C, and their corresponding star edges, must be within $3|C|$ consecutive distance layers.

For a fixed level ℓ, all level-ℓ components sum up to n in size, and hence they can be responsible for at most $3n$ bad cuts in total. Summing over the L layers, there are at most $3nL \leq h/2$ bad cuts.

Now, there are at least $h/2$ remaining non-bad cuts. We will argue that half of them are good. The total capacity of edges over these cuts is at most $9h \cdot |\boldsymbol{f}|$ from the backwards edges. Each edge in P_C has length at most $|C|$. Also, as $\psi \leq 1$, the corresponding star edges connected to P_C contribute at most the same amount (in terms of capacity) as the edges in P_C. Hence, if we add the

[16] This uses the fact that the weighted push-relabel of [BBST24] works by repeatedly pushing flow along simple augmenting paths. If an augmenting path increases the flow value by a, then this augmenting path can be responsible for at most $2a$ reduction in the capacity of edges in A_{E_C}, as the path passes through the center r at most once.

contribution from the edges in P_C (and the corresponding star edges) for all levels ℓ and level-ℓ components C, we have, by Lemma 4.5:

$$\begin{aligned}
&\sum_{\substack{0\le i\le h \text{ such that}\\ (S_{\le i},\overline{S_{\le i}}) \text{ is not bad}}} \boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(E_{(G_A)_{\boldsymbol{f}}}(S_{\le i},\overline{S_{\le i}})) \\
&\qquad \le 9h|\boldsymbol{f}| + \sum_{\ell=1,\dots,L}\ \sum_{\text{level-}\ell\text{ component } C} |C|\cdot 2\boldsymbol{c}^{\kappa}(P_C) \\
&\qquad \le 9h|\boldsymbol{f}| + n\cdot L\cdot \frac{4|\boldsymbol{f}|}{\psi} \\
&\qquad \le 10h|\boldsymbol{f}|.
\end{aligned}$$

The last inequality comes from our choice of h in (1), $h \ge \frac{4nL}{\psi}$. By an averaging argument, at most half of the non-good cuts have capacity more than $2\cdot\frac{10h|\boldsymbol{f}|}{h/2} = 40|\boldsymbol{f}|$. Hence, at least $h/4$ cuts have capacity, in the residual graph and disregarding F, at most $40|\boldsymbol{f}|$, i.e., they are good cuts. □

In case $F=\emptyset$ (as in Corollary 4.2), we would already be done. When $F\neq\emptyset$, what remains is to argue the contribution of these edges via a standard ball-growing argument stating that at least one of our $h/4$ good cuts $(S_{\le i},\overline{S_{\le i}})$ has capacity, in F, at most $\min\{\mathbf{vol}_F(S_{\le i}),\mathbf{vol}_F(\overline{S_{\le i}})\}$. We do this in Lemma 4.7 below, and note that our main Lemma 4.1 follows almost directly from this.

Lemma 4.7. *There exists a level cut $S_{\le i}$ with $0\le i<h$ such that*

$$\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}(E_{(G_A)_{\boldsymbol{f}}}(S_{\le i},\overline{S_{\le i}})) \le 40|\boldsymbol{f}| + \min\{\mathbf{vol}_F(S_{\le i}),\mathbf{vol}_F(\overline{S_{\le i}})\}. \tag{2}$$

Proof. Let $S_{\le i_1},\dots,S_{\le i_g}$ be the $g\ge h/4$ many good level cuts given by Lemma 4.6. Let $U_j \stackrel{\text{def}}{=} S_{\le i_{j\cdot n}}\setminus S_{\le i_{(j-1)\cdot n}}$ for each $1\le j\le \lfloor\frac{g}{n}\rfloor$. Let $k\stackrel{\text{def}}{=}\lfloor\frac{g}{n}\rfloor\ge\frac{g}{2n}$. That is, we first split the distance levels into g blocks at $i_1,\dots,i_g$, and then merge every n consecutive blocks to form the U_j's. Observe that since $i_{j\cdot n}\ge i_{(j-1)\cdot n}+n$, we must have

$$\mathrm{dist}^{\boldsymbol{w}_{\boldsymbol{f}}}_{(G_A)_{\boldsymbol{f}}}(U_j,U_{j+2}) > n \tag{3}$$

for every j. We will now only consider level cuts that are between some U_j and U_{j+1} and bound the contribution of edges from F to them using a ball-growing argument. Note that if $\mathbf{vol}_F(U_{\le 1})=0$ or $\mathbf{vol}_F(\overline{U_{\le k}})=0$, then the lemma is vacuously true, and therefore we assume otherwise. We show that there exists a $1\le j\le k$ such that

$$\boldsymbol{c}^{\kappa}_{\boldsymbol{f}}\left(E_{(G_A)_{\boldsymbol{f}}}(U_{\le j},\overline{U_{\le j}})\cap F\right) \le \min\{\mathbf{vol}_F(U_{\le j}),\mathbf{vol}_F(\overline{U_{\le j}})\}, \tag{4}$$

which proves the lemma. Assume for contradiction that none of the U_j satisfies (4). Because of (3) and that the weight of any edge is bounded by n we know that all edges in $E_{(G_A)_{\boldsymbol{f}}}(U_{\le j},\overline{U_{\le j}})$ with positive capacities must be in $E_{(G_A)_{\boldsymbol{f}}}(U_j,U_{j+1})$. Let $\mathbf{vol}^{\kappa}_F(S)\stackrel{\text{def}}{=}\kappa\cdot\mathbf{vol}_F(S)$. If $\mathbf{vol}_F(U_{\le k/2})\le\mathbf{vol}_F(\overline{U_{\le k/2}})$ then we have

$$\mathbf{vol}^{\kappa}_F(U_{\le j}) \ge \mathbf{vol}^{\kappa}_F(U_{\le j-1}) + \boldsymbol{c}^{\kappa}_{\boldsymbol{f}}\left(E_{(G_A)_{\boldsymbol{f}}}(U_{\le j},\overline{U_{\le j}})\cap F\right)$$

for $1 \le j \le k/2$ which with the assumption of (4), and that $\mathbf{vol}_F^\kappa(U_{\le 1}) = \kappa \cdot \mathbf{vol}_F(U_{\le 1}) \ge \kappa$, implies that

$$\mathbf{vol}_F^\kappa(U_{\le j}) \ge \left(1+\frac{1}{\kappa}\right)\mathbf{vol}_F^\kappa(U_{\le j-1}) \implies \mathbf{vol}_F^\kappa(U_{\le k/2}) \ge \left(1+\frac{1}{\kappa}\right)^{k/2-1}\kappa > 2^{(k/2-1)/\kappa}\kappa \ge 2\kappa M.$$

The last inequality above uses $k/2-1 \ge k/4 \ge \frac{h}{16n}$ (since by Lemma 4.6 we have $g \ge h/4$) and, by (1), that $h \ge 100n\kappa \log M$, where M is the total capacity of edges in the input graph. This is a contradiction because $\mathbf{vol}_F^\kappa(S)$ of any S should always be bounded by $2\kappa M$.

Similarly, if $\mathbf{vol}_F(U_{\le k/2}) > \mathbf{vol}_F(\overline{U_{\le k/2}})$, a symmetric argument gives

$$\mathbf{vol}_F^\kappa(\overline{U_{\le j}}) \ge \left(1+\frac{1}{\kappa}\right)\mathbf{vol}_F^\kappa(\overline{U_{\le j+1}}) \implies \mathbf{vol}_F^\kappa(\overline{U_{\le k/2+1}}) \ge \left(1+\frac{1}{\kappa}\right)^{k/2-1}\kappa$$

In both cases we have arrived at a contradiction, proving the lemma. □

We end the section by finishing Lemma 4.1.

Proof of Lemma 4.1. Note that the capacity, in the residual graph $(G_A)_{\boldsymbol{f}}$, of any cut can be at most $|\boldsymbol{f}|$ smaller than the capacity of the same cut in the original graph, i.e., $\boldsymbol{c}^\kappa(E_{(G_A)}(S_{\le i}, \overline{S_{\le i}})) \le \boldsymbol{c}_{\boldsymbol{f}}^\kappa(E_{(G_A)_{\boldsymbol{f}}}(S_{\le i}, \overline{S_{\le i}})) + |\boldsymbol{f}|$. Hence, we get $\boldsymbol{c}(E_{G_A}(S,\overline{S})) \le \frac{41|\boldsymbol{f}|+\min\{\mathbf{vol}_F(S),\mathbf{vol}_F(\bar{S})\}}{\kappa}$ when combined with Lemma 4.7, which concludes the proof of Lemma 4.1. □

5 Expander Hierarchies: Single-Pass Bottom-Up Construction

The goal of this section is to construct in $\tilde{O}(n^2)$ time a weak expander hierarchy $\mathcal{H}$ (see Definition 2.6) of G that is $\tilde{\Omega}(1)$-expanding. In fact, we show something stronger: we give a data structure that can, for any flow in the shortcut graph G_A, computationally "unfold" it to a corresponding flow in G of approximately the same value. This is useful in the final maximum flow algorithm in Section 6. This section proves the following Lemma 5.1, with $\phi_{\text{rand}} = \Omega(1/\log^7 n)$ being the universal parameter defined later in Lemma 5.2.

Lemma 5.1. *There is a randomized algorithm that, given an n-vertex graph G, in $O(n^2 \log^{18} n)$ time computes*

- *a hierarchy $\mathcal{H}$ of G with $L = O(\log n)$ levels,*
- *a shortcut A induced by $\mathcal{H}$ with capacity scale $\psi_{\text{rand}} = \Omega(\phi_{\text{rand}}/\log n)$, and*
- *a "flow-unfolding" data structure $\mathcal{D}_{\text{unfold}}$ such that, with high probability, given any ψ_{rand}-integral feasible flow $\boldsymbol{f}_A$ in $G_A \stackrel{\text{def}}{=} G \cup A$, $\mathcal{D}_{\text{unfold}}$ returns a flow $\boldsymbol{f} \stackrel{\text{def}}{=} \mathcal{D}_{\text{unfold}}(\boldsymbol{f}_A)$ in G routing the same vertex-demand with congestion 3 in $O(n^2 \log^{15} n)$ time.*

To formally see why the flow-unfolding data structure of Lemma 5.1 implies that $\mathcal{H}$ is $\tilde{\Omega}(1)$-expanding, consider any level i and the level-i edge set E_i in $\mathcal{H}$. Any $[G \setminus E_{i+1}]$-component-constrained $\mathbf{vol}_{E_i}$-respecting demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ can be routed in G_A with congestion $1/\psi_{\text{rand}} = \tilde{O}(1)$ by trivially routing through the shortcut edges. Denote this flow by $\boldsymbol{f}_A$. Then, Lemma 5.1 can map $\boldsymbol{f}_A$ to

a flow $\boldsymbol{f}$ in G with congestion $\tilde{O}(1)$. This shows that E_i is $[G \setminus E_{i+1}]$-component-constrained $\tilde{\Omega}(1)$-expanding in G for every i. Thus, $\mathcal{H}$ is indeed $\tilde{\Omega}(1)$-expanding according to Definition 2.6.[17]

We now describe on a high level how the weak expander hierarchy is constructed.

Dynamics of Levels and Hierarchies. We will construct the expander hierarchy in $L = \lceil \log_{10/9} \boldsymbol{c}(E) \rceil + 1 = O(\log n)$ rounds. In each round r, we have a level function $\mathsf{level}^{(r)} : E \to \{1, \ldots, r\}$. We define $E_i^{(r)} \stackrel{\text{def}}{=} \{e : \mathsf{level}^{(r)}(e) = i\}$ as the set of round-r level-i edges. In particular, for round 1, we initialize $\mathsf{level}^{(1)}(e) \stackrel{\text{def}}{=} 1$ for all edges e and therefore $E = E_1^{(1)}$.

In round $r \geq 1$, we will choose an edge set $E_{r+1}^{(r+1)}$ where $\boldsymbol{c}(E_{r+1}^{(r+1)}) \leq 0.9 \cdot \boldsymbol{c}(E_r^{(r)})$, and then set their level to $r+1$. That is, $\mathsf{level}^{(r+1)}(e) \stackrel{\text{def}}{=} r+1$ if $e \in E_{r+1}^{(r+1)}$ and the level of other edges remains the same: $\mathsf{level}^{(r+1)}(e) \stackrel{\text{def}}{=} \mathsf{level}^{(r)}(e)$ if $e \notin E_{r+1}^{(r+1)}$. In particular, we have $E_i^{(r+1)} = E_i^{(r)} \setminus E_{r+1}^{(r+1)}$ for all $i \leq r$.

We call $E_r^{(r)}$ the set of *top-level edges* in round r. We will build objects for the graph induced by non-top-level edges. Let $\mathcal{H}^{(r)} = \{\mathcal{C}_0^{(r)}, \ldots, \mathcal{C}_{r-1}^{(r)}\}$ be the hierarchy of $G \setminus E_r^{(r)}$ induced by $\mathsf{level}^{(r)}$ (restricted to non-top-level edges $E \setminus E_r^{(r)}$). Let $A^{(r)}$ be a shortcut induced by $\mathcal{H}^{(r)}$ with capacity scale ϕ_{rand}/L. Note that $A^{(r)}$ does not include the stars on the top-level edges $E_r^{(r)}$. For example, $A^{(1)} = \emptyset$ and $\mathcal{H}^{(1)} = \{\mathcal{C}_0^{(1)}\}$, where $\mathcal{C}_0^{(1)}$ are singleton components. Denote the round-r shortcut graph by $G_{A^{(r)}} \stackrel{\text{def}}{=} G \cup A^{(r)}$.

Observe that, for all $i < r$, $E_i^{(r)} \subseteq E_i^{(r-1)}$ and the family $\mathcal{C}_i^{(r)}$ of level-i components of $\mathcal{H}^{(r)}$ is a refinement of $\mathcal{C}_i^{(r-1)}$ because $E_{>i}^{(r)} \supseteq E_{>i}^{(r-1)}$. In words, as the number of rounds r increases, the levels of edges only move up. For a fixed level $i < r$, the family $\mathcal{C}_i^{(r)}$ of level-i components only undergoes refinements. Hence, the stars in the level-i shortcut $A_i^{(r)}$ only keep splitting.[18] At the end, we will return $\mathcal{H}^{(L+1)}$ and guarantee that $E_{L+1} = \emptyset$ which makes $\mathcal{H}^{(L+1)}$ a hierarchy of G.

Algorithm: Expander Decomposition Once per Level. For each $r \geq 1$, once we choose $E_{r+1}^{(r+1)}$, this determines how $\mathsf{level}^{(r+1)}, \mathcal{H}^{(r+1)}, G_{A^{(r+1)}}$ can be obtained from $\mathsf{level}^{(r)}, \mathcal{H}^{(r)}, G_{A^{(r)}}$. We construct $E_{r+1}^{(r+1)}$ using the following variant of (weak) expander decomposition, which we prove in Section 7.

Lemma 5.2 (Weak Expander Decomposition)**.** *There is a universal $\phi_{\text{rand}} = \Theta(1/\log^7 n)$ where $\phi_{\text{rand}} \in 1/\mathbb{N}$ such that there exists a randomized algorithm that, given the round-r level function $\mathsf{level}^{(r)}$ and the induced shortcut graph $G_{A^{(r)}}$ with capacity scale ψ of an n-vertex graph G, computes in $O(n^2 \log^9 n \cdot (\log^3 n + 1/\psi))$ time the* round-$(r+1)$ top-level edge set $E_{r+1}^{(r+1)}$ *such that*

1. $\boldsymbol{c}(E_{r+1}^{(r+1)}) \leq 0.9 \cdot \boldsymbol{c}(E_r^{(r)})$ *and*

[17] Meticulous readers might notice that we lose an extra $O(\log n)$ factor in congestion here. This is in fact unnecessary because our Lemma 5.2 only have a congestion blow-up of $1/\phi_{\text{rand}}$. That said, what is described here is an easy way to see how the flow-unfolding data structure is a stronger construct.

[18] In fact, our algorithm does not actually split the stars $A_i^{(r)}$. Instead, we maintain L different versions of shortcut graphs that are fixed once constructed, and delegate each level of routing to its corresponding shortcut graph.

2. $E_r^{(r+1)}$ *is* $[G \setminus E_{r+1}^{(r+1)}]$*-component-constrained*[19] $\Omega(1/\log^6 n)$*-expanding in* $G_{A^{(r)}}$*, with high probability.*

Furthermore, conditioned on Item 2 succeeding, given a $[G \setminus E_{r+1}^{(r+1)}]$*-component-constrained* $(\mathbf{vol}_{E_r^{(r+1)}} \cdot \kappa/z)$*-respecting* $1/z$*-integral demand* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ *for some* $\kappa, z \in \mathbb{N}$*, there is an* $O(n^2 \log^6 n \cdot (\log^3 n + 1/\psi))$ *time algorithm that returns a flow* $\boldsymbol{f}$ *routing* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ *in* $G_{A^{(r)}}$ *such that* $\boldsymbol{f}(e) \leq \frac{\lceil \mathbf{c}(e) \cdot \kappa/\phi_{\text{rand}} \rceil}{z}$.

Note crucially that the value of ϕ_{rand} does not depend on the scale of the hierarchy, and thus there is no circular dependence; see its proof in Section 7.

Remark 5.3. The above Lemma 5.2 is a form of expander decomposition. For example, in the first round when $r = 1$, we have $A^{(r)} = \emptyset$ and $E_1^{(1)} = E$. The lemma then computes a set $R \stackrel{\text{def}}{=} E_2^{(2)}$, of capacity $\boldsymbol{c}(R) \leq 0.9 \cdot \boldsymbol{c}(E)$, such that $E_1^{(2)} \stackrel{\text{def}}{=} E \setminus R$ is $[G \setminus R]$-component-constrained expanding in $G_{A^{(1)}} = G$. That is, the algorithm identifies a not-too-large set of edges R, after whose removal all strongly connected components of G are expanders.

Note that as in our new weighted push-relabel algorithm on shortcut graphs, we do not need the expansion of the hierarchy $\mathcal{H}^{(r)}$ to guarantee that $E_r^{(r+1)}$ is expanding in $G_{A^{(r)}}$. The expansion of $\mathcal{H}^{(r)}$ is used later when guaranteeing that $E_r^{(r+1)}$ is expanding in the *original graph* G or, equivalently, when proving the flow-unfolding data structure.

Given Lemma 5.2, the algorithm for computing our final expander hierarchy is very simple. See Algorithm 2.

Algorithm 2: A bottom-up construction of an expander hierarchy.

Initialize $\mathsf{level}^{(1)}(e) \stackrel{\text{def}}{=} 1$ for all edge $e \in E$.
Set $L \stackrel{\text{def}}{=} \lceil \log_{10/9} \boldsymbol{c}(E) \rceil + 1$.
for $r = 1, \ldots, L$ **do**
  Let $\mathcal{H}^{(r)}$ be the hierarchy of $G \setminus E_r^{(r)}$ induced by $\mathsf{level}^{(r)}$.
  Let $A^{(r)}$ be the shortcut induced by $\mathcal{H}^{(r)}$ with capacity scale $\psi_{\text{rand}} \stackrel{\text{def}}{=} \phi_{\text{rand}}/L$. Let $G_{A^{(r)}} \stackrel{\text{def}}{=} G \cup A^{(r)}$.
  Run Lemma 5.2 on $G_{A^{(r)}}$ and compute the round-$(r+1)$ top-level edge set $E_{r+1}^{(r+1)}$.
  Set $\mathsf{level}^{(r+1)}(e) \stackrel{\text{def}}{=} \begin{cases} r+1 & e \in E_{r+1}^{(r+1)}, \\ \mathsf{level}^{(r)}(e) & \text{otherwise.} \end{cases}$
Return the last hierarchy $\mathcal{H}^{(L+1)}$ and the shortcut $A^{(L+1)}$.

Analysis. We now analyze Algorithm 2. Let $z \stackrel{\text{def}}{=} 200L/\psi_{\text{rand}} \in \mathbb{N}$ be the integrality of the flows that we will work with. Note that we are being deliberately explicit about the integrality of flows as our push-relabel algorithm works with integral demands and flows (which up to scaling by a $\text{poly}(n)$ factor works for $1/\text{poly}(n)$-integral demands and flows). We first show that we can do a single level of unfolding by incurring a small congestion. For intuition one should think of the following

[19] Recall that this is defined to be $\mathcal{C}$-component-constrained for $\mathcal{C} \stackrel{\text{def}}{=} \text{SCC}(G \setminus E_{r+1}^{(r+1)})$, which is not necessarily the same as that of $G_{A^{(r)}} \setminus E_{r+1}^{(r+1)}$.

lemma as incurring a multiplicative $(1+1/L)$ congestion (while the extra $0.1/L$ factor stems from rounding to preserve integrality) in each level of unfolding, and thus the final congestion in G is $(1+1/L)^L = O(1)$.

Lemma 5.4. *Given a $1/z$-integral flow $\boldsymbol{f}$ in $G_{A^{(r+1)}}$ routing a demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ supported on V with congestion κ/z for integer $\kappa \geq z$, we can compute in $O(n \log^{14} n)$ time a $1/z$-integral flow $\boldsymbol{f}'$ in $G_{A^{(r)}}$ routing the same demand with congestion κ'/z where $\kappa' \in \mathbb{N}$ satisfies $\kappa' \leq \left(1 + \frac{1.01}{L}\right)\kappa$.*

Proof. Let $A_r^{(r+1)}$ be the set of level-r stars of the shortcut $A^{(r+1)}$. For each strongly connected component C of $G \setminus E_{r+1}^{(r+1)}$, there is a star A_C in $A_r^{(r+1)}$ whose leaves are vertices in the component C, with a bidirectional edge to each leaf v of capacity $\deg_{E_r^{(r+1)} \cap C}(v) \cdot \phi_{\text{rand}}/L = \deg_{E_r^{(r+1)} \cap C}(v) \cdot \psi_{\text{rand}}$. Let $\boldsymbol{f}^C$ be the flow $\boldsymbol{f}$ restricted to the (bidirectional) edges of the star A_C. Since $\boldsymbol{f}$ has congestion κ/z, the vertex-demand $(\boldsymbol{\Delta}^C, \boldsymbol{\nabla}^C)$ of $\boldsymbol{f}^C$ satisfies

$$\boldsymbol{\Delta}^C(v), \boldsymbol{\nabla}^C(v) \leq \frac{\psi_{\text{rand}}\kappa}{z} \cdot \deg_{E_r^{(r+1)} \cap C}(v) \leq \frac{\tilde{\kappa}}{z} \cdot \deg_{E_r^{(r+1)} \cap C}(v),$$

where $\tilde{\kappa} \stackrel{\text{def}}{=} \lceil \psi_{\text{rand}}\kappa \rceil$, for all vertices v in the component C. Moreover, for the center u of the star A_C, we have $\boldsymbol{\Delta}^C(u) = \boldsymbol{\nabla}^C(u) = \boldsymbol{f}^{\text{out}}(u) = 0$ since $u \notin V$ and the original demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is supported on V. It follows that $\boldsymbol{\Delta}^C(C) = \boldsymbol{\nabla}^C(C)$.

Consider the $1/z$-integral vertex-demand $(\boldsymbol{\Delta}^+, \boldsymbol{\nabla}^+) \stackrel{\text{def}}{=} \left(\sum_C \boldsymbol{\Delta}^C, \sum_C \boldsymbol{\nabla}^C\right)$, which is both $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained and $(\mathbf{vol}_{E_r^{(r+1)}} \cdot \tilde{\kappa}/z)$-respecting by the above discussion. Invoking Lemma 5.2, we can compute in $O(n \log^{14} n)$ time a $1/z$-integral flow $\boldsymbol{f}^+$ routing $(\boldsymbol{\Delta}^+, \boldsymbol{\nabla}^+)$ in $G_{A^{(r)}}$ such that

$$\boldsymbol{f}^+(e) \leq \frac{\lceil \tilde{\kappa} \cdot \boldsymbol{c}(e)/\phi_{\text{rand}} \rceil}{z} \leq \frac{\tilde{\kappa}/\phi_{\text{rand}} + 1/\psi_{\text{rand}}}{z} \cdot \boldsymbol{c}(e),$$

where we used that $\boldsymbol{c}(e) \geq \psi_{\text{rand}}$.

Note that vertex-demand $(\boldsymbol{\Delta}^+, \boldsymbol{\nabla}^+)$ is also routed by the sum of flows $\sum_C \boldsymbol{f}^C$. It follows that $\boldsymbol{f}^+$ routes the same vertex-demand as $\sum_C \boldsymbol{f}^C$. Therefore, the flow $\boldsymbol{f}' \stackrel{\text{def}}{=} \boldsymbol{f} - \sum_C \boldsymbol{f}^C + \boldsymbol{f}^+$ routes the same vertex-demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ as $\boldsymbol{f}$ in $G' \stackrel{\text{def}}{=} G \cup A^{(r)} \cup (A^{(r+1)} \setminus A_r^{(r+1)})$, is $1/z$-integral, and satisfies

$$\begin{aligned} \boldsymbol{f}'(e) \leq \frac{\kappa + \tilde{\kappa}/\phi_{\text{rand}} + 1/\psi_{\text{rand}}}{z} \cdot \boldsymbol{c}_{G'}(e) &\leq \frac{\kappa + \kappa/L + 1/\phi_{\text{rand}} + 1/\psi_{\text{rand}}}{z} \cdot \boldsymbol{c}_{G'}(e) \\ &\leq \frac{(1 + 1.01/L)\kappa}{z} \cdot \boldsymbol{c}_{G'}(e). \end{aligned} \tag{5}$$

The last inequality in (5) follows from

$$\frac{1}{\phi_{\text{rand}}} + \frac{1}{\psi_{\text{rand}}} \leq \frac{2}{\psi_{\text{rand}}} \leq \frac{0.01}{L} \cdot \kappa$$

since $\kappa \geq z = 200L/\psi_{\text{rand}}$. What remains is to transform the flow to be supported solely on $G_{A^{(r)}} \stackrel{\text{def}}{=} G \cup A^{(r)}$. Observe that for each $1 \leq i < r$, the partition $\mathcal{C}_i^{(r+1)}$ is a refinement of the partition $\mathcal{C}_i^{(r)}$ since the only differences in $\mathsf{level}^{(r)}$ and $\mathsf{level}^{(r+1)}$ is that $\mathsf{level}^{(r+1)}$ lifts some edges to level $r+1$. Therefore, we can naturally map each edge in level-i stars of the shortcut $A^{(r+1)}$ to a corresponding edge in a level-i star of the shortcut $A^{(r)}$. Under this mapping, the new flow $\boldsymbol{f}''$ is supported on $G_{A^{(r)}}$, and its vertex-demand is still supported on V. □

Now we can prove Lemma 5.1.

Proof of Lemma 5.1. We run Algorithm 2 to construct the hierarchy $\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}^{(L+1)}$ and shortcut $A \stackrel{\text{def}}{=} A^{(L+1)}$. By Lemma 5.2, we have $\boldsymbol{c}(E_{r+1}^{(r+1)}) \le 0.9 \cdot \boldsymbol{c}(E_r^{(r)})$ for each $r \ge 1$, and since $L = \lceil \log_{9/10} \boldsymbol{c}(E) \rceil + 1$, we have $\boldsymbol{c}(E_{L+1}^{(L+1)}) \le 0.9^{\lceil \log_{10/9} \boldsymbol{c}(E) \rceil} \cdot \boldsymbol{c}(E) < 1$, which implies $E_{L+1}^{(L+1)} = \emptyset$. Then, $\mathcal{H} \stackrel{\text{def}}{=} \mathcal{H}^{(L+1)}$ is the hierarchy of $G \setminus E_{L+1}^{(L+1)} = G$, and $A \stackrel{\text{def}}{=} A^{(L+1)}$ has capacity scale ϕ_{rand}/L by construction. The running time is $O(n^2 \log^{17} n)$ per level by Lemma 5.2, hence $O(n^2 \log^{18} n)$ overall. For the flow-unfolding data structure D_{unfold}, we proceed by repeatedly unfolding a level-$(r+1)$ flow $\boldsymbol{f}^{(r+1)}$ to a level-r flow $\boldsymbol{f}^{(r)}$ in $G_{A^{(r)}}$ with Lemma 5.4, starting from $\boldsymbol{f}^{(L+1)} \stackrel{\text{def}}{=} \boldsymbol{f}$ in $G_{A^{(L+1)}}$. Initially, $\boldsymbol{f}^{(L+1)}$ has congestion $1 = z/z$, and inductively the final flow $\boldsymbol{f}^{(1)}$ routes $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in $G_{A^{(1)}} = G$ and has congestion $(1 + 1.01/L)^L \le e^{1.01} \le 3$. The running time of the flow-unfolding data structure is $O(n^2 \log^{14} n \cdot L) = O(n^2 \log^{15} n)$. This completes the proof. □

6 The Maximum Flow Algorithm

Given the algorithm for computing an approximate maximum flow in a shortcut graph from Section 4 and the algorithm for mapping the flow on the shortcut graph to the original graph without blowing up congestion too much from Section 5, we obtain an approximate maximum flow algorithm summarized as follows.

Lemma 6.1. *There is a randomized algorithm that, given a graph G and $s, t \in V(G)$ with polynomially bounded integral edge capacities $\boldsymbol{c} \in \mathbb{N}^E$, in $O(n^2 \log^{18} n)$ time finds an **integral** $O(1)$-approximate maximum (s,t)-flow $\boldsymbol{0} \le \boldsymbol{f} \le \boldsymbol{c}$ and minimum (s,t)-cut S in G with high probability.*

Note that Lemma 6.1 immediately implies our main Theorem 1.1 by repeatedly solving the problem in the residual graph up to $O(\log n)$ times (recall that we have assumed the capacities are bounded by $\text{poly}(n)$ via capacity scaling). We state the algorithm for Lemma 6.1 in Algorithm 3 and prove its correctness below.

Algorithm 3: An $O(1)$-approximate maximum (s,t)-flow and minimum (s,t)-cut algorithm.

1. Using Lemma 5.1, compute a hierarchy $\mathcal{H}$, the shortcut A induced by $\mathcal{H}$ with capacity scale ψ, and the data structure $\mathcal{D}_{\text{unfold}}$.
2. Given the shortcut graph $G_A \stackrel{\text{def}}{=} G \cup A$ and $s, t \in V(G)$ as input to Corollary 4.2, compute a ψ-integral $O(1)$-approximate maximum (s,t)-flow $\boldsymbol{f}_A$ in G_A and $O(1)$-approximate minimum (s,t)-cut S_A in G_A.
3. Set $\boldsymbol{f} \leftarrow \mathcal{D}_{\text{unfold}}(\boldsymbol{f}_A)$, scale the flow value on each edge down by a factor of $1/3$, and round $\boldsymbol{f}$ to integral using [KP15].
4. Return $(\boldsymbol{f}, S_A \cap V)$.

Correctness. We will prove that $\boldsymbol{f}$ is a *feasible* (s,t)-flow in G and $S_A \cap V$ is a (s,t)-cut in G where $\boldsymbol{c}(E_G(S_A \cap V, \overline{S_A \cap V})) \le O(|\boldsymbol{f}|)$. We have $\boldsymbol{c}(E_G(S_A, \overline{S_A})) \le \boldsymbol{c}(E_{G_A}(S_A, \overline{S_A}))$ because $G \subseteq G_A$. Next, $\boldsymbol{c}(E_{G_A}(S_A, \overline{S_A})) \le O(|\boldsymbol{f}_A|)$ because, by Corollary 4.2, $\boldsymbol{f}_A$ is a $O(1)$-approximate feasible maximum flow in G_A.

It remains to prove that $\boldsymbol{f}$ is a feasible (s,t)-flow in G and has value $|\boldsymbol{f}| = \Omega(|\boldsymbol{f}_A|)$. Indeed, by Lemma 5.1, $\mathcal{D}_{\text{unfold}}(\boldsymbol{f}_A)$ is an (s,t)-flow in G with congestion at most 3 and $|\mathcal{D}_{\text{unfold}}(\boldsymbol{f}_A)| = |\boldsymbol{f}_A|$.

After scaling the flow value on each edge by $1/3$ and rounding, $\boldsymbol{f}$ becomes integral and is a feasible (s,t)-flow in G of value $|\boldsymbol{f}| = \lceil |\boldsymbol{f}_A|/3 \rceil$.

Running Time. Lemma 5.1 takes $O(n^2 \log^{18} n)$ time. Corollary 4.2 takes $O(n^2 \log^3 n/\psi = O(n^2 \log^{11} n)$ because $\psi = \Omega(1/\log^8 n)$. The data structure call $\mathcal{D}_{\text{unfold}}(\boldsymbol{f}_A)$ takes $O(n^2 \log^{15} n)$ time. Finally, flow rounding takes $O(m \log n) = O(n^2 \log n)$ time. Overall, it takes $O(n^2 \log^{18} n)$ time to compute an $O(1)$-approximate maximum flow. This proves Lemma 6.1.

Lastly, recursively applying Lemma 6.1 $O(\log n)$ times we solve maximum flow in $O(n^2 \log^{19} n)$ time, proving Theorem 1.1.

Theorem 1.1. *There is a randomized algorithm that, given an n-vertex directed graph G with edge capacities from $\{1, \ldots, U\}$ and $s, t \in V(G)$, finds a maximum (s,t)-flow in G with high probability in $O(n^2 \log^{19} n \log U)$ time.*

7 Weak Expander Decomposition: Proof of Lemma 5.2

In this section we give an algorithm for weak expander decomposition, proving Lemma 5.2.

7.1 Cut-Matching Game

To compute a weak expander decomposition, we use the now-standard procedures of cut-matching games. The cut-matching game constructs an expander by the interaction between a cut player and a matching player: Starting from an empty graph, in each iteration, the cut player chooses a cut of the vertices, and the matching player outputs a matching between the two sides of the cut, which is then added to the graph. The guarantee of the cut-matching games is that there exist good strategies for the cut players so that after a small number of iterations, regardless of the matchings chosen by the matching player, the final graph (which equals the union of these matchings) becomes an expander.

To use this procedure to certify whether a given graph G is an expander or not (or whether a vertex weight is expanding in it), we can implement the matching player by solving a flow problem on the graph that finds a matching between the vertices embeddable with low congestion into G via flow paths. Thus, after the cut-matching game, the expander constructed by the game is embeddable into G with low congestion, thus proving the expansion of G.

The standard cut-matching game requires the matchings we choose to be perfect. We use a slight variant that only requires *almost* perfect matchings, at the cost of only guaranteeing the expansion of *almost* all vertices. This procedure has become fairly standard for undirected graphs, and its directed generalization can be proven in various ways. We describe two ways of achieving this.

(Standard) Cut-Matching Game with Pruning. For instance, one can run the standard directed *cut-matching game* (e.g., [Lou10]). Since it requires the matchings to be perfect, we add *fake* edges to our matchings to make them artificially perfect. At the end of the cut-matching game, we get an expander that contains few fake edges. We then run the *expander pruning* algorithm (e.g., [SP25]) to extract an expander subgraph supported on a large fraction of vertices and that consists of only real edges (which proves the expansion of these vertices).

Non-Stop Cut-Matching Game. Alternatively, one can use a *non-stop* version of the cut-matching game that allows removing small subsets of vertices during the interaction between the cut and the matching players [SW19, LRW25, FLL25]. That is, the matching player does not need to return a perfect matching, and vertices that are unmatched in some iterations get removed from the game. The non-stop cut-matching game is able to guarantee the expansion of all non-removed vertices at the end. Note that this approach only returns weak expanders (in the sense that the expansion of non-removed vertices is potentially certified through removed ones), while the aforementioned one based on pruning returns strong expanders.

While the first version might be more modularized for presentation, we believe the *non-stop cut-matching game* is the simpler approach (both in implementation and analysis) as it avoids the pruning step. Thus we opt for using it for our algorithm[20] (see Lemma 7.3 below).

Matching Player. In either version of the cut-matching game, we need to implement the matching player in the game by providing the cut player a (directed and capacitated) matching from some vertex set A to B. We define the following notion of *perfectness* for these matchings.

Definition 7.1. Consider a partition (P, Q) of V. A capacitated graph M on V (with capacities $\boldsymbol{c}_M$) is a (P, Q)-*matching* if (1) $\deg^-_M(v) = 0$ for all $v \in P$ and (2) $\deg^+_M(v) = 0$ for all $v \in Q$. It is $\boldsymbol{d}$-perfect for some measure $\boldsymbol{d}$ satisfying $\boldsymbol{d}(P) = \boldsymbol{d}(Q)$ if it also holds that (i) $\deg^+_M(v) = \boldsymbol{d}(v)$ for all $v \in P$ and (ii) $\deg^-_M(v) = \boldsymbol{d}(v)$ for all $v \in Q$. It is $1/z$-*integral* if the capacities $\boldsymbol{c}_M$ are $1/z$-integral.

Consider a (P, Q)-matching M. Let $\boldsymbol{d}_M \in \mathbb{R}^V$ be defined by $\boldsymbol{d}_M(v) \stackrel{\text{def}}{=} \deg^+_M(v)$ for $v \in P$ and $\boldsymbol{d}_M(v) \stackrel{\text{def}}{=} \deg^-_M(v)$ for $v \in Q$. That is, $\boldsymbol{d}_M$ is the unique measure such that M is $\boldsymbol{d}_M$-perfect.

We encapsulate what we consider a valid output of the matching player in the non-stop cut-matching game as follows (note that $\boldsymbol{d}'$ is supposed to be a subdemand of $\boldsymbol{d}$ for which we are trying to certify almost expansion).

Definition 7.2. Let (P, Q) be a partition of V and $\boldsymbol{d}'$ be a vertex measure on V with $\boldsymbol{d}'(P) = \boldsymbol{d}'(Q)$. A $(P, Q, \boldsymbol{d}', \Delta, m)$-*bidirectional-matching* is a pair of matchings $(\overrightarrow{M}, \overleftarrow{M})$ consisting of m edges such that $\boldsymbol{d}_{\overrightarrow{M}}, \boldsymbol{d}_{\overleftarrow{M}} \leq \boldsymbol{d}'$ are submeasure of $\boldsymbol{d}'$ that satisfy $\boldsymbol{d}_{\overrightarrow{M}}(V), \boldsymbol{d}_{\overleftarrow{M}}(V) \geq \boldsymbol{d}'(V) - \Delta$. The bidirectional-matching is $1/z$-*integral* if both $\overrightarrow{M}$ and $\overleftarrow{M}$ are.

We use the following directed, capacitated, non-stop cut-matching game of [FLL25] as a black box. The cut-matching game is formulated not exactly this way in [FLL25], and we discuss how our Lemma 7.3 can be extracted from [FLL25] in Section B.

Lemma 7.3 (Non-Stop CMG [FLL25])**.** *Consider an integral vertex measure $\boldsymbol{d}$ on n vertices V where $\|\boldsymbol{d}\|_\infty \leq \text{poly}(n)$. Let $z \in \mathbb{N}$. Suppose there is an oracle that on input partition (P, Q) and $1/z$-integral $\boldsymbol{d}' \leq \boldsymbol{d}$ outputs a $1/z$-integral $(P, Q, \boldsymbol{d}', \Delta, m)$-bidirectional-matching for $\Delta \stackrel{\text{def}}{=} \delta_{\text{KRV}} \cdot \boldsymbol{d}(V)$ where $\delta_{\text{KRV}} = O(1/\log^2 n)$ is sufficiently small.*

Then, there is a randomized algorithm that invokes the oracle $T_{\text{KRV}} = O(\log^2 n)$ times and takes $O(m \log^2 n)$ additional running time to compute a submeasure $\tilde{\boldsymbol{d}} \leq \boldsymbol{d}$ with $\|\tilde{\boldsymbol{d}}\|_1 \geq \frac{7}{8}\|\boldsymbol{d}\|_1$ such that with high probability $\tilde{\boldsymbol{d}}$ is T_{KRV}-hop $\Omega(1)$-expanding in $W \stackrel{\text{def}}{=} \bigcup_{i \in [k]} \overrightarrow{M}_i \cup \overleftarrow{M}_i$, where $\overrightarrow{M}_i \cup \overleftarrow{M}_i$ is the output of the oracle in the i-th invocation.

[20] In Section A—where we show how to derandomize our algorithm for the special case of vertex-capacitated graphs—we instead use the former approach of standard cut-matching games plus pruning (as the non-stop version of the cut-matching games are randomized).

7.2 Implementing the Matching Player

Note that given a partition (P, Q) of V and vertex measure $\boldsymbol{d}'$, a flow $\boldsymbol{f}$ that routes the demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ defined by $\boldsymbol{\Delta} \stackrel{\text{def}}{=} \mathbf{1}_P \cdot \boldsymbol{d}'$ and $\boldsymbol{\nabla} \stackrel{\text{def}}{=} \mathbf{1}_Q \cdot \boldsymbol{d}'$ gives an $\boldsymbol{d}'$-perfect (P, Q)-matching via path decomposition of flows: For any flow $\boldsymbol{f}$ it is standard that we can write it as $\boldsymbol{f} = C_{\boldsymbol{f}} + P_{\boldsymbol{f}}$, where C_f is a circulation[21] and the support of $P_{\boldsymbol{f}}$ is acyclic. The part $P_{\boldsymbol{f}}$ can be further decomposed into $P_{\boldsymbol{f}} = \sum_{i\in[k]} \lambda_i \boldsymbol{f}_{P_i}$ where $\lambda_i > 0$, and each $\boldsymbol{f}_{P_i}$ is the flow that sends one unit of demand along a simple (s_i, t_i)-path P_i. We call such $(C_{\boldsymbol{f}}, \{(\lambda_i, P_i)\}_{i\in[k]})$ a *path decomposition* of $\boldsymbol{f}$.

Now, for any path decomposition $(C_{\boldsymbol{f}}, \{(\lambda_i, P_i)\}_{i\in[k]})$ of a flow $\boldsymbol{f}$ routing the above demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$, the capacitated matching M that contains for each (u, v)-path P_i an edge (u, v) with capacity $\boldsymbol{c}_M(u, v) \stackrel{\text{def}}{=} \lambda_i$ is a $\boldsymbol{d}'$-perfect (P, Q)-matching. This flow perspective will be used frequently. In particular, it is easy to see that our flow algorithm Lemma 4.1 can be used as the oracle required by Lemma 7.3 when $\boldsymbol{d} = \mathbf{vol}_F$. To achieve a stronger property that we need later, we further require that the flow we found is decomposable into short paths under the weight $\boldsymbol{w}_{\mathcal{H}}$. We call a path decomposition $(C_{\boldsymbol{f}}, \{(\lambda_i, P_i)\}_{i\in[k]})$ $(\boldsymbol{w}_{\mathcal{H}}, h)$-*short* if each path P_i satisfies $\boldsymbol{w}_{\mathcal{H}}(P_i) \le h$.[22] The path decomposition is $1/z$-*integral* if each λ_i is. We encapsulate this into the following lemma. Since it can be quite straightforwardly obtained by running Lemma 4.1 multiple times and performing path decomposition to extract short paths, we defer the proof to Section C. Note that to circumvent the path decomposition barrier in capacitated graphs, instead of outputting the paths $\{P_i\}_{i\in[k]}$, we only return the *representation* $\{(\lambda_i, s_i, t_i)\}_{i\in[k]}$ (i.e., the endpoints of the paths) of the decomposition which suffices for constructing the matchings.

Lemma 7.4. *Let $F \subseteq E$ and $\mathcal{H}$ be the hierarchy of $G \setminus F$. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$, and $G_A = G \cup A$ be the shortcut graph. Given a diffusion instance $\mathcal{I} = (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with ψ-integral demand and parameter $\kappa \in \mathbb{N}$, there is a deterministic algorithm that runs on G_A in $O(n^2 \log^4 n \cdot (\kappa + 1/\psi))$ time and finds a ψ-integral flow $\boldsymbol{f}$ in G_A with congestion $O(\kappa \log n)$ and the representation of a ψ-integral $(\boldsymbol{w}_{\mathcal{H}}, h)$-short path decomposition of $\boldsymbol{f}$ for some $h = O(n \log n \cdot (\kappa + 1/\psi))$.*

Additionally, if $|\boldsymbol{f}| < \|\boldsymbol{\Delta}\|_1$, then the algorithm also returns a cut $\emptyset \neq S \subsetneq V$ with $\boldsymbol{\nabla}_{\boldsymbol{f}}(S) = 0$ and $\boldsymbol{\Delta}_{\boldsymbol{f}}(\overline{S}) = 0$ of size

$$\boldsymbol{c}(E_{G_A}(S, \overline{S})) \le \frac{41 \min\{\boldsymbol{\Delta}(S), \boldsymbol{\nabla}(\bar{S})\} + \min\{\mathbf{vol}_F(S), \mathbf{vol}_F(\bar{S})\}}{\kappa}.$$

We now use the flow subroutine of Lemma 7.4 to construct our matching player.

Lemma 7.5 (Matching Player)**.** *Let $F \subseteq E$ and $\mathcal{H}$ be a hierarchy of $G \setminus F$. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$ and $G_A \stackrel{\text{def}}{=} G \cup A$ be the shortcut graph that contains m' edges. Then, for any $\phi < 1$ and $\delta < 1$, given a partition (P, Q) of V and ψ-integral $\boldsymbol{d}' \le \boldsymbol{d}$ for $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{vol}_F$, there is a deterministic algorithm that runs in $O(n^2 \log^4 n \cdot (1/\phi + 1/\psi))$ time and computes either*

- *a balanced ϕ-sparse cut S such that $\mathrm{vol}_F(S) \in \left[\frac{\delta}{2}\mathrm{vol}_F(V), \frac{1}{2}\mathrm{vol}_F(V)\right]$, or*

[21] A *circulation* is a flow that routes the trivial demand $(\mathbf{0}, \mathbf{0})$.

[22] While our Lemma 4.1 already guarantees that the *average* weight of any path decomposition is small, we want the *worst case* weight is small as well (as required by the definition of being $(\boldsymbol{w}_{\mathcal{H}}, h)$-short). The proof of Lemma 7.4 thus essentially consists of running Lemma 4.1, taking the paths that are short, and recursing on the demand that was not routed by these short paths.

- *a ψ-integral $(P, Q, \boldsymbol{d}', \Delta, m')$-bidirectional-matching $(\overrightarrow{M}, \overleftarrow{M})$ where $\Delta \stackrel{\text{def}}{=} \delta \cdot \mathrm{vol}_F(V)$.*

In the second case, the matchings are such that, given any ψ-integral flow $\boldsymbol{f}_M$ on $M \stackrel{\text{def}}{=} \overrightarrow{M} \cup \overleftarrow{M}$ of congestion κ, there exists a ψ-integral flow $\boldsymbol{f}_{G_A}$ routing the same demand as $\boldsymbol{f}_M$ does in G_A with congestion $O(\kappa \cdot \log n/\phi)$ such that $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}_{G_A}) \leq \|\boldsymbol{f}_M\|_1 \cdot h$ for some $h = O(n \log n \cdot (1/\phi + 1/\psi))$.

Proof. Given the partition (P, Q) and $\boldsymbol{d}'$, to find $\overrightarrow{\boldsymbol{d}}$ and $\overrightarrow{M}$ we solve the flow problem defined by $\boldsymbol{\Delta} \stackrel{\text{def}}{=} \mathbf{1}_P \cdot \boldsymbol{d}'$ and $\boldsymbol{\nabla} \stackrel{\text{def}}{=} \mathbf{1}_Q \cdot \boldsymbol{d}'$ in G_A (which is ψ-integral by the input guarantee). We invoke Lemma 7.4 on with $\kappa \stackrel{\text{def}}{=} \lceil 50/\phi \rceil$ to compute a flow $\overrightarrow{\boldsymbol{f}}$ in G_A with congestion $O(\log n/\phi)$ routing $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$. If $|\overrightarrow{\boldsymbol{f}}| < \|\boldsymbol{\Delta}\|_1$, then Lemma 7.4 additionally gives us a cut S that contains all the vertices with excess, meaning that (a subset of) $\overrightarrow{\boldsymbol{f}}$ routes the demand restricted to $\bar{S}$ (note that this subset of $\boldsymbol{f}$ may still use edges outside of $G_A[\bar{S}]$.) We first show that S, if non-empty, is indeed a ϕ-sparse cut with respect to F. By Lemma 7.4, we have

$$\begin{aligned}\boldsymbol{c}(E_{G_A}(S, \bar{S})) &\leq \frac{\phi}{50} \cdot \left(41 \min\{\boldsymbol{\Delta}(S), \boldsymbol{\nabla}(\bar{S})\} + \min\{\mathrm{vol}_F(S), \mathrm{vol}_F(\bar{S})\right) \\ &< \phi \cdot \min\{\mathrm{vol}_F(S), \mathrm{vol}_F(\bar{S})\}\end{aligned}$$

using $\boldsymbol{\Delta}(S) \leq \mathrm{vol}_F(S)$ and $\boldsymbol{\nabla}(\bar{S}) \leq \mathrm{vol}_F(\bar{S})$. If $\min\{\mathrm{vol}_F(S), \mathrm{vol}_F(\bar{S})\} \geq \delta/2 \cdot \mathrm{vol}_F(V)$, then we terminate the algorithm and return the balanced sparse cut. Otherwise, given the flow $\overrightarrow{\boldsymbol{f}}$, we extract the matching $\overrightarrow{M}$ from the representation $\{(\lambda_i, s_i, t_i)\}_{i \in [k]}$ of a $(\boldsymbol{w}_{\mathcal{H}}, h)$-short decomposition of $\overrightarrow{\boldsymbol{f}}$, where $h = O(n \log n \cdot (\kappa + 1/\psi)) = O(n \log n \cdot \max\{1/\phi + 1/\psi\})$ is the parameter from Lemma 7.4. That is, for each $i \in [k]$, we add an edge (s_i, t_i) to $\overrightarrow{M}$ with capacity λ_i, where we note that $\overrightarrow{M}$ is ψ-integral as the path decomposition is. Note that $\boldsymbol{d}_{\overrightarrow{M}}$ is equal to the source (for P) and sink (for Q) demand routed by $\overrightarrow{\boldsymbol{f}}$, and thus $\boldsymbol{d}_{\overrightarrow{M}}(V)$ is equal to $2|\overrightarrow{\boldsymbol{f}}|$. By Lemma 7.4, we have $\boldsymbol{\nabla}_{\boldsymbol{f}}(S) = 0$ and $\boldsymbol{\Delta}_{\boldsymbol{f}}(\bar{S}) = 0$, and using that $\boldsymbol{\Delta}_{\boldsymbol{f}}(S) = \boldsymbol{\nabla}_{\boldsymbol{f}}(\bar{S})$ we obtain

$$\boldsymbol{d}_{\overrightarrow{M}}(V) = 2|\overrightarrow{\boldsymbol{f}}| = \boldsymbol{d}'(V) - 2\boldsymbol{\Delta}_{\overrightarrow{\boldsymbol{f}}}(S) \geq \boldsymbol{d}'(V) - 2\min\{\mathrm{vol}_F(S), \mathrm{vol}_F(\bar{S})\} \geq \boldsymbol{d}'(V) - \delta \boldsymbol{d}(V).$$

This shows that $\overrightarrow{M}$ is a valid part of a $(P, Q, \boldsymbol{d}', \Delta, m')$-bidirectional-matching. To construct $\overleftarrow{M}$, we run the same algorithm in the reversed graph $\overleftarrow{G_A}$ with the same demand vector (and reverse the direction of the obtained flow and matching), noting that $\overleftarrow{G_A}$ is indeed the shortcut graph of $\overleftarrow{G}$. If neither of the executions found a balanced sparse cut, then we output $(\overrightarrow{M}, \overleftarrow{M})$; Otherwise, we output one of the sparse cuts. The algorithm runs in $O(n^2 \cdot (1/\phi + 1/\psi) \cdot \log^4 n)$ time by Lemma 7.4.

We now proceed to the second part of the lemma, i.e., any demand on $M \stackrel{\text{def}}{=} \overrightarrow{M} \cup \overleftarrow{M}$ can be routed with low congestion and short paths. Recall that our matchings $\overrightarrow{M}$ and $\overleftarrow{M}$ are constructed via Lemma 7.4, with the guarantee that the path decompositions of $\overrightarrow{\boldsymbol{f}}$ and $\overleftarrow{\boldsymbol{f}}$ are $(\boldsymbol{w}_{\mathcal{H}}, h)$-short. We prove the claim for $\overrightarrow{M}$ since $\overleftarrow{M}$ is analogous, and the overall statement follows by summing up the two flows. Consider a given ψ-integral flow $\boldsymbol{f}_{\overrightarrow{M}}$ on $\overrightarrow{M}$ of congestion κ. Let $(C_{\overrightarrow{\boldsymbol{f}}}, \{(\lambda_i, P_i)\}_{i \in [k]})$ be the underlying $(\boldsymbol{w}_{\mathcal{H}}, h)$-short path decomposition of $\overrightarrow{\boldsymbol{f}}$. We can write

$$\overrightarrow{\boldsymbol{f}} - C_{\overrightarrow{\boldsymbol{f}}} = \sum_{u,v} \sum_{(u,v)\text{-path } P_i \text{ with } \boldsymbol{w}_{\mathcal{H}}(P_i) \leq h} \lambda_i \boldsymbol{f}_{P_i},$$

where the λ_{P_i}'s for each (u,v) sum to the total capacity of the (u,v) edge in $\overrightarrow{M}$. We now construct a flow $\boldsymbol{f}'$ in G_A routing the same demand as $\boldsymbol{f}_{\overrightarrow{M}}$ by simply scaling up $\overrightarrow{\boldsymbol{f}}$ appropriately for each (u,v). In particular, we set

$$\boldsymbol{f}' \stackrel{\text{def}}{=} \sum_{u,v} \left(\frac{\boldsymbol{f}_{\overrightarrow{M}}(u,v)}{\boldsymbol{c}_{\overrightarrow{M}}(u,v)} \cdot \sum_{(u,v)\text{-path } P_i \text{ with } \boldsymbol{w}_{\mathcal{H}}(P_i) \le h} \lambda_i \boldsymbol{f}_{P_i} \right). \tag{6}$$

It is easy to see that $\boldsymbol{f}'$ routes the same demand as $\boldsymbol{f}_{\overrightarrow{M}}$. We also have $\boldsymbol{f}'(e) \le \overrightarrow{\boldsymbol{f}}(e) \cdot \max_{u,v} \frac{\boldsymbol{f}_{\overrightarrow{M}}(u,v)}{\boldsymbol{c}_{\overrightarrow{M}}(u,v)} \le \kappa \overrightarrow{\boldsymbol{f}}(e)$ for each e, meaning that $\boldsymbol{f}'$ is of congestion $O(\kappa \cdot \log n/\phi)$ in G_A. Finally, we have $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}') \le \|\boldsymbol{f}_{\overrightarrow{M}}\|_1 \cdot h$, since each P_i in (6) has weight $\boldsymbol{w}_{\mathcal{H}}(P_i) \le h$, and by definition those λ_i's sum up to $\boldsymbol{c}_{\overrightarrow{M}}(u,v)$. Combining the two $\boldsymbol{f}'$ for $\overrightarrow{M}$ and $\overleftarrow{M}$ as $\boldsymbol{f}_{G_A}$, this concludes the proof. □

7.3 Decomposing the Graph into Weak Expanders

The rest of the proof of our weak expander decomposition Lemma 5.2 is now mostly standard: We run the non-stop cut-matching game (Lemma 7.3) to attempt to certify the expansion of a large fraction of edges, using our matching player (Lemma 7.5) based on weighted push-relabel (Lemma 7.4). In each iteration, either our matching player finds a balanced cut—in which case we simply split the graph at the cut and recurse on both sides—or the cut is unbalanced in which case we have a matching that we feed to Lemma 7.3 and continue the algorithm. Thus, we either get a low recursion depth due to the balanced first case, or we successfully finish running the cut-matching game Lemma 7.3 and certify the expansion of a large fraction of the edges.

The crucial idea that makes our construction different (and simpler) from previous expander decomposition algorithms lies in how we leverage "weak" expander decomposition to handle the second case. To get a "strong" expander decomposition, one needs to ensure that each component is expanding *within* itself, meaning that the underlying witness cannot use any edges outside of the component. Previous work such as [SW19, SP25] thus needs to perform a *trimming* step that extracts from a "near" expander to an actual "strong" expander. In contrast, since weak expansion suffice for our purposes, we can allow the witnesses to be embedded into edges that we "cut" from the graph, potentially even outside of the components that they are in (as long as the *overall* congestion can be bounded). This nullifies the need of the trimming step, as we can simply remove the cut edges, and consequently greatly simplifies our algorithm.[23] We first claim that we can compute a large subset of expanding edges from an expanding vertex measure.

Claim 7.6. *Given an edge set F and a vertex measure $\boldsymbol{d}' \le \boldsymbol{d}$ with $\|\boldsymbol{d}'\|_1 \ge 0.8\|\boldsymbol{d}\|_1$ where $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{vol}_F$, there is an $O(|F|)$ time algorithm that computes an edge subset $F' \subseteq F$ such that $\mathbf{vol}_{F'} \le 2\boldsymbol{d}'$ and $\boldsymbol{c}(F') \ge 0.2 \cdot \boldsymbol{c}(F)$.*

Proof. Let $U \stackrel{\text{def}}{=} \{v \in V : \boldsymbol{d}'(v) \le \frac{1}{2}\boldsymbol{d}(v)\}$. We have

$$\boldsymbol{d}(U) \le 2\left(\boldsymbol{d}(U) - \boldsymbol{d}'(U)\right) \le 0.4 \cdot \boldsymbol{d}(V).$$

Thus, we simply let F' be the set of edges in F not incident to U which can be easily computed in linear time. We have $\boldsymbol{c}(F') \ge 0.2 \cdot \boldsymbol{c}(F)$ since $\boldsymbol{d}(V) = 2\boldsymbol{c}(F)$. Also, we have $\mathbf{vol}_{F'}|_{V\setminus U} \le \boldsymbol{d}|_{V\setminus U} \le 2\boldsymbol{d}'|_{V\setminus U}$ and that $\mathbf{vol}_{F'}|_U \le 2\boldsymbol{d}'|_U$ since each $u \in U$ has $\deg_{F'}(u) = 0$. □

[23] The state-of-the-art trimming algorithms for *directed* graphs [BPS20, SP25] are currently significantly more intricate than the undirected counterpart [SW19].

Splitting the Shortcut Graph. Our algorithm for Lemma 5.2 will involve "splitting" a shortcut graph G_A induced by $\mathcal{H}$ along a cut in G_A (in particular when we find a sparse cut in G_A when doing expander decomposition). Formally, we overload notation slightly and denote by $(G[S])_A$ the *induced* shortcut graph which is defined to be the shortcut graph of $G[S]$ induced by the hierarchy $\mathcal{H}[S]$. Note that while technically $(G[S])_A \cup (G[\bar{S}])_A$ is *not* a subgraph of G_A, it behaves essentially like one in the sense that there is a trivial one-way correspondence from edges in $(G[S])_A \cup (G[\bar{S}])_A$ to G_A. We thus have the following useful observation.

Observation 7.7. *Given a flow $\boldsymbol{f}$ in $(G[S])_A \cup (G[\bar{S}])_A$ routing a demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ supported on $V(G)$, there is an $O(m)$ time algorithm that computes a flow $\boldsymbol{f}_G$ routing $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in G_A.*

Finally, once the expander decomposition is computed, to support the second part of Lemma 5.2, we will prove that $E_r^{(r+1)}$ is not only expanding but also $(\boldsymbol{w}_{\mathcal{H}}, h)$-length expanding for some reasonably small h. This allows us to efficiently route the demand using the following lemma.

Lemma 7.8. *Let $F \subseteq E$ and $\mathcal{H}$ be a hierarchy of $G \setminus F$. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$ and $G_A \stackrel{\text{def}}{=} G \cup A$ be the shortcut graph. If $F' \subseteq F$ is $(\boldsymbol{w}_{\mathcal{H}}, h)$-length ϕ-expanding in G_A for some $h \in \mathbb{N}$, then there is a deterministic algorithm that, given any $1/z$-integral $(\mathbf{vol}_{F'} \cdot \kappa/z)$-respecting demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ for some $\kappa, z \in \mathbb{N}$, computes a $1/z$-flow $\boldsymbol{f}$ routing $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in G_A with $\boldsymbol{f}(e) \leq \frac{\lceil c_{7.8} \cdot \kappa \log n/\phi \cdot \boldsymbol{c}_{G_A}(e) \rceil}{z}$ in $O(m \log n + hn \log^3 n)$ time for some universal constant $c_{7.8} > 0$.*

Proof. Consider the given $1/z$-integral $(\mathbf{vol}_{F'} \cdot \kappa/z)$-respecting demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$. Note that $z \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ is an integral $(\mathbf{vol}_{F'} \cdot \kappa)$-respecting demand. By definition of F' being $(\boldsymbol{w}_{\mathcal{H}}, h)$-length ϕ-expanding in G_A, there exists a flow $\boldsymbol{f}^*$ routing $z \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in G_A with congestion κ/ϕ such that $\frac{\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f})}{|\boldsymbol{f}|} \leq h$. Running Theorem 2.2 on G_A with demand $z \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$, we can compute a ψ-integral flow $\boldsymbol{f}'$ routing $1/6$-fraction of the demand in G_A with congestion κ/ϕ in $O(m + hn \log^2 n)$ time.[24] Repeating this $O(\log n)$ times and summing all the flows, we get a ψ-integral flow $\boldsymbol{f}_\psi$ routing $z \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in G_A with congestion $O(\kappa \log n/\phi)$ in $O(m \log n + hn \log^3 n)$ time. We set $c_{7.8}$ to be the universal constant hidden in the above $O(\cdot)$ expression. Using Lemma 2.1, we can round $1/\psi \cdot \boldsymbol{f}_\psi$ to an integral flow $\boldsymbol{f}_z$ routing $z \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ such that $\boldsymbol{f}_z(e) \leq \lceil c_{7.8} \cdot \kappa \log n/\phi \cdot \boldsymbol{c}_{G_A}(e) \rceil$.[25] Setting $\boldsymbol{f} \stackrel{\text{def}}{=} \boldsymbol{f}_z/z$ completes the proof. □

We are now ready to prove Lemma 5.2, which we restate below.

Lemma 5.2 (Weak Expander Decomposition)**.** *There is a universal $\phi_{\text{rand}} = \Theta(1/\log^7 n)$ where $\phi_{\text{rand}} \in 1/\mathbb{N}$ such that there exists a randomized algorithm that, given the round-r level function $\mathsf{level}^{(r)}$ and the induced shortcut graph $G_{A^{(r)}}$ with capacity scale ψ of an n-vertex graph G, computes in $O(n^2 \log^9 n \cdot (\log^3 n + 1/\psi))$ time the* round-$(r+1)$ top-level edge set $E_{r+1}^{(r+1)}$ *such that*

1. $\boldsymbol{c}(E_{r+1}^{(r+1)}) \leq 0.9 \cdot \boldsymbol{c}(E_r^{(r)})$ *and*
2. $E_r^{(r+1)}$ *is $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained*[26] *$\Omega(1/\log^6 n)$-expanding in $G_{A^{(r)}}$, with high probability.*

[24] This bound was established in the proof of Lemma 4.1 and for brevity we omit repeating the calculation here.

[25] More specifically, $1/\psi \cdot \boldsymbol{f}_\psi$ routes $z/\psi \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$, and thus Lemma 2.1 gives us an $\boldsymbol{f}_z \leq \lceil \boldsymbol{f}_\psi \rceil$ routing $z \cdot (\boldsymbol{\Delta}, \boldsymbol{\nabla})$.

[26] Recall that this is defined to be $\mathcal{C}$-component-constrained for $\mathcal{C} \stackrel{\text{def}}{=} \text{SCC}(G \setminus E_{r+1}^{(r+1)})$, which is not necessarily the same as that of $G_{A^{(r)}} \setminus E_{r+1}^{(r+1)}$.

Furthermore, conditioned on Item 2 succeeding, given a $[G \setminus E_{r+1}^{(r+1)}]$*-component-constrained* $(\mathbf{vol}_{E_r^{(r+1)}} \cdot \kappa/z)$*-respecting* $1/z$*-integral demand* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ *for some* $\kappa, z \in \mathbb{N}$*, there is an* $O(n^2 \log^6 n \cdot (\log^3 n + 1/\psi))$ *time algorithm that returns a flow* $\boldsymbol{f}$ *routing* $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ *in* $G_{A^{(r)}}$ *such that* $\boldsymbol{f}(e) \leq \frac{\lceil \boldsymbol{c}(e) \cdot \kappa/\phi_{\text{rand}} \rceil}{z}$.

Proof. Let $\phi_{\text{exp}} \stackrel{\text{def}}{=} \Theta(1/\log^3 n)$ be sufficiently small. Let $F \stackrel{\text{def}}{=} E_r^{(r)}$ be the terminal edge set. We apply the cut-matching game of Lemma 7.3 on the graph $G_{A^{(r)}}$ and vertex measure $\boldsymbol{d} \stackrel{\text{def}}{=} \mathbf{vol}_F$, using the output of Lemma 7.5 with parameters ϕ_{exp} and δ_{KRV} as the oracle. This takes $O(\log^2 n) \cdot O(n^2 \log^4 n \cdot (\log^3 n + 1/\psi))$ time. There are the following two cases.

1. If in any of the $T_{\text{KRV}} = O(\log^2 n)$ iterations, Lemma 7.5 outputs a balanced sparse cut S (i.e., $\text{vol}_F(S) > \frac{\delta_{\text{KRV}}}{2} \cdot \text{vol}_F(V)$), then we terminate Lemma 7.3 and instead simply recurse on both sides of the cuts. In particular, let X be the edge set among $E_{G_{A^{(r)}}}(S, \bar{S})$ and $E_{G_{A^{(r)}}}(\bar{S}, S)$ that has a smaller total capacities. Since S is ϕ-sparse, we have $\boldsymbol{c}_{G_{A^{(r)}}}(X) < \phi_{\text{exp}} \cdot \text{vol}_F(S)$. We add $X \cap E(G)$ into the set $E_{r+1}^{(r+1)}$ and split G into two parts $(G[S])_{A^{(r)}}$ and $(G[\bar{S}])_{A^{(r)}}$, recursing on both of them (with terminal edges $F \cap G[S]$ and $F \cap G[\bar{S}]$).

2. On the other hand, if Lemma 7.5 outputs a valid ψ-integral bidirectional matching for all the T_{KRV} iterations, then by Lemma 7.3 we get a vertex measure $\tilde{\boldsymbol{d}} \leq \boldsymbol{d}$ with $\|\tilde{\boldsymbol{d}}\|_1 \geq \frac{7}{8}\|\boldsymbol{d}\|_1$ that is $O(\log^2 n)$-hop $\Omega(1/\log^2 n)$-expanding in the graph W defined in Lemma 7.3, with high probability. We terminate the recursion here and conclude that most of F is expanding in $G_{A^{(r)}}$ and obtain the appropriate routing data structure required. Indeed, by Claim 7.6, we can compute in $O(m)$ time an edge set $F' \subseteq F$ such that $\mathbf{vol}_{F'} \leq 2\tilde{\boldsymbol{d}}$ and $\boldsymbol{c}(F') \geq 0.2 \cdot \boldsymbol{c}(F)$, and then add the remaining edges $F \setminus F'$ to $E_{r+1}^{(r+1)}$. That $\mathbf{vol}_{F'} \leq 2\tilde{\boldsymbol{d}}$ means that F' is also $O(\log^2 n)$-hop $\Omega(1)$-expanding in W.

   Claim 7.9. *The edge set* F' *is* $(\boldsymbol{w}_{\mathcal{H}}, O(h \log^2 n))$*-length* $\Omega(\phi_{\text{exp}}/\log^3 n)$*-expanding in* G_A *for some* $h = O(n \log n \cdot (\log^3 n + 1/\psi))$.

   Proof. For any $\mathbf{vol}_{F'}$-respecting demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$, we construct a flow $\boldsymbol{f}$ routing in G_A with low congestion and average weight under $\boldsymbol{w}_{\mathcal{H}}$ as follows. Since F' is $O(\log^2 n)$-hop $\Omega(1)$-expanding in W, there exists a flow $\boldsymbol{f}_W$ routing $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in W with congestion $O(1)$ such that $\frac{\|\boldsymbol{f}_W\|_1}{|\boldsymbol{f}_W|} \leq O(\log^2 n)$. The flow $\boldsymbol{f}_W$ induces a demand on the matchings M_i's that form W: For each M_i, by Lemma 7.5, the vertex demand induced by $\boldsymbol{f}_W|_{M_i}$ can be routed in G_A by a flow $\boldsymbol{f}_i$ with congestion $O(\log n/\phi_{\text{exp}})$ such that $\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}_i) \leq \|\boldsymbol{f}_W|_{M_i}\|_1 \cdot h$ for $h = O(n \log n \cdot (1/\phi_{\text{exp}} + 1/\psi)) = O(n \log n \cdot (\log^3 n + 1/\psi))$. Setting $\boldsymbol{f}$ to be the sum of the $\boldsymbol{f}_i$'s, we have that $\boldsymbol{f}$ is of congestion $O(\log^3 n/\phi_{\text{exp}})$ in G_A and

$$\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}) = \sum_i \boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}_i) \leq h \cdot \sum_i \|\boldsymbol{f}_W|_{M_i}\|_1 \leq O(h \cdot \log^2 n) \cdot |\boldsymbol{f}_W|.$$

   Using the fact that $\boldsymbol{f}$ routes the same demand as $\boldsymbol{f}_W$ does, this completes the proof. □

Overall, algorithm recursively decomposes the graph $G_{A^{(r)}}$ until no balanced sparse cut is found by Lemma 7.5. The recursion depth is bounded by $O(\log^3 n)$ as the volume of the graph decreases by a factor of $1 - \Omega(1/\log^2 n)$ in each recursion. Since the graphs on each level are edge-disjoint, the total capacities of edges added to $E_{r+1}^{(r+1)}$ in Case 1 is bounded by $O(\log^3 n) \cdot \phi_{\text{exp}} \cdot \boldsymbol{c}(E_r^{(r)}) \leq 0.1 \cdot \boldsymbol{c}(E_r^{(r)})$

for $\phi_{\text{exp}} = O(1/\log^3 n)$ sufficiently small. For edges added in Case 2 (i.e., for each leaf nodes in the recursion tree), since the leaves are edge-disjoint and for each of them the capacities of edges we added to $E_{r+1}^{(r+1)}$ is at most $0.8 \cdot \boldsymbol{c}(F)$ for the respective set F of the leaf node, in total we added at most $0.8 \cdot \boldsymbol{c}(E_r^{(r)})$ units of capacity to $E_{r+1}^{(r+1)}$ in Case 2. Therefore, we can bound the final total capacities of $E_{r+1}^{(r+1)}$ by $0.9 \cdot \boldsymbol{c}(E_r^{(r)})$, as required. The running time for constructing $E_{r+1}^{(r+1)}$ is $O(n^2 \log^9 n \cdot (\log^3 n + 1/\psi))$. This is because at each level we spend $O(n^2 \log^6 n \cdot (\log^3 n + 1/\psi))$ time certifying the expansion of the current edge set $E_r^{(r)}$ and constructing the witness (at the leaf nodes), and there are $O(\log^3 n)$ levels.

To show that $E_r^{(r+1)}$, which by definition is just $E_r^{(r)} \setminus E_{r+1}^{(r+1)}$, is $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained $\Omega(1/\log^6 n)$-expanding, we first note that the components of $G \setminus E_{r+1}^{(r+1)}$ is a refinement of the components corresponding to the leaf nodes in the recursion. By construction, the set $E_r^{(r+1)}$ is the disjoint union of the F'_C's, the edge set F' for each leaf node C that we have proven expansion in Claim 7.9 in the corresponding graph $(G[C])_A$. Combined with Observation 7.7, this implies $E_r^{(r+1)}$ is $\Omega(1/\log^6 n)$-expanding because any flow routing $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained $\mathbf{vol}_{E_r^{(r+1)}}$-respecting demand in $\bigcup_G G[C]_{A^{(r)}}$ (which exists by the expansion guarantee of each F'_C) can be converted into an equivalent one in $G_{A^{(r)}}$. Using the same argument, for any given $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained $(\mathbf{vol}_{E_r^{(r+1)}} \cdot \kappa/z)$-respecting $1/z$-integral demand, we can route it in $G_{A^{(r)}}$ by first splitting split the demand to each component. For each component, we apply Lemma 7.8 to route it within the component and then combine the flows using Observation 7.7, leveraging Claim 7.9 that F' is $(\boldsymbol{w}_{\mathcal{H}}, O(n \log^3 n \cdot (\log^3 n + 1/\psi)))$-length $\Omega(1/\log^6 n)$-expanding. The resulting flow $\boldsymbol{f}$ satisfies $\boldsymbol{f}(e) \le \frac{\lceil \boldsymbol{c}(e) \cdot \kappa/\phi_{\text{rand}} \rceil}{z}$ if we properly define $\phi_{\text{rand}} \stackrel{\text{def}}{=} \Theta(1/\log^7 n)$ (note that the ϕ value in Lemma 7.8 is $\Theta(1/\log^6 n)$ per Claim 7.9). The time to compute such a flow is $O(n^2 \log^6 n \cdot (\log^3 n + 1/\psi))$ by Lemma 7.8 with $h = O(n \log^3 n \cdot (\log^3 n + 1/\psi))$, noting that the leaf components of the recursion tree are disjoint. This completes the proof. □

Acknowledgements

We thank Aaron Sidford for helpful discussions on deterministic flow algorithms.

References

[AKR07] Baruch Awerbuch, Rohit Khandekar, and Satish Rao. Distributed algorithms for multicommodity flow problems via approximate steepest descent framework. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 949–957. SIAM, 2007. 1

[ALPS23] Amir Abboud, Jason Li, Debmalya Panigrahi, and Thatchaphol Saranurak. All-pairs max-flow is no harder than single-pair max-flow: Gomory-hu trees in almost-linear time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 2204–2212. IEEE, 2023. 1

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996. 4

[BBST24] Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, and Ta-Wei Tu. Maximum flow by augmenting paths in $n^{2+o(1)}$ time. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 2056–2077. IEEE, 2024. , 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 40, 43

[BPS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, SCC, and shortest paths via directed expanders and congestion balancing. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1123–1134. IEEE, 2020. 3, 27, 39, 40, 43

[BT25] Joakim Blikstad and Ta-Wei Tu. Implementation of weighted push-relabel maximum flow algorithm on shortcut. https://github.com/TaWeiTu/combinatorial-max-flow, 2025. 3

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1158–1167. IEEE, 2020. 3, 36, 41, 42, 43, 44

[CHLP23] Ruoxu Cen, William He, Jason Li, and Debmalya Panigrahi. Steiner connectivity augmentation and splitting-off in poly-logarithmic maximum flows. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 2449–2488. SIAM, 2023. 1

[CK24a] Julia Chuzhoy and Sanjeev Khanna. A faster combinatorial algorithm for maximum bipartite matching. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 2185–2235. SIAM, 2024. 2

[CK24b] 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 SIGACT Symposium on Theory of Computing, STOC 2024*. ACM, 2024. 2

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 612–623. IEEE, 2022. 1, 2, 3

[CKL+24] Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, s-t shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 1165–1173. ACM, 2024. 1, 2

[CLN+21] Ruoxu Cen, Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Kent Quanrud. Minimum cuts in directed graphs via partial sparsification. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1147–1158. IEEE, 2021. 1

[CMGS25] Daoyuan Chen, Simon Meierhans, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Parallel and distributed expander decomposition: Simple, fast, and near-optimal. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 1705–1719. SIAM, 2025. 1

[Coh95] Edith Cohen. Approximate max-flow on small depth networks. *SIAM J. Comput.*, 24(3):579–597, 1995. 1

[CRRT09] Kamalika Chaudhuri, Satish Rao, Samantha J. Riesenfeld, and Kunal Talwar. A push-relabel approximation algorithm for approximating the minimum-degree MST problem and its generalization to matroids. *Theor. Comput. Sci.*, 410(44):4489–4503, 2009. 1

[CS20] Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 377–388. IEEE, 2020. 1

[Dan51] George B Dantzig. Application of the simplex method to a transportation problem. *Activity analysis and production and allocation*, 1951. 1

[Din70] Efim A Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970. 1

[DS08] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 451–460. ACM, 2008. 1

[ET75] Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. 1

[FF56] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. 1

[FI03] Lisa Fleischer and Satoru Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discret. Appl. Math.*, 131(2):311–322, 2003. 1

[FLL25] Henry Fleischmann, George Z. Li, and Jason Li. Improved directed expander decompositions. *CoRR*, abs/2507.09729, 2025. 3, 24, 45, 46, 47

[FM12] András Frank and Zoltán Miklós. Simple push-relabel algorithms for matroids and submodular flows. *Japan journal of industrial and applied mathematics*, 29:419–439, 2012. 1

[GGT89] Giorgio Gallo, Michael D. Grigoriadis, and Robert Endre Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989. 1

[GH61] 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. 1

[GLP21] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than Goldberg-Rao. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 516–527. IEEE, 2021. 1

[GMQT12] Frieda Granot, S. Thomas McCormick, Maurice Queyranne, and Fabio Tardella. Structural and algorithmic properties for parametric minimum cuts. *Math. Program.*, 135(1-2):337–367, 2012. 1

[GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998. 1

[GRST21] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2212–2228. SIAM, 2021. 2

[GT88] Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988. 1

[HHL+24] Bernhard Haeupler, D. Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. Low-step multi-commodity flow emulators. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 71–82. ACM, 2024. 4, 6

[HHS23] Bernhard Haeupler, D. Ellis Hershkowitz, and Thatchaphol Saranurak. Maximum length-constrained flows and disjoint paths: Distributed, deterministic, and fast. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 1371–1383. ACM, 2023. 1, 6

[HO94] Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994. 1

[HRG00] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000. 1

[HRG22] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1325–1338. ACM, 2022. 6

[HRW17] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1919–1938. SIAM, 2017. 1

[Iwa03] Satoru Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM J. Comput.*, 32(4):833–840, 2003. 1

[Kar73] Alexander V Karzanov. On finding maximum flows in networks with special structure and some applications. *Matematicheskie Voprosy Upravleniya Proizvodstvom*, 5:81–94, 1973. 1

[Kat20] Tarun Kathuria. A potential reduction inspired algorithm for exact max flow in almost $\widetilde{O}(m^{4/3})$ time. *CoRR*, abs/2009.03260, 2020. 2

[Kav24] Nithin Kavi. Partial implementation of max flow and min cost flow in almost-linear time. *CoRR*, abs/2407.10034, 2024. 3

[KKOV07] Rohit Khandekar, Subhash A. Khot, Lorenzo Orecchia, and Nisheeth K. Vishnoi. On a cut-matching game for the sparsest cut problem. Technical Report UCB/EECS-2007-177, EECS Department, University of California, Berkeley, Dec 2007. 40

[KL15] David R. Karger and Matthew S. Levine. Fast augmenting paths by random sampling from residual graphs. *SIAM J. Comput.*, 44(2):320–339, 2015. 2

[KLS20] Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $m^{4/3}$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 119–130. IEEE, 2020. 1

[KP15] Donggu Kang and James Payor. Flow rounding. *CoRR*, abs/1507.08139, 2015. 5, 22

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 317–329. ACM, 2021. 1

[Lou10] Anand Louis. Cut-matching games on directed graphs. *CoRR*, abs/1010.1047, 2010. 23

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 85–92. IEEE, 2020. 1

[LRW25] Jason Li, Satish Rao, and Di Wang. Congestion-approximators from the bottom up. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 2111–2131. SIAM, 2025. 3, 4, 24

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\widetilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 424–433. IEEE Computer Society, 2014. 1, 2

[LS20a] Yang P. Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow. *CoRR*, abs/2003.08929, 2020. 2

[LS20b] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 803–814. ACM, 2020. 1, 2

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 253–262. IEEE Computer Society, 2013. 1

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pages 593–602. IEEE Computer Society, 2016. 1, 2

[OZ14] Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1267–1286. SIAM, 2014. 1

[Qua24] Kent Quanrud. Faster exact and approximation algorithms for packing and covering matroids via push-relabel. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 2305–2336. SIAM, 2024. 1

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 227–238. SIAM, 2014. 3

[SP25] Aurelio L. Sulser and Maximilian Probst Gutenberg. Near-optimal algorithm for directed expander decompositions. In *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025*, volume 334 of *LIPIcs*, pages 132:1–132:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. 23, 27, 44

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. 47

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90. ACM, 2004. 1

[SW19] 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, SODA 2019*, pages 2616–2635. SIAM, 2019. 1, 3, 24, 27

[vdBCK+24] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-linear time algorithms for decremental graphs: Min-cost flow and more via duality. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 2010–2032. IEEE, 2024. 1, 2, 41

[vdBCP+23] Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 503–514. IEEE, 2023. 1, 2

[vdBGJ+22] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 543–556. ACM, 2022. 1

[vdBLL+21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 859–869. ACM, 2021. 1, 2, 3

[vdBLN+20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 919–930. IEEE, 2020. 1, 3

[Vis92] Uzi Vishkin. A parallel blocking flow algorithm for acyclic networks. *J. Algorithms*, 13(3):489–501, 1992. 1

A Deterministic Vertex-Capacitated Flow

In this section we show that how our algorithm can be derandomized in the special case of vertex-capacitated flows, proving Theorem 1.2. Note that the only randomized component of our algorithm is the non-stop cut-matching game of Lemma 7.3 that runs in near-linear time with respect to the number of edges returned by the matching player which can be as large as $\Omega(n^2)$ in the edge-capacitated maximum flow instances. However, when the graph is vertex-capacitated, we can substitute the randomized cut-matching game with a deterministic one (see, e.g., [CGL+20]) that runs in near-quadratic time instead of near-linear time, leveraging the fact that the matching player always outputs a matching of size $\tilde{O}(n)$.[27]

Consider a graph $G = (V, E)$ with vertex capacities $\boldsymbol{c}_G \in \mathbb{N}^V$. We do the standard transformation of splitting each vertex v into an in-vertex v_{in} and an out-vertex v_{out} with a directed edge $(v_{\text{in}}, v_{\text{out}})$ of capacity $\boldsymbol{c}_G(v)$ between them. Edges in G are directed from the out-vertex of its tail to the in-vertex of its head, with infinite edge capacity. Let $\tilde{G}$ be this edge-capacitated graph. It is easy to see that there is a bijection between $(s_{\text{out}}, t_{\text{in}})$-flows in $\tilde{G}$ respecting edge capacities and (s, t)-flow in G respecting vertex capacities of the same value. Thus we will run our maximum flow algorithm on $\tilde{G}$. Note that $\tilde{G}$ contains n finite-capacity edges which we exploit to argue that the matching player always outputs a sparse matching. Recall that our algorithm works by iteratively finding $O(1)$-approximate flows in residual graphs, and while this property holds for the initial input graph, it may not hold for an arbitrary residual graph. Therefore, we first show that we can "round" any flow in $\tilde{G}$ to only be supported on few, $O(n)$ edges. This ensures that the residual graph with respect to this rounded flow will only contain $O(n)$ finite-capacity edges as well.

Lemma A.1. *Given an integral (s, t)-flow $\boldsymbol{f}$ in $\tilde{G}$, there is a **deterministic** $O(m \log n)$ time algorithm that computes another integral (s, t)-flow $\boldsymbol{f}'$, of value $|\boldsymbol{f}'| = |\boldsymbol{f}|$, that contains $3n$ non-zero entries.*

Proof. We consider a correspondence between vertex-capacitated flows and bipartite $\boldsymbol{b}$-matchings. Consider a bipartite graph where the left bipartition consists of all out-vertices v_{in} and the right bipartition all in-vertices v_{in}. Each edge $(u_{\text{out}}, v_{\text{in}})$ is included to the bipartite graph as an undirected edge. We can interpret the flow $\boldsymbol{f}$ naturally as a $\boldsymbol{b}$-matching in the bipartite graph where $\boldsymbol{b}(v_{\text{in}}) =$

[27] Recall that the size of the matching is the number of edges it is supported by, which in Section 7 was bounded only by $\tilde{O}(m)$.

$\boldsymbol{b}(v_{\text{out}}) = \boldsymbol{f}^{\text{out}}(v)$ for $v \notin \{s,t\}$, $\boldsymbol{b}(s_{\text{out}}) = \boldsymbol{f}^{\text{out}}(s)$, and $\boldsymbol{b}(t_{\text{in}}) = \boldsymbol{f}^{\text{in}}(t)$. Conversely, any $\boldsymbol{b}$-matching $\boldsymbol{x}$ in the bipartite graph corresponds to an (s,t)-flow in G with the same value as $\boldsymbol{f}$ and the same support (modulo the edges between in- and out-vertices).

Suppose there is a cycle $C = (e_1, \ldots, e_2, \ldots, e_t)$ (as a sequence of edges) in the support of $\boldsymbol{x}$. We can observe that if we add δ to $\boldsymbol{x}(e_i)$ for odd i and $-\delta$ to $\boldsymbol{x}(e_i)$ for even i, where $\delta \stackrel{\text{def}}{=} \min_j \boldsymbol{x}(e_{2j})$, then this is still a $\boldsymbol{b}$-matching and with one less non-zero edges. Repeating this process until there is no cycles in the support of $\boldsymbol{x}$, we see that now $\boldsymbol{x}$ contains at most $2n$ non-zero entries (and thus the corresponding flow contains $3n$ non-zero entries). To implement the algorithm, we can use a standard link-cut tree data structure to efficiently discover cycles and alter the values of $\boldsymbol{x}$ along the cycle. This will take $O(m \log n)$ time. □

In Section A.1 we prove the following lemma that computes an $O(1)$-approximate maximum flow *deterministically* when there are few finite-capacity edges.

Lemma A.2. *There is a* ***deterministic*** *algorithm that, given a graph $G = (V, E)$ with edge capacities $\boldsymbol{c} \in (\mathbb{N} \cup \{\infty\})^E$ such that $E_\infty \stackrel{\text{def}}{=} \{e : \boldsymbol{c}(e) = \infty\}$ forms a DAG and $|E \setminus E_\infty| = O(n)$, computes an $O(1)$-approximate integral (s,t)-flow in $O(n^2 \log^{45} n)$ time.*

Now Theorem 1.2 follows from Lemma A.2.

Theorem 1.2. *There is a deterministic algorithm that, given an n-vertex directed graph G with vertex capacities from $\{1, \ldots, U\}$ and $s, t \in V(G)$, finds a maximum (s,t)-flow in G in $O(n^2 \log^{46} n \log U)$ time.*

Proof. We start with the empty flow $\boldsymbol{f}$ and iteratively apply Lemma A.2. We maintain the invariant that $\boldsymbol{f}$ always contains at most $3n$ non-zero entries. In each iteration, in $O(n^2 \log^{45} n)$ time we can find an $O(1)$-approximate integral flow $\boldsymbol{f}'$ in $G_{\boldsymbol{f}}$, noting that (1) $G_{\boldsymbol{f}}$ contains $4n$ edges with finite capacities (including the initial n edges between in- and out-vertices and the additional $3n$ ones from the residual edges) and (2) the infinite-capacity edges form a DAG (since they all go from out-vertices to in-vertices). We update our $\boldsymbol{f}$ by combining it with $\boldsymbol{f}'$, and then apply Lemma A.1 to reduce the number of non-zero edges back to $3n$. This takes $O(\log n)$ iterations until we find a maximum flow, proving the theorem. □

A.1 Deterministic Approximate Flow Algorithm

Throughout this section we consider the input graph G to Lemma A.2. In particular we let E_∞ denote the infinite-capacity edges and assume that they form a DAG and contain the majority of edges.

Assumption A.3. The set of infinite-capacity edges $E_\infty \stackrel{\text{def}}{=} \{e : \boldsymbol{c}(e) = \infty\}$ forms a DAG and the number of finite-capacity edges is $|E \setminus E_\infty| = O(n)$.

Recall that the randomized analogue of Lemma A.2, i.e., Lemma 6.1, is proven via Algorithm 3. The first step of Algorithm 3 is to construct an expander hierarchy $\mathcal{H}$, its shortcut, and an unfolding data structure. We provide a determinstic algorithm analogous to Lemma 5.1, where $\phi_{\text{det}} = O(1/\log^{24} n)$ is universal and defined in Lemma A.5—the deterministic analogue of Lemma 5.2.

Lemma A.4. *There is a* ***deterministic*** *algorithm that, given an n-vertex graph G satisfying Assumption A.3, in $O(n^2 \log^{36} n)$ time computes*

- *a hierarchy $\mathcal{H}$ of G with $L = O(\log n)$ levels,*
- *a shortcut A induced by $\mathcal{H}$ with capacity scale $\psi_{\mathrm{det}} = \Theta(\phi_{\mathrm{det}}/\log n)$, and*
- *a "flow-unfolding" data structure $\mathcal{D}_{\mathrm{unfold}}$ such that, given any ψ-integral feasible flow $\boldsymbol{f}_A$ in $G_A \stackrel{\mathrm{def}}{=} G \cup A$, $\mathcal{D}_{\mathrm{unfold}}$ returns a flow $\boldsymbol{f} \stackrel{\mathrm{def}}{=} \mathcal{D}_{\mathrm{unfold}}(\boldsymbol{f}_A)$ in G routing the same vertex-demand with congestion 3 in $O(n^2 \log^{45} n)$ time.*

Using Lemma A.4, Lemma A.2 follows in exactly the same way as Lemma 6.1 in Section 6. The running time is $O(n^2 \log^{45} n)$. To prove Lemma A.4, we use a deterministic analog of Lemma 5.2.

Lemma A.5. *There is a universal $\phi_{\mathrm{det}} = \Theta(1/\log^{24} n)$ where $\phi_{\mathrm{det}} \in 1/\mathbb{N}$ such that there exists a **deterministic** algorithm that, given the round-r level function $\mathsf{level}^{(r)}$ such that $E_r^{(r)} \cap E_\infty = \emptyset$ and the induced shortcut graph $G_{A^{(r)}}$ with capacity scale ψ of an n-vertex graph G, computes in $O(n^2 \log^{10} n \cdot (\log^7 n + 1/\psi))$ time the* round-$(r+1)$ top-level edge set $E_{r+1}^{(r+1)}$ *such that*

- $\boldsymbol{c}(E_{r+1}^{(r+1)}) \le 0.9 \cdot \boldsymbol{c}(E_r^{(r)})$, *and*
- $E_r^{(r+1)}$ *is $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained $\Omega(1/\log^{23} n)$-expanding in $G_{A^{(r)}}$.*

Furthermore, given a $[G \setminus E_{r+1}^{(r+1)}]$-component-constrained $(\mathbf{vol}_{E_r^{(r+1)}} \cdot \kappa/z)$-respecting $1/z$-integral demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ for some $\kappa, z \in \mathbb{N}$, there is an $O(n^2 \log^{19} n \cdot (\log^5 n + 1/\psi))$ time algorithm that returns a flow $\boldsymbol{f}$ routing $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ in $G_{A^{(r)}}$ such that $\boldsymbol{f}(e) \le \frac{\lceil \boldsymbol{c}(e) \cdot \kappa/\phi_{\mathrm{det}} \rceil}{z}$.

Proof of Lemma A.4. We first put E_∞ into the first level of the hierarchy and then build a hierarchy on the rest of the edges, i.e., we set $\mathsf{level}^{(1)}(e) \stackrel{\mathrm{def}}{=} 1$ for all $e \in E$, $\mathsf{level}^{(2)}(e) \stackrel{\mathrm{def}}{=} 1$ for $e \in E_\infty$ and $\mathsf{level}^{(2)}(e) \stackrel{\mathrm{def}}{=} 2$ otherwise. Note that this is vacuously valid because E_∞ forms a DAG by Assumption A.3, so everything is $(G \setminus E_2^{(2)})$-component-constrained expanding (the shortcut $A^{(1)}$ is trivial since we only consider intra-component edges in the construction of stars; see Definition 2.7). We now build the rest of the hierarchy via Lemma A.5. Note that infinite-capacity edges will never appear in upper levels since $\boldsymbol{c}(E_{r+1}^{(r+1)}) \le \boldsymbol{c}(E_r^{(r)})$ where the right-hand side is finite for $r \ge 2$. This and the flow unfolding data structure are the same as Lemma 5.1. □

Now, we prove Lemma A.5. Recall that Lemma 5.2 is obtained by running the randomized non-stop cut-matching game where the matching player (Lemma 7.5) runs the weighted push-relabel algorithm on the shortcut graphs. Note that the matching player obtains the matching by decomposing the flow into paths, and standard path-decomposition algorithm (e.g., Lemma C.1) guarantees that each path it found either saturates a source/sink vertex or saturates an edge. Since we put all infinite-capacity edges in the bottom level of the hierarchy, our algorithms are dealing with finite demand, and therefore we only saturate finite-capacity edges. This implies that the size of the matching is in fact bounded by $n + |E(G_A) \setminus E_\infty|$ (which in our case is $O(n \log n)$; $O(n)$ from the original graph, and $O(n \log n)$ finite-capacity star edges) instead of the cruder bound of $m' = |E(G_A)|$ in Lemma 7.5.

Lemma A.6 (Matching Player for Vertex-Capacitated Graphs)**.** *Let $F \subseteq E$ and $\mathcal{H}$ be a hierarchy of $G \setminus F$. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$ and $G_A \stackrel{\mathrm{def}}{=} G \cup A$ be the shortcut graph that contains m' edges. Then, for any $\phi < 1$ and $\delta < 1$, given a partition (P, Q) of V and ψ-integral $\boldsymbol{d}' \le \boldsymbol{d}$ for $\boldsymbol{d} \stackrel{\mathrm{def}}{=} \mathbf{vol}_F$, there is a deterministic algorithm that runs in $O(n^2 \log^4 n \cdot (1/\phi + 1/\psi))$ time and computes either*

- *a balanced ϕ-sparse cut S such that* $\mathrm{vol}_F(S) \geq \delta/2 \cdot \mathrm{vol}_F(V)$*, or*
- *a ψ-integral $(P, Q, \boldsymbol{d}', \Delta, n + |E(G_A) \setminus E_\infty|)$-bidirectional-matching $(\overrightarrow{M}, \overleftarrow{M})$ where $\Delta \stackrel{\text{def}}{=} \delta \cdot \mathrm{vol}_F(V)$.*

In the second case, the matchings are such that, given any ψ-integral flow $\boldsymbol{f}_M$ on $M \stackrel{\text{def}}{=} \overrightarrow{M} \cup \overleftarrow{M}$ of congestion κ, there exists a ψ-integral flow $\boldsymbol{f}_{G_A}$ routing the same demand as $\boldsymbol{f}$ does in G_A with congestion $O(\kappa \cdot \log n/\phi)$ such that $\frac{\boldsymbol{w}_{\mathcal{H}}(\boldsymbol{f}_{G_A})}{|\boldsymbol{f}_{G_A}|} \leq h$ for some $h = O(n \log n \cdot (1/\phi + 1/\psi))$.

Our derandomization comes from substituting the randomized non-stop KRV-style cut-matching game with a deterministic KKOV-style variant. Note that we do not prove a non-stop version as in Section 7; Instead, we require the matching player to have $\delta = 0$, or in other words that it always returns a perfect matching.

Lemma A.7 (Deterministic Cut-Matching Game [BPS20])**.** *There exists a universal constant $c_{A.7}$ such that the following holds.*

Consider an integral vertex measure $\boldsymbol{d}$ on n vertices V where $\|\boldsymbol{d}\|_\infty \leq \mathrm{poly}(n)$. Let $z \in \mathbb{N}$. Suppose there is an oracle that on input partition (P, Q) and $1/z$-integral $\boldsymbol{d}' \leq \boldsymbol{d}$ outputs a $1/z$-integral $(P, Q, \boldsymbol{d}', \Delta, m)$-bidirectional-matching for $\Delta \stackrel{\text{def}}{=} c_{A.7}/\log^4 n \cdot \boldsymbol{d}(V)$.

*Then, there is a **deterministic** algorithm that invokes the oracle $T_{\mathrm{KKOV}} = O(\log n)$ times and takes $O(m^2 \log^9 n)$ additional running time to compute a submeasure $\tilde{\boldsymbol{d}} \leq \boldsymbol{d}$ with $\|\tilde{\boldsymbol{d}}\|_1 \geq \frac{7}{8}\|\boldsymbol{d}\|_1$ such that $\tilde{\boldsymbol{d}}_1$ is $O(\log^{15} n)$-hop $\Omega(1/\log^{16} n)$-expanding in $W \stackrel{\text{def}}{=} \bigcup_{i \in [k]} \overrightarrow{M}_i \cup \overleftarrow{M}_i$, where $\overrightarrow{M}_i \cup \overleftarrow{M}_i$ is the output of the oracle in the i-th invocation.*

Proof of Lemma A.5. The proof remains largely the same as that of Lemma 5.2 and thus we only describe the difference. Recall that the strategy is to use cut-matching games to either certify that a large fraction of the terminal edges are expanding (which we add to $E_r^{(r+1)}$), or find a balanced sparse cut for us to recurse on.

We replace the randomized non-stop cut-matching game Lemma 7.3 with Lemma A.7, using Lemma A.6 with $\phi_{\mathrm{exp}} = O(1/\log^5 n)$ and $\delta = O(1/\log^4 n)$, both sufficiently small, as the oracle. If Lemma A.6 ever returns a balanced sparse cut, then we recurse on both sides of the cut. Otherwise, we use the output of Lemma A.6 as the matching that is fed to Lemma A.7. In $T_{\mathrm{KKOV}} = O(\log n)$ rounds, the cut-matching game Lemma A.7 terminates with a $\tilde{\boldsymbol{d}}$ that is $O(\log^{15} n)$-hop $\Omega(1/\log^{16} n)$-expanding in W. Using the same reasoning as in the proof of Lemma 5.2, we can extract a large edge set that is $(\boldsymbol{w}_{\mathcal{H}}, O(h \log^{15} n))$-length $\Omega(\phi_{\mathrm{exp}}/\log^{18} n)$-expanding in G_A for $h = O(n \log n \cdot (\log^5 n + 1/\psi))$.

The correctness of the algorithm is the same as in Lemma 5.2. Since the size of the subgraph decreases by a $(1 - \Omega(1/\log^4 n))$ fraction in each recursion, the recursion depth is bounded by $O(\log^5 n)$ and for ϕ_{exp} sufficiently small the capacity of $E_{r+1}^{(r+1)}$ is bounded by $0.9 \cdot \boldsymbol{c}(E_r^{(r)})$ as desired. The construction takes $O(\log^5 n) \cdot O(\log n) \cdot (O(n^2 \log^{11} n) + O(n^2 \log^4 n \cdot (\log^5 n + 1/\psi))) = O(n \log^{10} n \cdot (\log^7 n + 1/\psi))$ time. Given any $(\mathbf{vol}_{E_r^{(r+1)}} \cdot \kappa/z)$-respecting $1/z$-integral demand, we can route it using Lemma 7.8 with a flow $\boldsymbol{f}$ with $\boldsymbol{f}(e) \leq \frac{\lceil \boldsymbol{c}_{G_A}(e) \cdot \kappa/\phi_{\mathrm{det}} \rceil}{z}$, for some $\phi_{\mathrm{det}} \stackrel{\text{def}}{=} \Theta(1/\log^{24} n)$, in $O(n^2 \log^{19} n \cdot (\log^5 n + 1/\psi))$ time. This completes the proof. □

This finishes the description of our deterministic vertex-capacitated max-flow algorithm, modulo the proof of Lemma A.7 which we do in the next section.

A.2 Deterministic Cut-Matching Game

In this section we show how the directed version of the deterministic cut-matching game (i.e., Lemma A.7) can be proved. The cut-matching game works as follows: Let W_{all} initially contain only self loops on the vertices so that $\deg_{W_{\text{all}}}(v) = 2\boldsymbol{d}(v)$ for all v. We maintain the invariant that $\deg_{W_{\text{all}}}(v) = 2k \cdot \boldsymbol{d}(v)$ for all v at the start of the k-th iteration, and that $W_{\text{all}} = W \cup F$ for some *real* edges W and *fake* edges F, with $\boldsymbol{c}(F) \leq c \cdot \boldsymbol{c}(W_{\text{all}})/\log^4 n$ for some sufficiently small constant c (depending on the value of $c_{A.9}$ below in Lemma A.9). In each iteration, we run the following Lemma A.9 on W_{all} (with F being the set of fake edges). If for some iteration, Lemma A.9 concludes that for some $X \subseteq V$ with $\text{vol}_{W_{\text{all}}}(X) \geq 0.88 \cdot \text{vol}_{W_{\text{all}}}(V)$ we have that $W_{\text{all}}[X] \setminus F = W[X]$ is $\Omega(1/\log^{14} n)$-expander, then we show that $\tilde{\boldsymbol{d}} \stackrel{\text{def}}{=} \boldsymbol{d}|_X$ satisfies the output requirement of Lemma A.7. Indeed, we have that $\|\tilde{\boldsymbol{d}}\|_1 = \text{vol}_{W_{\text{all}}}(X)/2k \geq 0.88 \cdot \text{vol}_{W_{\text{all}}}(V)/2k \geq \frac{7}{8}\|\boldsymbol{d}\|_1$. To see the expansion guarantee of $\tilde{\boldsymbol{d}}$, note that $\deg_{W[X]}(v) \geq \tilde{\boldsymbol{d}}(v)$ for all v. Combining the following claim and that $W[X]$ is an $\Omega(1/\log^{14} n)$-expander, we get that $\tilde{\boldsymbol{d}}$ is $O(\log^{15} n)$-hop $\Omega(1/\log^{16} n)$-expanding in W.

Fact A.8 (see, e.g., [BBST24, Lemma 5.10])**.** *Let W be an n-vertex ϕ-expander. Then, $\mathbf{vol}_W$ is $O(\log n/\phi)$-hop $\Omega(\phi/\log^2 n)$-expanding in W.*

If, on the other hand, Lemma A.9 finds a cut $(S, \overline{S})$, we use the matching player to compute a matching between S and $\overline{S}$ which is then added to W_{all}.[28] Note crucially that while the matchings the oracle outputs are not necessarily perfect, we add a (fairly arbitrary) set of fake edges with capacity bounded by $2\Delta = 2c_{A.7}/\log^4 n \cdot \boldsymbol{d}(V)$ to make them perfect before including them in W_{all}. With $c_{A.7}$ sufficiently small, the edge set F always has capacity bounded by $c_{A.9} \cdot U/\log^4 n$, where $U \stackrel{\text{def}}{=} \sum_{e \in W_{\text{all}}} \boldsymbol{c}(e)$, which ensures that Lemma A.9 is always applicable in the next iteration.

It then remains to bound the number of iterations this process takes. Note that in W_{all}, we always add a perfect matching between a balanced sparse cut $(S, \overline{S})$. This was first analyzed by [KKOV07] for unweighted undirected graphs, and [BPS20] used the same entropy analysis to show that this works for unweighted directed graphs. The same proof strategy generalizes to the weighted case as well. The argument is essentially that whenever there is a cut as in Lemma A.9(2), then some entropic measure of the flow matrix increases by $\Omega(n)$ after adding the matchings to the witness. Using the fact that the measure is always upper-bounded by $O(n \log n)$, this shows that we must arrive at Lemma A.9(1) in $T_{\text{KKOV}} = O(\log n)$ rounds.

Overall, the running time of Lemma A.7 is $O(m^2 \log^9 n)$ where m is the size of the matchings, since the witness contains at most $O(m \log n)$ edges. We omit the details on the entropy analysis and focus on proving the balanced sparse cut algorithm.

Lemma A.9. *There exists a universal constant $c_{A.9} > 0$ such that, given an n-vertex m-edge capacitated graph G and an edge set $F \subseteq E(G)$ with $\boldsymbol{c}_G(F) \leq c_{A.9} \cdot U/\log^4 n$ where $U \stackrel{\text{def}}{=} \sum_{e \in E} \boldsymbol{c}_G(e)$, there is a **deterministic** $O((m+n)^2 \log^7 n)$ time algorithm that either*

1. *outputs a vertex set X of $\text{vol}_G(X) \geq 0.88 \cdot \text{vol}_G(V)$ such that $G[X] \setminus F$ is an $\Omega(1/\log^{14} n)$-expander, or*
2. *a cut $S \subseteq V(G)$ with $\text{vol}_G(S), \text{vol}_G(\overline{S}) \geq \text{vol}_G(V)/50$ and $\min\{\boldsymbol{c}_G(E_G(S, \overline{S})), \boldsymbol{c}_G(E_G(\overline{S}, S))\} \leq \text{vol}_G(V)/400$.*

[28] Technically, the bipartition is made perfectly balanced by moving vertices in the larger side to the smaller side arbitrarily. See [BPS20, Section 7.1] for more details.

To prove Lemma A.9, we are going to reduce to the case where the graph is unweighted and has constant degree and then apply the following algorithm from [CGL+20]. We would like to point out that the following Lemma A.10 does not follow exactly from [CGL+20] because their algorithm is for undirected. We give a simplified proof in Section A.3 that is specialized to the specific form that we are using.

Lemma A.10. *There is a universal constant $c_{A.10} > 0$ such that, given an n-vertex unit-capacitated graph G with maximum degree $\Delta_G = O(1)$, an edge set $F \subseteq E(G)$ with $|F| < c_{A.10} \cdot n/\log^4 n$, and a constant $\psi < 1$, there is an $O(n^2 \log^7 n)$ time algorithm that either*

1. *outputs a vertex set $X \subseteq V$ of size $|X| \geq 0.89 \cdot n$ such that $G[X] \setminus F$ is an $\Omega(1/\log^{14} n)$-expander, or*
2. *outputs a cut $S \subseteq V(G)$ with $|S|, |\overline{S}| \geq n/20$ such that $\min\{|E_G(S, \overline{S})|, |E_G(\overline{S}, S)|\} \leq \psi \cdot \mathrm{vol}_G(V)$.*

Reducing to Unweighted Graphs. We first transform G into a unit-capacitated graph G' by throwing away low-capacity edges and replacing each remaining edge with a proportional number of parallel unit-capacitated edges. This uses ideas from [vdBCK+24]. Let $U \stackrel{\text{def}}{=} \sum_{e \in E} \boldsymbol{c}_G(e)$ be the sum of edge capacities, and let $\tau \stackrel{\text{def}}{=} U/(1000m)$ be a threshold. For each edge e with $\boldsymbol{c}_G(e) \geq \tau$, we add $\lceil \boldsymbol{c}_G(e)/\tau \rceil$ parallel copies of e into G'. For each vertex v, we also add $\lceil \deg_G(v)/(20\tau) \rceil$ self-loops on v to G' (each contributing 2 units of degree). Edges with small capacities are discarded. Let m' be the number of edges in G', and let F' be edges in G' that correspond to F.

Claim A.11. *It holds that $1099m \leq m' \leq 1102m$.*

Proof. We first lower bound m'. Edges with $\boldsymbol{c}_G(e) < \tau$ have a total capacities of at most $U/1000$, and for edges with $\boldsymbol{c}_G(e) \geq \tau$ we have $\lceil \boldsymbol{c}_G(e)/\tau \rceil \geq \boldsymbol{c}_G(e)/\tau$ copies of edges in G'. This contributes $\frac{U - U/1000}{\tau} = 999m$ edges. Likewise, we have $\lceil \deg_G(v)/(20\tau) \rceil \geq \deg_G(v)/(20\tau)$, and since the degrees sum up to $2U$, this contributes at least $100m$ edges.

For the upper bound, we have $\lceil \boldsymbol{c}_G(e)/\tau \rceil \leq \boldsymbol{c}_G(e)/\tau + 1$, and thus this contributes at most $U/\tau + m = 1000m + m = 1001m$ edges. Likewise, the degrees contribute at most $2U/(20\tau) + n \leq 101m$ edges. □

Claim A.12. *Let $X \subseteq V$. If $G'[X] \setminus F'$ is a ϕ-expander, then $G[X] \setminus F$ is also a $\phi/20$-expander.*

Proof. Note that the self loops in G' contributes a degree of at least $2 \cdot \deg_G(v)/(20\tau) = \deg_G(v)/(10\tau)$ to each v, which is not affected by the removal of F' and the restriction to X. In particular, this suggests that $\mathrm{vol}_{G'[X] \setminus F'}(S) \geq \mathrm{vol}_{G[X] \setminus F}(S)/(10\tau)$ for all $S \subseteq X$. Let $e_{(u,v)}$ denote the number of parallel edges (u, v) in $G' \setminus F'$. We know that if $e_{(u,v)} > 0$, then $\boldsymbol{c}_G(u, v) \geq e_{(u,v)} \cdot \tau/2$. Overall, this suggests that

$$\frac{\boldsymbol{c}(E_{G[X] \setminus F}(A, B))}{\min\{\mathrm{vol}_{G[X] \setminus F}(A), \mathrm{vol}_{G[X] \setminus F}(B)\}} \geq \frac{1}{20} \cdot \frac{|E_{G'[X] \setminus F'}(A, B)|}{\min\{\mathrm{vol}_{G'[X] \setminus F'}(A), \mathrm{vol}_{G'[X] \setminus F'}(B)\}}$$

for all $A, B \subseteq X$, and thus if $G'[X] \setminus F'$ is a ϕ-expander then $G[X] \setminus F$ is a $\phi/20$-expander. □

Claim A.13. *For any cut $S \subseteq V(G)$ we have $\mathrm{vol}_G(S) \geq \frac{10}{11}\tau \cdot \mathrm{vol}_{G'}(S) - U/250$ and $\boldsymbol{c}_G(E_G(S, \overline{S})) \leq \tau \cdot |E_{G'}(S, \overline{S})| + U/1000$.*

Proof. We first bound $\mathrm{vol}_G(S)$. For a vertex v we have $\deg_{G'}(v) \le 2 \cdot \lceil \deg_G(v)/(20\tau) \rceil + \deg_G(v)/\tau + e_v \le 11/10 \cdot \deg_G(v)/\tau + e_v + 2$, where e_v is the number of edges incident to v (the first term in the inequality comes from the self-loops we added; the second comes from edges incident to v). The bound follows since e_v sums up to at most $2m$. Likewise, for each edge e we have $\lceil \boldsymbol{c}_G(e)/\tau \rceil \le \boldsymbol{c}_G(e)/\tau + 1$ edges in G'. The $+1$ terms amount to $\tau \cdot m = U/1000$ units of capacities. □

Reducing to Low-Degree Graphs. We apply a second transformation that reduces the maximum degree of G' to a constant and obtain G'', following the ideas in [CGL+20]. For each vertex v we use [CGL+20, Theorem 2.4] to deterministically construct an $\Omega(1)$-expander H_v with maximum degree 9 containing $\deg_{G'}(v)$ vertices and replace v by H_v in G''. Let us number these vertices by $U_v \stackrel{\text{def}}{=} \{v_1, \ldots, v_{\deg_{G'}(v)}\}$. We also number the edges incident to v arbitrarily by $\{e_1, \ldots, e_{\deg_{G'}(v)}\}$. Now, for each edge $e = (u, v)$, if e is the i-edge incident to u and the j-th edge incident to v, then we add (u_i, v_j) to G''. Note that since G' might contain self-loops from the first transformation, we treat each self-loop on v as two edges in the list $\{e_i\}$ of v (since they contribute two units of degree each) and thus map it to an edge (v_i, v_j) for some i and j. Now G'' has n'' vertices and m'' edges where $n'' = 2m'$ and $m'' \le 9n'' + m' \le 20m'$, and has maximum degree 10. Again, let F'' denote edges in G'' that correspond to F' (and thus F).

Proof of Lemma A.9. We do the two transformations and apply Lemma A.10 with the constant ψ sufficiently small on G'' (with the edge set F'') that runs in $O(n''^2 \log^7 n'') = O((m+n)^2 \log^7 n)$ time. Indeed, note that if $\boldsymbol{c}_G(F) \le c_{A.9} \cdot U/\log^4 n$ for some constant $c > 0$ (as required by Lemma A.9), then $|F''| \le 2\boldsymbol{c}_G(F)/\tau \le 2000c \cdot m/\log^4 n$ which is less than $c_{A.10} \cdot n''/\log^4 n''$ for $c_{A.9}$ being sufficiently small, depending on $c_{A.10}$. Consider the following two cases that Lemma A.10 returns.

- If Lemma A.10 terminates in Case 1, i.e., it outputs a vertex set X'' of size $|X''| \ge 0.89n''$ so that $G''\{X''\} \setminus F''$ is an $\Omega(1/\log^{14} n)$-expander. Then, we first map X'' to an $X' \subseteq V(G)$ by including all $v \in V$ with $|U_v \cap X''| \ge |U_v|/100$. We claim that this new set X' in G' is an expander.

  Claim A.14. *It holds that $G'\{X'\} \setminus F'$ is an $\Omega(1/\log^{14} n)$-expander.*

  Proof. Consider any $\mathbf{vol}_{G'\{X'\}\setminus F'}$-respecting demand $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$. We may treat $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ as a demand on G'' by distributing the demand on v evenly to U_v. Since H_v is an $\Omega(1)$-expander, we can route the demand on $U_v \setminus X''$ to $U_v \cap X''$ by a flow in H_v with congestion $O(1)$. We then route the resulting demand, which is $\mathbf{vol}_{G''\{X''\}\setminus F''} \cdot O(1)$-respecting, in $G''\{X''\} \setminus F''$ with congestion $O(\log^{14} n)$. Such a flow naturally corresponds to a flow in $G'\{X'\} \setminus F'$ with congestion $O(\log^{14} n)$. Combining this with the flow in H_v proves the claim. □

  Now it follows from Claim A.12 that $G\{X'\} \setminus F$ is an $\Omega(1/\log^{14} n)$-expander. The volume of X' in G can also be bounded as $\mathrm{vol}_G(X') \ge \frac{10}{11}\tau \cdot \mathrm{vol}_{G'}(X') - U/250$ by Claim A.13, where

$$\mathrm{vol}_{G'}(X') = 2m' - \sum_{v \in V : |U_v \cap X''| < |U_v|/100} |U_v| \ge 2m' - n''/9 = 16m'/9.$$

  The inequalities above follow since $\sum_{|U_v \cap X''| < |U_v|/100} |U_v| \le \frac{100}{99} \cdot \sum |U_v \setminus X''| \le \frac{100}{99} \cdot 0.11n''$. This suggests that $\mathrm{vol}_G(X') \ge \frac{10}{11}\tau \cdot (\frac{16}{9} \cdot 1099m) - U/250 \ge 0.88\mathrm{vol}_G(V)$ (note that $\mathrm{vol}_G(V) = 2U$), as desired.

- On the other hand, if Lemma A.10 terminates in Case 2, i.e., it outputs a cut $S'' \subseteq V(G'')$ where $|S''| \geq n''/20$ and $\min\{|E_{G''}(S'', \overline{S''})|, |E_{G''}(\overline{S''}, S'')|\} \leq \psi \cdot n''$. Suppose that our constant-expanders H_v have expansion α_0 (as in [CGL+20, Theorem 2.4]). If $\psi \leq \alpha_0/2$ (this gives the bound on how small ψ should be), then by [CGL+20, Lemma 5.4] we can compute another cut $T'' \subseteq V(G'')$ such that $|T''| \geq |S''|/2$, $|\overline{T''}| \geq |\overline{S''}|/2$, and $\min\{|E_{G''}(T'', \overline{T''})|, |E_{G''}(\overline{T''}, T'')|\} \leq O(\min\{|E_{G''}(S'', \overline{S''})|, |E_{G''}(\overline{S''}, S'')|\})$. More importantly, for each $v \in V(G)$, either $U_v \subseteq T''$ or $U_v \cap T'' = \emptyset$, i.e., the cut T'' is consistent with the original vertices. We can thus naturally map T'' to a cut $S' \subseteq V(G')$ such that $\mathrm{vol}_{G'}(S') = |T''|$ and $\min\{|E_{G'}(S', \overline{S'})|, |E_{G'}(\overline{S'}, S')|\} \leq O(\psi) \cdot n'' = O(\psi) \cdot m''$. By Claim A.13, the cut S' in G has $\mathrm{vol}_G(S') \geq \frac{10}{11}\tau \cdot \mathrm{vol}_{G'}(S') - U/250 \geq U/25 = \mathrm{vol}_G(V)/50$. The same bound holds for $\mathrm{vol}_G(\overline{S})$ as well. Finally, we have $\min\{\boldsymbol{c}_G(E_G(S', \overline{S'})), \boldsymbol{c}_G(E_G(\overline{S'}, S'))\} \leq O(\psi) \cdot U + U/1000 \leq \mathrm{vol}_G(V)/400$ for ψ sufficiently small, as desired.

This concludes the proof. □

A.3 Proving Lemma A.10

We will reduce Lemma A.10 to the following similar lemma.

Lemma A.15. *There exists a universal constant $c_{A.15} > 0$ such that, given an n-vertex unit-capacitated graph G with maximum degree $\Delta_G = O(1)$, an edge set $F \subseteq E(G)$ with $|F| < c_{A.15} \cdot n/\log^4 n$, and a constant $\psi < 1$, there is a deterministic $O(n^2 \log^2 n)$ time algorithm that either*

1. *output $X \subseteq V(G)$ such that $G[X] \setminus F$ is an $\Omega(1/\log^{14} n)$-expander, or*
2. *outputs a cut $S \subseteq V(G)$ with $|\overline{S}| \geq |S| \geq \Omega(n/\log^4 n)$ such that $\min\{|E_G(S, \overline{S})|, |E_G(\overline{S}, S)|\} < \psi \cdot |S|$.*

We first prove Lemma A.10 assuming Lemma A.15.

Proof of Lemma A.10. We start with $S \stackrel{\text{def}}{=} \emptyset$ being initially empty. While $|S| < n/10$, we invoke Lemma A.15 on $G[V \setminus S]$ with the given value of ψ. If we get an $X \subseteq V \setminus S$ for which $G[X] \setminus F$ is an $\Omega(1/\log^{14} n)$-expander, then we return the same X. The size of X is at least $0.99 \cdot 0.9n \geq 0.89n$. Otherwise, we get a cut $S' \subseteq V \setminus S$, and we include S' into S, i.e., set $S \leftarrow S \cup S'$.

Now, suppose that we have reached a state where $|S| \geq n/10$ but none of the invocations of Lemma A.15 returns an expander subgraph X. Then, since the set S' returned by Lemma A.15 has $|S'| \leq n/2$, we have $|S| \leq 3n/4$. Now we show that we can extract a sparse cut within S. Indeed, this follows from the following observation from prior work (e.g., [BPS20, BBST24]) that a union of sparse cuts contains a large sparse cut.[29]

Claim A.16. *Consider a graph W and suppose there is a sequence of cuts $S_1, \ldots, S_k$ where S_i is in $W_i \stackrel{\text{def}}{=} W \setminus (S_1 \cup \cdots \cup S_{i-1})$ and $|S_i| \leq |V(W_i)|/2$. Then, there is an $O(|E(W)|)$ time algorithm that computes an $S \subseteq S_1 \cup \cdots \cup S_k$ with $|S| \geq (|S_1| + \cdots + |S_k|)/2$ and $\min\{|E_W(S, \overline{S})|, |E_W(\overline{S}, S)|\} < \sum_i \min\{|E_{W_i}(S_i, \overline{S_i})|, |E_{W_i}(S_i, \overline{S_i})|\}$.*

[29] This is in contrast to the undirecte case where the union of sparse cuts *is* a sparse cut itself.

By Claim A.16, we can compute a sparse cut $S' \subseteq S$ of size $|S'| \geq |S|/2 \geq n/20$ such that $\min\{|E_G(S', \overline{S'})|, |E_G(\overline{S'}, S')| < 2\psi \cdot |S'| \leq \psi \cdot n$. This S' can thus be used as the output. The running time is $O(\log^5 n) \cdot O(n^2 \log^2 n)$ since the size of the graph decreases by a $1 - \Omega(1/\log^4 n)$ factor for each invocation of Lemma A.15. This concludes the proof. □

We now show Lemma A.15 whose algorithm works as follows. We deterministically construct an $\Omega(1)$-expander H on n vertices via [CGL+20, Theorem 2.4], and then attempt to embed H into G. Note that H can be decomposed into $k = O(1)$ disjoint matchings and we embed each of them separately. This is done via the following algorithm in Lemma A.17 which is a simplified version of [CGL+20, Theorem 3.2] specialized to our use case.

Lemma A.17. *Given an n-vertex graph G with maximum degree $\Delta_G = O(1)$ and a matching M where $V(M) \subseteq V(G)$, a $z \in \mathbb{N}$ where $z < n/4$, and a constant $\psi < 1$, there is a deterministic $O(n^2 \log^2 n)$ time algorithm that either*

- *outputs a subset of $Z \subseteq M$ of size $|Z| \leq z$ such that M can be embedded into $G \cup Z$ with congestion $O(\log^2 n)$, or*
- *outputs a cut $S \subseteq V(G)$ with $|S|, |\overline{S}| \geq z/6$ such that $\Psi_G(S) \leq \psi$, where $\Psi_G(S) \stackrel{\text{def}}{=} \frac{\min\{|E_G(S,\overline{S})|, |E_G(\overline{S},S)|\}}{\min\{|S|, |\overline{S}|\}}$.*

If Lemma A.17 with $z \stackrel{\text{def}}{=} \frac{dn/\log^4 n}{k}$ with a small constant $d > 0$ returns a cut when embedding one of the matchings, then Lemma A.15 returns the same cut. Otherwise, overall, that the $\Omega(1)$-expander H is embeddable into $G \cup Z$ with congestion $O(\log^2 n)$ means that $G \cup Z$ is an $\Omega(1/\log^2 n)$-expander. Here $Z \stackrel{\text{def}}{=} \bigcup_{i \in [k]} Z_i$ where Z_i is the set of fake edges outputted by Lemma A.17 when embedding the i-th matching. Since $|F \cup Z| \leq (c+d)n/\log^4 n$, we can apply the following expander pruning algorithm to extract an $X \subseteq V(G)$ with $|X| \geq n - |V(G) \setminus X| \geq n - \text{vol}_{G \cup Z}(V(G) \setminus X) \geq 0.99n$ for c and d sufficiently small such that $G[X] \setminus F$ is an $\Omega(\log^{14} n)$-expander.

Lemma A.18 ([SP25, Theorem 4.1])**.** *Given an m-edge ϕ-expander G and $D \subseteq E(G)$ with $|D| < \frac{\phi^2}{200} m$, there is an $O(|D| \log m/\phi^4)$ time algorithm that outputs an $X \subseteq V(G)$ with $\text{vol}_G(V(G) \setminus X) \leq \frac{8}{\psi}|D|$ such that $(G \setminus D)[X]$ is an $\Omega(\phi^7)$-expander.*

It thus remains to prove Lemma A.17.

Proof of Lemma A.17. The algorithm works in phases, where in each phase we try to embed a subgraph $M' \subseteq M$ into G. We start with $M' \stackrel{\text{def}}{=} M$ and stop running the phases when $|M'| < z$. For each phase, we let G' be a copy of G and go through the edges $(u, v) \in M'$ in an arbitrary order. Let M_{embed} initially empty be the set of edges in M' that we embed in this phase. We check in $O(m)$ time if there exists an (u, v)-path P in G' with length at most $\ell \stackrel{\text{def}}{=} \frac{24\Delta_G \log n}{\psi} = O(\log n)$. If there is, then we embed $(u, v) \in M'$ into the path P, add (u, v) to M_{embed}, and remove P from G'. If we have successfully embedded at least $\frac{\psi}{12\ell}$ fraction of the edges in M', i.e., $|M_{\text{embed}}| \geq \frac{\psi}{12\ell} \cdot |M'|$, then we proceed to the next phase by setting $M' \leftarrow M' \setminus M_{\text{embed}}$. Once we reach the state where $|M'| < z$, we know that $M \setminus M'$ can be embedded into G with congestion $O(\ell \log n) = O(\log^2 n)$ (since there are $O(\ell \log n)$ phases), and thus we set $F \stackrel{\text{def}}{=} E(M')$ which makes M trivially embeddable into $G \cup F$ with congestion $O(\log^2 n)$.

Otherwise, if for one of the phases we did not successfully embed $\frac{\psi}{12\ell}$ fraction of the edges in the current M', then we will find a sparse cut in G. We first prove the following claim.

Claim A.19. *Given u and v in $\tilde{G} \subseteq G$ such that $\mathrm{dist}_{\tilde{G}}(u,v) > \ell$, there is an $O(m)$ time algorithm that finds a cut $S \subseteq V(\tilde{G})$ such that $|S| \leq n/2$ and $\min\{|E_{\tilde{G}}(S,\overline{S})|, |E_{\tilde{G}}(\overline{S},S)|\} < \psi/12 \cdot |S|$.*

Proof. This follows from a standard ball-growing argument. Let S_i denote the set of vertices reachable from u in $\tilde{G}$ by a path of length at most i. Assume without loss of generality that $|S_{\ell/2}| \leq n/2$, otherwise we swap u and v and work in the reversed G' instead. If none of the cut S_i for $i < \ell/2$ is sparse, then we have $|S_{i+1}| \geq |S_i| \cdot (1 + \psi/(12\Delta_G))$. This leads to a contradiction since $(1+\psi/(12\Delta_G))^{\ell/2} > n/2$. □

Now, we maintain a set $U \subseteq V$ which is initially empty. While $|U| < z/2$, since $|M' \setminus M_{\text{embed}}| \geq z/2$, an edge $(u,v) \in M' \setminus M_{\text{embed}}$ where $u, v \notin U$ exists (since M is a matching). We apply Claim A.19 and u and v to find a cut S in $\tilde{G} \stackrel{\text{def}}{=} G'[V \setminus U]$ and set $U \leftarrow U \cup S$. When this loop terminates, we know that $z/2 \leq |U| \leq z + n/2 \leq 3n/4$. We can then extract a balanced sparse cut from U via Claim A.16. Applying Claim A.16, we get an $S \subseteq V$ with $z/2 \leq |S| \leq 3n/4$ such that $\min\{|E_{G'}(S,\overline{S})|, |E_{G'}(\overline{S},S)|\} < \psi/6 \cdot |S|$. We also know that $V \setminus S \geq n/4 \geq |S|/3$ and therefore $\Psi_{G'}(S) \leq \psi/2$. Note that G' is obtained from G by removing $|M' \setminus M_{\text{embed}}| \leq \frac{\psi}{12\ell} \cdot z$ paths of length ℓ, and thus $\Psi_G(S) \leq \Psi_{G'}(S) + \psi/12 \cdot \frac{z}{\min\{|S|,|V\setminus S|\}} \leq \psi/2 + \psi/2 = \psi$. Finally, note that the running time is $O(\ell \log n) \times O(n^2) = O(n^2 \log^2 n)$. This completes the proof. □

B Discussion of Non-Stop Cut-Matching Game

In this section we discuss how our Lemma 7.3 is implied by the work of [FLL25]. Its main result, [FLL25, Theorem 5.1], is a combination of the abstract non-stop cut-matching game we described together with a specific matching player tailored to their use case, i.e., computing directed expander decomposition of an input graph.

How the Cut-Matching Game Works. To describe the differences in presentation between Lemma 7.3 and [FLL25], let us first outline how this non-stop cut-matching game works. The algorithm runs in $O(\log^2 n)$ iterations. In each iteration t, it maintains for each vertex v the "fraction" $\boldsymbol{d}_t(v) \leq \boldsymbol{d}(v)$ that is "alive", and let $A_t \stackrel{\text{def}}{=} \mathrm{supp}(\boldsymbol{d}_t)$ be the set of "alive" vertices at iteration t.[30] The algorithm also maintains a multi-commodity flow matrix $\boldsymbol{F}_t$, where $\boldsymbol{F}_t(u,v)$ denotes the unit of u-commodity that is sent to v. Initially, we have $A_0 = \mathrm{supp}(\boldsymbol{d})$ and $\boldsymbol{F}_0(u,u) = \boldsymbol{d}(u)$, indicating that we have $\boldsymbol{d}(u)$ units of u-commodity at u at the beginning of the cut-matching game. The goal of the cut-matching game is to have the commodities spread out among the vertices to prove expansion.

To do so, in each iteration t, based on $\boldsymbol{F}_t$, the algorithm computes a random $\boldsymbol{d}_t$-balanced bipartition (L,R) of A_t, i.e., $\boldsymbol{d}_t(L) = \boldsymbol{d}_t(R)$, and attempts to mix the commodities between L and R. The mixing is done through two "matchings", one from L to R and the other from R to L that indicate how much commodity currently at $\ell \in L$ is to be sent to $r \in R$ and vice versa. Ideally, the two matchings should be such that each $v \in A_t$ receives and sends out exactly $\boldsymbol{d}_t(v)$ units of commodities. However, this is not always possible in directed graphs. Instead, the algorithm computes $\boldsymbol{d}_{t+1}(v) \leq \boldsymbol{d}_t(v)$ as the minimum between the units v receives and sends out, and once $\boldsymbol{d}_{t+1}(v) < \boldsymbol{d}(v)/2$, v is completely "removed", i.e., we set $\boldsymbol{d}_{t+1}(v) \leftarrow 0$. After this, the flow matrix is updated by sending flows between L and R through the matchings (see [FLL25, Section 5.1]).

[30] This reflects the fact that in the weighted setting, a vertex might get partially matched, which is in contrast to the unweighted case where vertices are simply either matched or unmatched.

Finally, at the end of $O(\log^2 n)$ iterations, it is shown that $\boldsymbol{F}$ "mixes" well enough in the graph, proving expansion of $\boldsymbol{d}$ on the alive vertices A_t.

For more details, see [FLL25, Section 5] and in particular [FLL25, Algorithm 1].

Comparison Between the Two Presentations. We now compare the presentations of the cut-matching game in our paper and [FLL25]. To start, [FLL25, Theorem 5.1] is described as a combination of the above abstract game with a specific matching player implemented by running the standard unweighted push-relabel algorithm (in particular [FLL25, Lemma 5.19]) on the input graph to [FLL25, Theorem 5.1]. This matching player works by attempting to route from each $\ell \in L$ (with $\boldsymbol{d}_t(\ell)$ units of source demand) to each $r \in R$ (with $\boldsymbol{d}_t(r)$ units of sink demand), and it will compute a flow $\boldsymbol{f}$ partially routing the demand and a sparse cut S such that $\boldsymbol{f}$ rotues the demand in $V \setminus S$ almost perfectly, up to $O(\boldsymbol{d}_t(S))$ units of source/sink. A matching can be extracted from this flow through standard path decomposition, similar to what we did in Lemma 7.5.

The main difference is that for their end result, [FLL25, Theorem 5.1] wants to find a sequence of *non-overlapping* cuts in the input graph such that the uncut part is expanding (in the whole graph). To do so, even though a vertex $v \in S$ might still be partially matched by $\boldsymbol{f}$, the algorithm will completely remove it (i.e., set $\boldsymbol{d}_{t+1}(v) \leftarrow 0$ for $v \in S$). For other vertices, some of them might not fully send/receive $\boldsymbol{d}_t(v)$ units of commodities due to the flow $\boldsymbol{f}$ not routing the demand perfectly. The algorithm of [FLL25] proceeds to set $\boldsymbol{d}_{t+1}(v)$ to be the minimum between the amount v sends and receives, and removes those with $\boldsymbol{d}_{t+1}(v) < \boldsymbol{d}(v)/2$, as described earlier. Having the vertices in the cut completely removed allows them to add the cut S to their sequence, and then solve the flow problems of future iterations *only on the uncut part of the graph* to make the cuts non-overlapping.

Our formulation differs from [FLL25] mainly in that our cut-matching game is not concerned about the input graph, hence there is no notion of sparse cuts for Lemma 7.3. In particular, given $\boldsymbol{d}$, the $\boldsymbol{d}'$ vector that we use (in Definition 7.2) in each iteration will simply be the $\boldsymbol{d}_t$ vector in the cut-matching game, and the bidirectional matchings will naturally define the matching matrix $\boldsymbol{M}_t$ of [FLL25] to update $\boldsymbol{F}_t$. Crucially, since there is no notion of cuts in the abstract form of the cut-matching game that we use, we *do not* remove vertices in the cut like [FLL25]. Instead, all vertices v just set their $\boldsymbol{d}_{t+1}(v)$ based on the amount they receive and send, and only those with $\boldsymbol{d}_{t+1}(v) < \boldsymbol{d}(v)/2$ are removed. It is easy to see this does not invalidate the proofs in [FLL25]. The final measure $\tilde{\boldsymbol{d}}$ in Lemma 7.3 will simply be the final alive demand $\boldsymbol{d}_t$. Since we ensure that each bidirectional matching is perfect up to some $O(1/\log^2 n) \cdot \boldsymbol{d}(V)$ units of demand, over $O(\log^2 n)$ iterations we only remove a small constant fraction from $\boldsymbol{d}_t(V)$. The removal of $\boldsymbol{d}_t(V)$ caused by completely setting $\boldsymbol{d}_t(v)$ to 0 when it reaches below $\boldsymbol{d}(v)/2$ can also be charged to the former quantity, and thus we can guarantee a bound of $\tilde{\boldsymbol{d}}(V) \geq \frac{7}{8}\boldsymbol{d}(V)$ in Lemma 7.3.

Our formulation also differs from [FLL25] in where and how the final subset of demand is expanding. In [FLL25, Theorem 5.1], since it is explicit in the theorem setup that the matchings are embeddable with low congestion in the input graph G, the final demand is $O(\phi/\log^2 n)$-expanding in G, where $1/\phi$ is the congestion of each embedding. Inspecting the proof of this fact (see [FLL25, Section 5.4]) and as standard in the cut-matching game literature, that $\boldsymbol{d}_t$ is $O(\phi/\log^2 n)$-expanding in G is because each $\boldsymbol{d}_t$-respecting demand can be routed by (1) following the flow matrix $\boldsymbol{F}_t$ (which "mixes" well at the end of the game)—indeed, [FLL25, Page 26, Proof of Lemma 5.26] defined an explicit flow routing the demand—and (2) using the fact that $\boldsymbol{F}_t$ can be routed in G through the embedding with low congestion $O(\log^2 n/\phi)$.

Since our formulation does not involve the underlying graph G, we simply say that $\boldsymbol{F}_t$ can be

routed with congestion $O(1)$ in the witness $W \stackrel{\text{def}}{=} M_1 \cup M_2 \cup \cdots \cup M_k$. Indeed, based on how $\boldsymbol{F}_t$ is updated (see [FLL25, Section 5.1]), the graph W is defined so that $\boldsymbol{F}_t$ can be trivially routed with congestion $O(1)$—the routing just follows how $\boldsymbol{F}_t$ is "mixed" between L and R in each iteration. This establishes the expansion guarantee listed in Lemma 7.3. Finally, we need a low-hop guarantee in this routing on W. This also follows fairly simply from the definition of W. The routing through W is obtained as $\boldsymbol{F}_t$ evolves as new matchings are added: In each iteration we simply send flow through the newly added matching M_t to update $\boldsymbol{F}_t$. Since there are only $T_{\text{KRV}} = O(\log^2 n)$ matchings, the hop of the routing is naturally T_{KRV}.

C Omitted Proofs

In this section we provide proofs that are omitted from the main body of the paper.

Lemma 7.4. *Let $F \subseteq E$ and $\mathcal{H}$ be the hierarchy of $G \setminus F$. Let A be the shortcut induced by $\mathcal{H}$ with capacity scale $\psi \in 1/\mathbb{N}$, and $G_A = G \cup A$ be the shortcut graph. Given a diffusion instance $\mathcal{I} = (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with ψ-integral demand and parameter $\kappa \in \mathbb{N}$, there is a deterministic algorithm that runs on G_A in $O(n^2 \log^4 n \cdot (\kappa + 1/\psi))$ time and finds a ψ-integral flow $\boldsymbol{f}$ in G_A with congestion $O(\kappa \log n)$ and the representation of a ψ-integral $(\boldsymbol{w}_{\mathcal{H}}, h)$-short path decomposition of $\boldsymbol{f}$ for some $h = O(n \log n \cdot (\kappa + 1/\psi))$.*

Additionally, if $|\boldsymbol{f}| < \|\boldsymbol{\Delta}\|_1$, then the algorithm also returns a cut $\emptyset \neq S \subsetneq V$ with $\boldsymbol{\nabla}_{\boldsymbol{f}}(S) = 0$ and $\boldsymbol{\Delta}_{\boldsymbol{f}}(\overline{S}) = 0$ of size

$$\boldsymbol{c}(E_{G_A}(S, \overline{S})) \leq \frac{41 \min\{\boldsymbol{\Delta}(S), \boldsymbol{\nabla}(\bar{S})\} + \min\{\mathbf{vol}_F(S), \mathbf{vol}_F(\bar{S})\}}{\kappa}.$$

We will use the following lemma that finds a path decomposition of a flow and the weights of the paths. Computing a path decomposition in capacitated graphs is fairly standard using a dynamic tree, and by simply augmenting the dynamic tree to maintain the weights we can obtain the desired property.

Lemma C.1 ([ST83])**.** *Given a flow $\boldsymbol{f}$ on an m-edge graph G with edge weights $\boldsymbol{w}$, there is an $O(m \log m)$ time algorithm that computes the representation $\{(\lambda_i, s_i, t_i)\}_{i \in [k]}$ together with $\{W_i\}_{i \in [k]}$ for some path decomposition $(C_{\boldsymbol{f}}, \{(\lambda_i, P_i)\}_{i \in [k]})$ of $\boldsymbol{f}$, where $\boldsymbol{w}(P_i) \leq W_i$.*

Proof of Lemma 7.4. We run Lemma 4.1 multiple iterations, each time routing a large fraction of demand or return the desired cut. Let $(\boldsymbol{\Delta}^{(1)}, \boldsymbol{\nabla}^{(1)}) \stackrel{\text{def}}{=} (\boldsymbol{\Delta}, \boldsymbol{\nabla})$ be the initial demand. For each iteration i, we run Lemma 4.1 to route $(\boldsymbol{\Delta}^{(i)}, \boldsymbol{\nabla}^{(i)})$ in G_A, getting a flow $\boldsymbol{f}^{(i)}$. If $|\boldsymbol{f}^{(i)}| \geq \|\boldsymbol{\Delta}^{(i)}\|_1/2$, then we run Lemma C.1 on $\boldsymbol{f}^{(i)}$ with the weight function $\boldsymbol{w}_{\mathcal{H}}$. Let $h_{4.1} = O(n \cdot (\kappa + 1/\psi) \cdot \log n)$ be the bound on the average weight of flow paths returned by Lemma 4.1. Let $\tilde{\boldsymbol{f}}^{(i)} \stackrel{\text{def}}{=} \sum_{i \in [k]: W_i \leq 2h} \lambda_i \boldsymbol{f}_{P_i}$ consists of only paths with weight bounded by $2h$. By Markov's inequality, $\tilde{\boldsymbol{f}}^{(i)}$ routes at least half of the demand that $\boldsymbol{f}^{(i)}$ does, i.e., $|\tilde{\boldsymbol{f}}^{(i)}| \geq \|\boldsymbol{\Delta}^{(i)}\|_1/4$. We set $(\boldsymbol{\Delta}^{(i+1)}, \boldsymbol{\nabla}^{(i+1)}) \stackrel{\text{def}}{=} (\boldsymbol{\Delta}^{(i)}_{\tilde{\boldsymbol{f}}^{(i)}}, \boldsymbol{\nabla}^{(i)}_{\tilde{\boldsymbol{f}}^{(i)}})$ be the residual demand (which is ψ-integral since $\tilde{\boldsymbol{f}}^{(i)}$ is) and continue to the next iteration. If all the $O(\log n)$ executions of Lemma 4.1 route at least half of the demand (and thus, the corresponding $\tilde{\boldsymbol{f}}^{(i)}$ routes at least a quarter), then $\boldsymbol{f} \stackrel{\text{def}}{=} \tilde{\boldsymbol{f}}^{(1)} + \cdots + \tilde{\boldsymbol{f}}^{(O(\log n))}$ routes $(\boldsymbol{\Delta}, \boldsymbol{\nabla})$ with congestion $O(\kappa \log n)$. Additionally, $\boldsymbol{f}$ trivially admits a $(\boldsymbol{w}_{\mathcal{H}}, 2h_{4.1})$-short path decomposition since each $\tilde{\boldsymbol{f}}^{(i)}$ by construction does. The representation of this decomposition can also be easily computed from that of each $\tilde{\boldsymbol{f}}^{(i)}$ (which we obtained from Lemma C.1).

Otherwise, we have $|\boldsymbol{f}^{(i)}| < \|\boldsymbol{\Delta}^{(i)}\|/2$ and get a cut $S^{(i)}$ from Lemma 4.1. We have

$$\begin{aligned}\boldsymbol{c}(E_{G_A}(S^{(i)}, \overline{S^{(i)}})) &\leq \frac{41|\boldsymbol{f}^{(i)}| + \min\{\mathrm{vol}_F(S^{(i)}), \mathrm{vol}_F(\overline{S^{(i)}})\}}{\kappa} \\ &\leq \frac{41\min\{\boldsymbol{\Delta}^{(i)}(S^{(i)}), \boldsymbol{\nabla}^{(i)}(\overline{S^{(i)}})\} + \min\{\mathrm{vol}_F(S^{(i)}), \mathrm{vol}_F(\overline{S^{(i)}})\}}{\kappa} \\ &\leq \frac{41\min\{\boldsymbol{\Delta}(S^{(i)}), \boldsymbol{\nabla}(\overline{S^{(i)}})\} + \min\{\mathrm{vol}_F(S^{(i)}), \mathrm{vol}_F(\overline{S^{(i)}})\}}{\kappa},\end{aligned}$$

where we used that $|\boldsymbol{f}^{(i)}| \leq \boldsymbol{\Delta}^{(i)}(S^{(i)})$ since $\boldsymbol{\Delta}^{(i)}(S) \geq \boldsymbol{\Delta}^{(i+1)}(S^{(i)}) = \|\boldsymbol{\Delta}^{(i+1)}\|_1$ while $|\boldsymbol{f}^{(i)}| < \|\boldsymbol{\Delta}^{(i)}\|_1/2 < \|\boldsymbol{\Delta}^{(i+1)}\|_1$, and that $|\boldsymbol{f}^{(i)}| \leq \boldsymbol{\nabla}^{(i)}(S^{(i)})$ since $\boldsymbol{\nabla}^{(i+1)}(S^{(i+1)}) = 0$ and therefore $\boldsymbol{\nabla}^{(i)}(\overline{S^{(i+1)}}) \geq \boldsymbol{\nabla}^{(i+1)}(\overline{S^{(i+1)}}) \geq \|\boldsymbol{\Delta}^{(i+1)}\|_1 > |\boldsymbol{f}^{(i)}|$. Moreover, we have $(\boldsymbol{\Delta}^{(i+1)}, \boldsymbol{\nabla}^{(i+1)}) = (\boldsymbol{\Delta}_{\boldsymbol{f}}, \boldsymbol{\nabla}_{\boldsymbol{f}})$ for $\boldsymbol{f} \stackrel{\text{def}}{=} \tilde{\boldsymbol{f}}^{(1)} + \cdots + \tilde{\boldsymbol{f}}^{(i)}$, and thus $\boldsymbol{f}$ routes all sources except than those from $S^{(i)}$, and $S^{(i)}$ is fully saturated. Again, $\boldsymbol{f}$ admits a $(\boldsymbol{w}_{\mathcal{H}}, 2h_{4.1})$-short path decomposition which we can compute from that of $\tilde{\boldsymbol{f}}^{(i)}$. The congestion of $\boldsymbol{f}$ in this case is, similar to above, $O(\kappa \log n)$. □