Low-Step Multi-Commodity Flow Emulators

Bernhard Haeupler*
bernhard.haeupler@inf.ethz.ch
INSAIT & ETH Zürich

D Ellis Hershkowitz
delhersh@brown.edu
Brown University

Jason Li
jmli@cs.cmu.edu
CMU

Antti Roeyskoe*
antti.roeyskoe@inf.ethz.ch
ETH Zürich

Thatchaphol Saranurak†
thsa@umich.edu
University of Michigan

Abstract

We introduce the concept of *low-step multi-commodity flow emulators* for any undirected, capacitated graph. At a high level, these emulators contain approximate multi-commodity flows whose paths contain a small number of edges, shattering the infamous *flow decomposition barrier* for multi-commodity flow.

We prove the existence of low-step multi-commodity flow emulators and develop efficient algorithms to compute them. We then apply them to solve constant-approximate k-commodity flow in $O((m+k)^{1+\epsilon})$ time. To bypass the $O(mk)$ flow decomposition barrier, we represent our output multi-commodity flow *implicitly*; prior to our work, even the existence of implicit constant-approximate multi-commodity flows of size $o(mk)$ was unknown.

Our results generalize to the *minimum cost* setting, where each edge has an associated cost and the multi-commodity flow must satisfy a cost budget. Our algorithms are also parallel.

*Partially funded by the European Union's Horizon 2020 ERC grant 949272.
†Supported by NSF grant CCF-2238138.

Contents

1 Introduction

In the maximum flow problem, we are given an edge-capacitated graph $G = (V, E)$ and two vertices $s, t \in V$ with the aim to send as much flow as possible from s to t. Maximum flow is a fundamental problem in combinatorial optimization with a long history, from the classic Ford-Fulkerson algorithm [CLRS22] to the recent breakthrough almost-linear time algorithm of Chen et al. [CKL+22]. The maximum flow problem also exhibits rich structural properties, most notably in the *max-flow min-cut theorem*, which states that the value of the maximum flow between s and t is equal to the size of the minimum edge cut separating s and t.

A well-studied generalization of the maximum flow problem is the maximum *multi-commodity flow* problem. Here, we are given $k \geq 1$ pairs $(s_i, t_i)_{i\in[k]}$ of vertices and wish to maximize the amount of flow sent between each pair. On the algorithmic side, this problem can be solved in polynomial time by a linear program, and there has been exciting recent progress towards obtaining faster algorithms, both exact [vdBZ23] and $(1+\epsilon)$-approximate [She17]. However, these algorithms output the flow for each commodity explicitly and, hence, must take at least $\Omega(mk)$ time because there exists a graph such that the total size of flow representation overall k commodities of any α-approximate solution has size at least $\Omega(mk/\alpha)$. Designing multi-commodity flow algorithms is further complicated by the loss of structure exhibited in the single-commodity setting. Specifically, the max-flow min-cut equality no longer holds in the multi-commodity case, and the flow-cut gap (i.e. the multiplicative difference between the max-flow and min-cut) is known to be $\Theta(\log n)$ for undirected graphs [LR99] and $\Omega(n^{1/7})$ for directed graphs [CK09].

The algorithmic and structural issues above suggest that outputting a (near) optimal multi-commodity flow in time less than $O(mk)$ may be impossible. Even for the problem of efficiently approximating *the value* of the optimal multi-commodity flow in undirected graphs, current techniques (e.g. [Rac02, RST14]) fail to produce approximations below the multi-commodity flow-cut gap, i.e., no $o(\log n)$-approximations running in $o(mk)$ time are known.

1.1 Our Results

In this paper, we break the $O(mk)$-time barrier by giving $(m+k)^{1+\epsilon}$-time algorithms with $O(1)$-approximations to the value of the maximum k-commodity flow on undirected graphs for any constant $\epsilon > 0$.

We consider both concurrent and non-concurrent flow problems. In the concurrent flow problem, given a capacitated graph and a demand function, the goal is to find a maximum-value capacity-respecting flow routing a multiple of the demand. In the non-concurrent flow problem, given a capacitated graph and a set of vertex pairs, the goal is to find a maximum-value capacity-respecting flow routing flow only between vertex pairs in the given set.

Our results can be informally summarized as follows:

Theorem 1.1 (Constant-Approximate Concurrent/Non-Concurrent Flow (informal))**.** *For every constant $\epsilon \in (0, 1)$, there exists a $(m+k)^{1+\mathrm{poly}(\epsilon)}$-time $O(2^{-1/\epsilon})$-approximate algorithm for the concurrent and non-concurrent multicommodity flow value problems, where k is the number of demand pairs. The algorithms work in parallel with depth $(m+k)^{\mathrm{poly}(\epsilon)}$.*

The key to the above result is a powerful new tool we introduce called *low-step (multi-commodity) flow emulators*. Informally, a low-step flow emulator of an undirected graph is another graph which

contains approximate multi-commodity flows whose flow paths contain a small number of edges. Because they are not based on cuts, such emulators face no $\Omega(\log n)$ flow-cut barriers, unlike the above-mentioned cut sparsifiers. It is instructive to view low-step flow emulators as a generalization of *hopsets*, which are graphs that contain approximate shortest paths of low step-length (but do not respect capacities). We additionally give efficient algorithms to construct these objects, and generalize them to achieve two additional important properties:

- **Cost-Constrained / Length-Constrained:** Our emulators generalize to various min-cost multi-commodity flow problems, where each edge has an associated length or cost (independent of its capacity) and any flow path must not exceed a given bound on the length or cost.
- **Implicit Mappings:** Our flow emulators also support *implicit* flow mappings from the emulator back to the original graph. In other words, we can even maintain an implicit solution to a constant-approximate multi-commodity flow, and Theorem 1.1 can be modified to output such an implicit representation for a flow of the approximated value. (Recall that implicit solutions are required to obtain any $o(mk)$ running time.) With this implicit solution, we can answer the following queries in $O((m+k)^{1+\epsilon})$ time: given any subset of the k commodities, return the union of the flows of each of these commodities. As this is a single-commodity flow, it is representable explicitly within the allotted time.

The combination of the two generalizations will allow us to apply flow *boosting* in the spirit of Garg-Konemann [GK07] to obtain the above algorithms, and further allows us to obtain the above multi-commodity flow result subject to a cost constraint.

1.2 Our Techniques

At a high-level, our approach is as follows. First, we compute low-step emulators by building on recent advances in length-constrained expander decompositions. Next, we use our low-step emulators to compute (implicit) flows on the original graph with potentially large congestion. Lastly, we use "boosting" to reduce this congestion down to a constant to get our final flow approximation.

Step 1: Low-Step Emulators via Length-Constrained Expander Decompositions. Our techniques build on recent developments in *length-constrained expander decompositions* [HRG22, HHG22, HHS23, HHT23, HHT24]. At a high level, h-length expanders are graphs with edge lengths and capacities for which any "reasonable" multi-commodity demand can be routed along (about) h-length paths with low congestion (by capacity). Informally, a reasonable demand is one where each demand pair (s_i, t_i) is within h by distance (so that sending flow from s_i to t_i along about-h-length paths is actually possible), and each vertex does not belong to too many demand pairs (so that the degree of the vertex is not an immediate bottleneck for congestion). Recent work [HRG22, HHT24] has studied how to, in almost-linear time, compute *length-constrained expander decompositions* which are length-increases—a.k.a. moving cuts—to the graph that make it an h-length expander. One caveat, however, is that these algorithms run in time polynomial in the *length* parameter h of the length-constrained expander decomposition.

In this paper, we remove this polynomial dependency of h by way of the above-mentioned low-step emulators. Specifically, we "stack" low-step emulators on top of each other with geometrically

increasing lengths, similar to how hopsets are stacked in parallel algorithms [Coh00]. Each low-step emulator is responsible for flow paths of its corresponding length, and the union of all low-step emulators obeys all distance scales simultaneously.

Our construction of low-step emulators comes with an *embedding* of the emulator into the base graph: each edge of the emulator maps to a small-length path in the base graph such that the set of all embedded paths has low congestion. When emulators are stacked on top of each other, an edge at the top level expands to a path at the previous level, each of whose edges expands to a path at the previous level, and so on. This hierarchical structure allows us to provide the aforementioned implicit representation of very long paths while keeping the total representation size small.

Step 2: Flows on Emulators. Our next contribution is a fast algorithm that computes a multi-commodity flow on a low-step emulator, where an approximate flow with small representation size is indeed possible (since flow paths now have low step-length). We explicitly compute such a flow, and then (implicitly) map each flow path down the hierarchical structure to form our final implicit flow on the input graph.

By setting parameters appropriately, we can guarantee a bicriteria approximation with constant-approximate cost and n^ϵ-approximate congestion for any constant $\epsilon > 0$. The n^ϵ-approximate congestion appears in both the multi-commodity flow algorithm on a low-step emulator, and the hierarchical embeddings of the emulators into the base graph.

Step 3: Boosting Away Congestion. Finally, through Garg and Konemann's approach [GK07] based on multiplicative weight updates, we can *boost* the congestion approximation of n^ϵ down to the cost approximation, which is constant. While Garg and Konemann's algorithm requires a near-linear number of calls to (approximate) shortest path, we only require roughly n^ϵ calls to n^ϵ-approximate congestion, constant-approximate cost multi-commodity flow. Our final result is a multi-commodity flow with constant-approximate cost and congestion which is implicitly represented by the hierarchical emulator embeddings. Given any subset of commodities, we can then output the union of their flows by collecting the flow down the hierarchy of embeddings.

2 Overview

The rest of the paper is organized as follows. The first part of the paper, Sections 4 to 6, develops the theory of low-step emulators. In Section 4, we study the h-length-constrained version of low-step emulators. We provide an existential result and an algorithm with a running time that depends on $\text{poly}(h)$. These results are obtained by reducing to h-length-constrained expander decomposition. To remove the dependency on $\text{poly}(h)$ in the running time, in Section 5 we demonstrate how to bootstrap the construction of h-length-constrained low-step emulators with small h to the large ones without spending $\text{poly}(h)$ in the running time. Finally, by combining h-length-constrained low-step emulators from different h values, we obtain our low-step emulator in Section 6.

The next part of the paper, Sections 7 and 8, develops a fast k-commodity flow algorithm designed for running on top of low-step emulators. The crucial property of this algorithm is that its running time is independent of k. In Section 7, we first show a prerequisite subroutine called "routing on an expansion witness" for efficiently routing a length-constrained flow. This subroutine

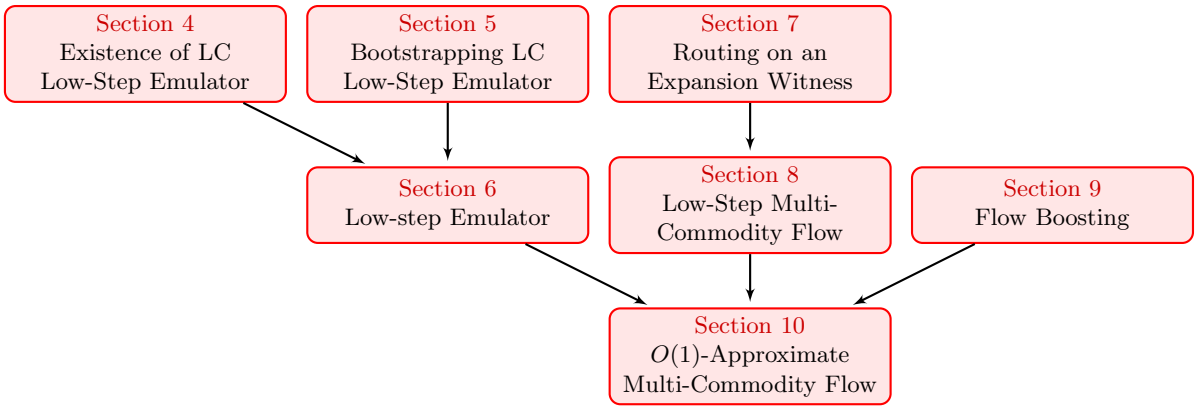

Figure 1: Dependencies between sections in this paper.

assumes that the input graph is a length-constrained expander and also requires the expansion witness of this expander. We then use this subroutine in Section 7 to develop an algorithm that computes a t-step k-commodity flow (each flow path contains at most t edges) with a running time of $m^{1+\epsilon}\text{poly}(t)$. The algorithms have an $O(1)$-approximation in step and total length, but an n^{ϵ}-approximation in congestion. Note that the dependency on $\text{poly}(t)$ in the running time is inconsequential, as the algorithm will be run on a t-step emulator for constant t. The issue of the large approximation factor in congestion will be addressed in the next part.

The final part of the paper, Sections 9 and 10, shows how we can combine the techniques from the previous two parts to achieve an $O(1)$-approximate k-commodity flow in $(m+k)^{1+\epsilon}$ time. To address the problem of the n^{ϵ}-approximation in congestion, in Section 9, we give a boosting algorithm that improves the congestion approximation to $O(1)$. Finally, in Section 10, we combine these tools together to obtain the final result.

The dependencies between sections are summarized in Figure 1.

3 Preliminaries

Let $G = (V, E)$ be a graph. We denote the number of vertices of G by $n := |V|$ and the number of edges by $m := |E|$, respectively. Let $u_G : E \to \mathbb{R}_{>0}$ denote the edge capacity function of G. The degree of a vertex v is $\deg_G(v) = \sum_{(v,w)\in E} u_G(v,w)$. Note that a self-loop (v,v) contributes $u_G(v,v)$ to the degree of v. Let $\ell_G : E \to \mathbb{R}_{>0}$ denote the edge length function of G.

In this paper, we will use the word "hop" and "length" interchangeably[1]. A path from vertex v to vertex w is called a (v,w)-path. For any path P, the length $\ell_G(P) = \sum_{e\in E} \ell_G(e)$ of the path equals the total length of its edges, and the step-length $|P|$ of the path is simply the number of edges in P. The *distance* between vertices v and w is $\text{dist}_G(v,w) = \min_{P:(v,w)\text{-path}} \ell_G(P)$. A *ball* of radius r around a vertex v is $\text{ball}_G(v,r) = \{w \mid \text{dist}_G(v,w) \le r\}$. The diameter $\text{diam}_G(S) := \max_{v,w\in S} \text{dist}_G(v,w)$ of a vertex set S is the maximum distance between two vertices in the set.

[1]We sometimes use the term "hop" instead of "length" in this paper and even use h for parameters related to length so that terminologies are consistent with previous literature [HRG22, HHT24, HHG22, HHT23] on length-constrained expanders. In the previous papers, they used "hop" and h because edges usually have unit length.

We assume all graphs to have polynomially bounded integral edge capacities and lengths. This assumption allows us to write $\log(\max_e \ell(e)), \log(\max_e u(e)) = O(\log N)$, which simplifies notation.

Assumption 3.1. *The capacity and length of all edges in the input graphs are integral and at most* $N = \text{poly}(n)$.

Although we will work with fractional capacities and lengths in the body of the paper, all these values will always be a multiple of some integer reciprocal $1/\text{poly}(n)$ and are upper bounded by $\text{poly}(n)$; by scaling, we can always obtain integral polynomially-bounded capacities and lengths.

Multicommodity Flows. A *(multicommodity) flow* F in G is a function that assigns each simple path P in G a flow value $F(P) \geq 0$. We say P is a *flow-path* of F if $F(P) > 0$, and write $\text{path}(F) = \text{supp}(F) := \{P : F(P) > 0\}$ for the set of flow-paths, equivalently the *support* of F, and occasionally abuse notation to write $P \in F$ to mean $P \in \text{supp}(F)$. $|\text{path}(F)|$ is called the *path count* of F, and the value of the flow F is $\text{val}(F) := \sum_P F(P)$. P is a (v, w)*-flow-path* of F if P is both a (v, w)-path and a flow-path of F. The (v, w)-flow $f_{(v,w)}$ of F is the flow for which $f_{(v,w)}(P) = F(P)$ if P is a (v, w)-path, otherwise $f_{(v,w)}(P) = 0$.

The *congestion of* F *on an edge* e is $\text{cong}_F(e) = F(e)/u_G(e)$ where $F(e) = \sum_{P:e\in P} F(P)$ denotes the total flow value of all paths going through e. The *congestion* of F is $\text{cong}_F = \max_{e\in E(G)} \text{cong}_F(e)$. The *length* of F, denoted by $\text{leng}_F = \max_{P\in\text{supp}(F)} \ell(P)$, measures the maximum length of all flow-paths of F. The *(maximum) step-length* of F is $\text{step}_F = \max_{P\in\text{supp}(F)} |P|$, which measures the maximum number of edges in all flow-paths of F. Note that step_F is completely independent from leng_F. We sometimes write $\text{cong}_{G,F}, \text{leng}_{G,F}, \text{step}_{G,F}$ to emphasize that they are with respect to G.

Edge Representation of Flows. When we want to emphasize an edge (v, w) is undirected, we use the notation $\{v, w\}$. Let $\overleftrightarrow{E}(G)$ denote the set of bidirectional edges of G. That is, for each edge $e = \{v, w\} \in E(G)$, we have $(v, w), (w, v) \in \overleftrightarrow{E}(G)$. The *edge representation* of F in G is a function $\text{flow}_F : \overleftrightarrow{E}(G) \to \mathbb{R}_{\geq 0}$ where, for each edge $e = \{v, w\} \in E(G)$, let $\text{flow}_F(v, w)$ and $\text{flow}_F(w, v)$ denote the total flow-value of F routed through e from v to w and from w to v, respectively. We sometimes use $F(v, w) = \text{flow}_F(v, w)$ and $F(w, v) = \text{flow}_F(w, v)$ for convenience. Note that $F(\{v, w\}) = F((v, w)) + F((w, v))$.

Demands. A *demand* $D : V \times V \to \mathbb{R}_{\geq 0}$ assigns a value $D(v, w) \geq 0$ to each ordered pair of vertices. Given a flow F, the *demand routed/satisfied by* F is denoted by D_F where, for each $u, v \in V$, $D_F(u, v) = \sum_{P \text{ is a } (u,v)\text{-path}} F(P)$ is the value of (u, v)-flow of F. We say that a *demand* D *is routable in* G *with congestion* η*, length* h*, and* t *steps* if there exists a flow F in G where $D_F = D$, $\text{cong}_F \leq \eta$, $\text{leng}_F \leq h$, and $\text{step}_F \leq t$. The total demand size is $|D| = \sum_{u,v} D(u, v)$. The support of D is $\text{supp}(D) = \{(v, w) \mid D(v, w) > 0\}$.

A *node-weighting* $A : V \to \mathbb{R}_{\geq 0}$ of G assigns a value $A(v)$ to each vertex v. The *size* of A is denoted by $|A| = \sum_v A(v)$. For any two node-weightings A and B, we write $A \leq B$ if $A(v) \leq B(v)$ for all v. We say D is A*-respecting* if both $\sum_w D(v, w) \leq A(v)$ and $\sum_w D(w, v) \leq A(v)$. We say that D is *degree-respecting* if D is $\deg_G$-respecting. Next, D is h*-length-bounded* (or simply h*-length* for short) if it assigns positive values only to pairs that are within distance at most h, i.e.,

$D(v,w) > 0$ implies that $\text{dist}_G(v,w) \leq h$. The *load* of a demand D is defined as the node weighting $\text{load}(D)(v) = \sum_{w \in V} D(v,w) + D(w,v)$.

3.1 Length-Constrained Expanders

Moving Cuts. An *h-length moving cut* $C : E \to \{0, \frac{1}{h}, \frac{2}{h}, \ldots, 1\}$ assigns a fractional which is a multiple of $\frac{1}{h}$ between zero and one to each edge e. The *size* of C is defined as $|C| = \sum_e u(e) \cdot C(e)$. We denote with $G - C$ the graph G with a new length function $\ell_{G-C}(e) = \ell_G(e) + h \cdot C(e)$ for all e. We refer to $G - C$ as the *graph G after cutting C* or *after applying the moving cut C*.

Given a h-length demand D, we usually work with a (hs)-length moving cut C where $s > 1$. The *total demand of D separated by C* is then denoted by

$$\text{sep}_{hs}(C, D) = \sum_{(v,w):\text{dist}_{G-C}(v,w) > hs} D(v,w),$$

which measures the total amount of demand between vertex pairs whose distance is increased (from at most h, as D is h-length) to strictly greater than hs by applying the cut C. Note that an integral moving cut (i.e. one for which $C(e) \in \{0,1\}$) functions somewhat like a classic cut: the size $|C|$ is the total capacity as a classic cut, and $\text{sep}_{hs}(C,D) = \sum_{(v,w):\text{all } hs\text{-length } (v,w)\text{-paths in } G \text{ are cut}} D(v,w)$.

The *sparsity of moving cut C with respect to a demand D*, for a (hs)-length moving cut C and a h-length demand D, denoted by

$$\text{spars}_{(h,s)}(C, D) = \frac{|C|}{\text{sep}_{hs}(C, D)}$$

is the ratio between the size of the cut C and the total demand of D separated by C. We say that C is *ϕ-sparse with respect to D* is $\text{spars}(C,D) < \phi$.

We are ready to define the key notion of length-constrained expanders.

Definition 3.2 (Cut Characterization of Length-Constrainted Expanders)**.** *The* (h,s)-length conductance *of a graph G is*

$$\text{cond}_{(h,s)}(G) = \min_{D:h\text{-length, degree-respecting}} \min_{C:(hs)\text{-moving cut}} \text{spars}_{(h,s)}(C,D).$$

If $\text{cond}_{(h,s)}(G) \geq \phi$*, we say that G is a* (h,s)-length ϕ-expander*. More generally, for any node-weighting A, the* (h,s)-length conductance *of A is*

$$\text{cond}_{(h,s)}(A) = \min_{D:h\text{-length } A\text{-respecting}} \min_{C:(hs)\text{-moving cut}} \text{spars}_{(h,s)}(C,D).$$

If $\text{cond}_{(h,s)}(A) \geq \phi$*, then we say that A is* (h,s)-length ϕ-expanding in G and that G is a (h,s)-length ϕ-expander for A.

Note that $\text{cond}_{(h,s)}(G) = \text{cond}_{(h,s)}(\deg_G)$. In words, if G is an (h,s)-length ϕ-expander, then there is no ϕ-sparse (hs)-moving cut with respect to any h-length degree-respecting demand. The following observation draws a connection between length-constrained expanders and normal expanders.

Proposition 3.3. *When $h \to \infty$, G is an (h,s)-length ϕ-expander if and only every connected component of G is a ϕ-expander.*

The theorem below shows that, similar to normal expanders, there is a flow characterization of length-constrained expanders that is almost equivalent to its cut characterization within a logarithmic factor.

Theorem 3.4 (Flow Characterization of Length-Constrainted Expanders (Lemma 3.16 of [HRG22]))**.** *We have the following:*

1. *If A is (h,s)-length ϕ-expanding in G, then every h-length A-respecting demand can be routed in G with congestion at most $O(\log(N)/\phi)$ and length at most $s \cdot h$.*
2. *If A is not (h,s)-length ϕ-expanding in G, then some h-length A-respecting demand cannot be routed in G with congestion at most $1/2\phi$ and length at most $\frac{s}{2} \cdot h$.*

3.2 Length-Constrained Expander Decomposition

Expander decomposition are a powerful tool that allows algorithm designers to exploit the power of expanders in an arbitrary graph. The hop-constrained version of them is stated below.

Definition 3.5 (Expander Decomposition)**.** *A* (h,s)-length (ϕ,κ)-expander decomposition *for a node-weighting A in a graph G is a (hs)-moving cut C of size at most $\kappa\phi|A|$ such that A is (h,s)-length ϕ-expanding in $G - C$. C is also called a (h,s,ϕ,κ)-decomposition for A, for short.*

We refer to the parameters s and κ as the *length slack* and *congestion slack*, respectively.

For any moving cut C, the *degree with respect to C* of a vertex v is defined as

$$\deg_C(v) = \sum_{e \text{ incident to } v} u(e) \cdot C(e).$$

By the definition, observe that $\deg_C(v) \leq \deg_G(v)$ for any moving cut C. As discovered in [GRST21], expander decompositions become even more versatile when the vertices incident to the cut-edges of the decomposition are more "well-linked". The definition of a *linked expander decomposition* is given below.

Definition 3.6 (Linked Expander Decomposition)**.** *A* β-linked (h,s)-length (ϕ,κ)-expander decomposition *for a node-weighting A in a graph G is a (hs)-moving cut C of size at most $\kappa\phi|A|$ such that $A + \beta \cdot \deg_C$ is (h,s)-length ϕ-expanding in $G - C$.*

Notice that an expander decomposition is simply a β-linked expander decomposition for $\beta = 0$. Requiring $A + \beta \cdot \deg_C$ instead of just A to be expanding captures the intuition of requiring vertices incident to C to be more "well-linked".

Theorem 3.7 (implicit in [HRG22], explicit in Theorem 3 of [HHT23])**.** *For any node-weighting A, length bound h, length slack $s \geq 100$, conductance bound $\phi > 0$, congestion slack $\kappa \geq N^{O(1/s)} \log N$, and linkedness $\beta = O(1/(\phi\kappa))$, there exists a β-linked (h,s)-length (ϕ,κ)-expander decomposition for A.*

3.3 Routers

Next, we define the notion of *routers*.

Definition 3.8 (Routers)**.** *For any node-weighting A, we say that a* unit-length *capacitated graph G is a t-*step κ-router for A *(or simply a* router*) if every A-respecting demand can be routed in G with congestion κ and t steps. If G is a t-step κ-router for $\deg_G$, then we simply say that G is t-step κ-router.*

By the flow characterization of length-constrained expanders (Theorem 3.4), observe that routers are essentially the same object as length-constrained expanders when the graph has unit length and bounded diameter.

Proposition 3.9. *Let G be a graph with unit edge-length. Let A be a node-weighting where $\mathrm{diam}_G(\mathrm{supp}(A)) \leq h$.*

1. *If G is a (h, s)-length ϕ-expander for A, then G is a (hs)-step $O(\frac{\log N}{\phi})$-router for A.*
2. *If G is not a (h, s)-length ϕ-expander for A, then G is not a $(\frac{hs}{2})$-step $\frac{1}{2\phi}$-router for A.*

Proof. Since $\mathrm{diam}_G(\mathrm{supp}(A)) \leq h$, the set of A-respecting demands and the set of h-length A-respecting demands are identical. Also, as G is unit-length, every path has length equal to the path's step-length. Therefore, by Theorem 3.4, if G is a (h, s)-length ϕ-expander for A, every A-respecting demand can be routed in G with congestion $O(\log(N)/\phi)$ and sh steps. Otherwise, some A-respecting demand cannot be routed in G with congestion $1/2\phi$ and $(sh)/2$ steps. □

Although routers and length-constrained expanders are very similar, they focus on different things. We bound the maximum step-length of flow on routers, and bound the length of flows on length-constrained expanders. The clear distinction between the two is made to avoid confusion.

A simple example of a router is a star.

Proposition 3.10. *For any node-weighting A, let H be a star rooted at $r \notin \mathrm{supp}(A)$ with leaf set $\mathrm{supp}(A)$, each star-edge $(r, v) \in E(H)$ having capacity $A(v)$. Then, H is a 2-step 1-router for A.*

While the above router has great quality, it requires an additional vertex $r \notin \mathrm{supp}(A)$.

A strong router without steiner vertices (one for which $V(H) \subseteq \mathrm{supp}(A)$) can be constructed using constant-degree expanders. The parameter in the construction below can likely be improved, but we choose to present a simple construction.

Lemma 3.11. *For any node-weighting A and positive integer parameter t, there exists a t-step 1-router $H = \mathrm{router}(A, t)$ for A such that*

- $\deg_H \leq \Delta \cdot A$
- $|E(H)| \leq \Delta \cdot |\mathrm{supp}(A)|$

where $\Delta = tn^{O(1/t)} \log^2 N$.

Proof. First, suppose that the node-weighting A is uniform, i.e., $A(v) = 1$ for all v. Let H_0 be a $O(1)$-degree expander of constant conductance on the vertex set $|\mathrm{supp}(A)|$. It is well-known that such a H_0 is a $t' := O(\log n)$-step $O(\log n)$-router. Moreover, for $k = \left\lceil \frac{t'}{t} \right\rceil$, the power graph H_0^k is a t-step $O(t)$-router for A. Each vertex in H_0^k has degree $\Delta_0 = O(1)^k = n^{O(1/t)}$. Define $H = H_0^k$. We have that $\deg_H \leq \Delta_0 \cdot A$ and $|E(H)| \leq \Delta_0 \cdot |\mathrm{supp}(A)|$ as desired. If $A(v) = c_0$ for all v for some c_0, then H can be constructed in the same way but we scale up the capacity by c_0.

Now, we handle the general node-weighting A. By paying at most a factor of 2 in the congestion, we assume that $A(v) = 2^i$ for some $i \in [\log N]$ for every v. Let $V_i = \{v \in \mathrm{supp}(A) \mid A(v) = 2^i\}$ and $A_i = A \cap V_i$ be the restriction of A to V_i. Let H_i be the t-step $O(t)$-router for A_i. The final router H is contained by connecting these H_i together. For each i, j where $i > j$, we construct a bipartite graph $H_{i,j}$ where $V(H_{i,j}) = V_i \cup V_j$ such that every edge of $H_{i,j}$ has capacity 2^j, $\deg_{H_{i,j}}(v) \leq 2^i$ for all $v \in V_i$, and $\deg_{H_{i,j}}(v) \leq 2^j$ for all $v \in V_j$. If $2^i|V_i| \leq 2^j|V_j|$, then $\deg_{H_{i,j}}(v) = 2^i$ for all $v \in V_i$, otherwise $\deg_{H_{i,j}}(v) = 2^j$ for all $v \in V_j$. This can be done be greedily adding edges of capacity 2^j between V_i and V_j in a natural way. We have $|E(H_{i,j})| \leq \max\{|V_i|, |V_j|\}$. We define the final router as $H = (\bigcup_i H_i) \cup (\bigcup_{i,j} H_{i,j})$. Observe that

$$|E(H)| \leq \sum_i |E(H_i)| + \sum_{i,j} |E(H_{i,j})| \leq \Delta_0 |\mathrm{supp}(A)| + \log N |\mathrm{supp}(A)|.$$

Similarly, $\deg_H \leq (\Delta_0 + \log N) \cdot A$. We claim that H is a $2t$-step $O(t \log N)$-router. To see the claim, given any demand D respecting A, we first route the demand from V_i to V_j through $H_{i,j}$ with congestion 1 for all i, j. The residual demand will respects $\log N \cdot A$ and only need to route inside each H_i, which can be routed using $O(t \log N)$ congestion.

Finally, to reduce the congestion to 1, we simply make $O(t \log N)$ parallel copies of each edge. Scaling the parameters by an appropriate constant, this implies the lemma. □

3.4 Length-Constrained Expansion Witness

We first recall two more standard notions.

Neighborhood Covers. Given a graph G with lengths, a *clustering* $\mathcal{S}$ in G is a collection of mutually disjoint vertex sets $S_1, \ldots, S_{|\mathcal{S}|}$, called *clusters*. A *neighborhood cover* $\mathcal{N}$ with width ω and covering radius h is a collection of ω many clusterings $\mathcal{S}_1, \ldots, \mathcal{S}_\omega$ such that for every node v there exists a cluster $S \in \mathcal{S}_i$ where $\mathrm{ball}(v, h) \subseteq S$. We use $S \in \mathcal{N}$ to say that S is a cluster in some clustering of $\mathcal{N}$. We say that $\mathcal{N}$ has diameter h_{diam} if every cluster $S \in \mathcal{N}$ has (weak) diameter at most h_{diam}, i.e., $\max_{u,v \in S} \mathrm{dist}_G(u, v) \leq h_{\mathrm{diam}}$. The following is a classic result.

Theorem 3.12 ([Pel00])**.** *For any h, integer $k \geq 1$, and graph G, there a deterministic parallel algorithm that computes a neighborhood cover $\mathcal{N}$ with covering radius h, diameter $h_{\mathrm{diam}} \leq (2k-1) \cdot h$ and width $\omega = N^{O(1/k)} \log N$. The algorithm has $O(|E(G)|hk\omega)$ work and $O(hk\omega)$ depth.*

Embedding. Next, we recall the notion of *graph embedding.* We view it as a flow as follows.

Definition 3.13 (Edge Demand and Embedding)**.** *Given graphs G and H where $V(H) \subseteq V(G)$, the* edge-demand $D_{E(H)}$ *of H on G is the demand where for all $(v, w) \in E(H)$, $D_{E(H)}(v, w) = u_H(v, w)$.*

The embedding $\Pi_{H\to G}$ of H into G *is a multicommodity flow that routes* $D_{E(H)}$ *in* G. *We can write* $\Pi_{H\to G} = \sum_{(v,w)\in E(H)} f_{(v,w)}$ *where* $f_{(v,w)}$ *is a* (v,w)*-flow in* G *of value* $\mathrm{val}(f_{(v,w)}) = u_H(v,w)$.

The embedding $\Pi_{H\to G}$ *is said to have* length slack s *if* $\mathrm{leng}_{f_e} \le s \cdot \ell_H(e)$ *for all* $e \in E(H)$.

We use the same terminology as for flows for the embedding $\Pi_{H\to G}$. For example, $\Pi_{H\to G}$ is said to have length h and congestion κ if $\mathrm{leng}_{\Pi_{H\to G}} \le h$ and $\mathrm{cong}_{\Pi_{H\to G}} \le \kappa$.

Expansion Witness. Given the above definitions of neighborhood covers, routers, and embeddings, we can define a *witness* of length-constrained expansion.

Definition 3.14 (Expansion Witness)**.** *Let* G *be a graph and* A *a node weighting. A* $(h, t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$-witness *of* A *in* G *is a tuple* $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$.

- $\mathcal{N}$ *is a neighborhood cover with covering radius* h.
- $\mathcal{R}$ *is a collection of routers. For each cluster* $S \in \mathcal{N}$*, there exists a* $(t_{\mathcal{R}}, 1)$*-router* $R^S \in \mathcal{R}$ *on vertex set* S *for node-weighting* $S \cap A$.
- $\Pi_{\mathcal{R}\to G}$ *is an embedding of all routers in* $\mathcal{R}$ *to* G. $\Pi_{\mathcal{R}\to G}$ *has length* h_Π *and congestion* κ_Π.

Below, we show that (1) an expansion witness indeed certifies the expansion, and moreover, gives an explicit routing structure, and (2) an expansion witness exists for every expander.

Corollary 3.15 (Expansion Witness Certifies Expansion)**.** *Suppose that there exists a* $(h, t_{\mathcal{R}}, t_\Pi, \kappa_\Pi)$-witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$ *of* A *in* G. *Then,* G *is a* $(h, 2t_{\mathcal{R}}t_\Pi)$*-length* $(1/2\kappa_\Pi)$*-expander for* A. *Moreover, given any* A*-respecting* h*-length-constrained demand* D *in* G*, there is a flow* F *routing* D *in* G *where* $\mathrm{leng}_F \le h_\Pi t_{\mathcal{R}}$ *and* $\mathrm{cong}_F \le \kappa_\Pi$. *Moreover,* F *is "routed through" the embedding* $\Pi_{\mathcal{R}\to G}$*, i.e.,* $F = \sum_{f\in\Pi_{\mathcal{R}\to G}} \mathrm{val}_f \cdot f$ *where* $\mathrm{val}_f \ge 0$.

Proof. It suffices to prove the "moreover" part by Theorem 3.4. For each (v,w) where $D(v,w) > 0$, we have $\mathrm{dist}_{G-C}(v,w) \le h$ and so there exists $S \in \mathcal{N}$ where $v, w \in S$. Choose such cluster S arbitrarily and assign the demand $D(v,w)$ to S. For each cluster, let D_S denote the demand induced by this process. Note that $D_S \le D$ entry-wise and so D_S respects $S \cap A$. So D_S can be routed via a flow F_S in R^S with $t_{\mathcal{R}}$-step and 1-congestion. Let $F_{\mathcal{R}} = \sum_{S\in\mathcal{N}} F_S$ be a flow on $\mathcal{R} = \cup_{S\in\mathcal{N}} R^S$.

We define F from $F_{\mathcal{R}}$ as follows. From the embedding $\Pi_{\mathcal{R}\to G} = \sum_{e\in E(\mathcal{R})} u_{\mathcal{R}}(e) \cdot f_e$ that embeds $\mathcal{R}$ into G, define the flow $F = \sum_{e\in E(\mathcal{R})} v_{f_e} \cdot f_e$ where $f_e \in \Pi_{\mathcal{R}\to G}$ and $v_{f_e} = F_{\mathcal{R}}(e)$ denotes the total flow of $F_{\mathcal{R}}$ through e in $\mathcal{R}$. Now, we bound the length and congestion of F. Since $F_{\mathcal{R}}$ has at most $t_{\mathcal{R}}$-step on $\mathcal{R}$, we have $\mathrm{leng}_F \le t_{\mathcal{R}} \cdot \mathrm{leng}_{\Pi_{\mathcal{R}\to G}} \le h_\Pi t_{\mathcal{R}}$. Since $F_{\mathcal{R}}$ has congestion 1 on $\mathcal{R}$, we have $v_{f_e} \le u_{\mathcal{R}}(e)$ for all $e \in E(\mathcal{R})$ and so $\mathrm{cong}_F \le \mathrm{cong}_{\Pi_{\mathcal{R}\to G}} = \kappa_\Pi$. □

Corollary 3.16 (Expansion Witness Exists for Expanders)**.** *Let* G *be a* (h, s)*-length* ϕ*-expander for a node-weighting* A. *There is an* $(h', t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$*-witness* $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$ *for* A *in* G *where* $h' = h/s$*,* $t_{\mathcal{R}} = s$*,* $h_\Pi = hs$*,* $\kappa_\Pi = sN^{O(1/s)}\mathrm{poly}(\log N)/\phi$.

Proof. From Theorem 3.12, let $\mathcal{N}$ be a neighborhood cover on G with covering radius $h' = h/s$, diameter h, and width $\omega \le N^{O(1/s)} \log N$. For each cluster $S \in \mathcal{N}$, let $R^S \in \mathcal{R}$ be a s-step 1-router

for $B_S = S \cap A$ where $\deg_{R^S} \leq \Delta \cdot B_S$ and $|E(R^S)| \leq \Delta \cdot |\mathrm{supp}(B_S)|$ and $\Delta \leq sN^{O(1/s)} \log^2 N$. This follows from Lemma 3.11.

Let $D_{\mathcal{R}}$ be the edge demand of $\cup_{S \in \mathcal{N}} R^S$. First, observe that $D_{\mathcal{R}}$ is h-length-constrained in G. Indeed, for each edge $(v, w) \in R^S$ for any $S \in \mathcal{N}$, we have $\mathrm{dist}_G(v, w) \leq h$ as $\mathcal{N}$ has diameter h. Second, observe that $D_{\mathcal{R}}$ respects $\Delta\omega A$. Indeed, for each vertex v, we have

$$D_{\mathcal{R}}(v) \leq \sum_{S \in \mathcal{N}} \deg_{R^S}(v) \leq \Delta \cdot \sum_{S \in \mathcal{N}} B_S(v) \leq \Delta\omega \cdot A(v)$$

where the last inequality is because there are most ω many clusters $S \in \mathcal{N}$ containing v.

Since A is (h, s)-length ϕ-expanding in G, by Theorem 3.4, we have that $D_{\mathcal{R}}$ can be routed in G with length $hs = h_\Pi$ and congestion $\kappa_\Pi = O(\Delta\omega \log N/\phi)$. Let $\Pi_{\mathcal{R} \to G}$ be such flow that routes $D_{\mathcal{R}}$ in G. □

3.5 Length-Constrained Witnessed Expander Decomposition

By combining the existence of the expander decomposition and the expansion witness, we obtain the following.

Corollary 3.17 (Existential Witnessed Expander Decomposition)**.** *Let G be a graph with edge lengths with node-weighting A. Given parameters (h, ϕ, β, s) where $\beta \leq 1/(\phi \log N)$ and $s \geq 100$, there exists a h_C-moving cut C and a $(h, t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$-witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R} \to G})$ of $A + \beta\deg_C$ in $G - C$ with the following guarantees:*

- $|C| \leq \phi|A|$.
- *The total number of edges in all routers is $|E(\mathcal{R})| \leq nN^{O(1/s)}\mathrm{poly}(\log N)$.*
- $t_{\mathcal{R}} = s$, $h_C, h_\Pi = hs^2$, $\kappa_\Pi = N^{O(1/s)}\mathrm{poly}(\log N)/\phi$.

Proof. From Theorem 3.7, there exists a hs^2-moving cut C of size at most $\phi|A|$ such that $A + \beta\deg_C$ is (hs, s)-length (ϕ/κ)-expanding in $G - C$ where $\kappa = N^{O(1/s)} \log N$. By Corollary 3.16, there is a $(h', t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$-witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R} \to G})$ for $A + \beta\deg_C$ in $G - C$ where $h' = hs/s$, $t_{\mathcal{R}} = s$, $s_\Pi = hs^2$, $\kappa_\Pi = \frac{\kappa}{\phi} sN^{O(1/s)}\mathrm{poly}\log N$. □

The key subroutine that this paper relies on as a blackbox is an efficient parallel algorithm for computing a length-constrained expander decomposition C for A, together with the expansion witness for A in $G - C$.

Theorem 3.18 (Algorithmic Witnessed Expander Decomposition from Theorem 1.1 of [HHT24])**.** *Let G be a graph with edge lengths with node-weighting A. Given parameters (h, ϕ, β, s) where $\beta \leq 1/(\phi \log N)$ and $s \leq \log^c N$ for some sufficiently small constant c, let $\epsilon = 1/s$. There exists an algorithm that computes an h_C-moving cut C and a $(h, t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$-witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R} \to G})$ of $A + \beta\deg_C$ in $G - C$ with the following guarantees:*

- $|C| \leq \phi|A|$.
- *The total number of edges in all routers is $|E(\mathcal{R})| \leq |E(G)|N^{\mathrm{poly}\epsilon}$. Moreover, $\Pi_{\mathcal{R} \to G}$ is an integral embedding with path count at most $|E(G)|N^{\mathrm{poly}\epsilon}$.*

- $t_{\mathcal{R}} = s$, $h_C, h_\Pi \leq h \cdot \exp(\text{poly}(1/\epsilon))$, *and* $\kappa_\Pi = N^{\text{poly}\epsilon}/\phi$.

The algorithm takes $|E(G)|\text{poly}(h)N^{\text{poly}\epsilon}$ *work and has depth* $\text{poly}(h)N^{\text{poly}\epsilon}$.

4 Length-Constrained Low-Step Emulators: Existence

The goal of this section is to show that, given any graph G, there is another graph G' such that any length-constrained multi-commodity flow in G can be routed in G' with approximately the same congestion *and* length, and moreover such flow in G' can be routed via paths containing few edges. The definition below formalize this idea.

Definition 4.1 (Length-Constrained Low-Step Emulators)**.** *Given a graph G and a node-weighting A in G, we say that G' is an* h-length-constrained t-step emulator *of A with length slack s and congestion slack κ if*

1. *Edges in G' have uniform length of $h' = sh$.*
2. *Given a flow F in G routing an A-respecting demand and $\text{leng}_F \leq h$, there is a flow F' in G' routing the same demand where $\text{cong}_{F'} \leq \kappa \cdot \text{cong}_F$ and $\text{step}_{F'} \leq t$ (equivalently, $\text{leng}_{F'} \leq t \cdot sh$ by Item 1).*
3. *G' can be embedded into G with congestion 1 and length slack 1.*

The condition requiring that edges in G' have uniform length is crucial. This will allow us to bootstrap the construction for h-length-constrained t-step emulator for large h using based on the ones for small h. Our main technical contribution of this section is showing the existence of length-constrained low-step emulators.

Theorem 4.2 (Existential)**.** *Given any graph G with n vertices, a node-weighting A of G, and parameters h and t, there exists an h-length-constrained $O(t^2)$-step emulator G' for A in G with length slack $O(t^2)$ and congestion slack $\text{poly}(t \log N)N^{O(1/t)}$. The emulator contains $nN^{O(1/t)}\text{poly}(t \log N)$ edges.*

We also give a parallel algorithm for constructing an emulator with a worse trade-off.

Theorem 4.3 (Algorithmic)**.** *There exists a parallel algorithm that, given any graph G with m edges, a node-weighting A of G, and parameters h and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, computes an h-length-constrained $O(t^2)$-step emulator G' for A in G where $t = \exp(\text{poly}1/\epsilon)$ with length slack $O(t^2)$ and congestion slack $N^{\text{poly}\epsilon}$. The emulator contains $mN^{\text{poly}\epsilon}$ edges and the embedding $\Pi_{G' \to G}$ has path count $mN^{\text{poly}\epsilon}$.*

The algorithm has $m \cdot \text{poly}(h)N^{\text{poly}\epsilon}$ work and $\text{poly}(h)N^{\text{poly}\epsilon}$ depth.

4.1 Construction and Analysis

In this section, we show a construction of h-length-constrained t-step emulator.

Let us start with basic technical observations on Algorithm 1.

Algorithm 1 EMULATOR(G, t, h)

Initialize $A_0 \leftarrow A$, $h_{\mathrm{cov}} \leftarrow 4h$, $s_{\mathrm{dcp}} \leftarrow t$, $\beta \leftarrow t^2$, and $\phi \leftarrow 1/2\beta N^{1/t}$. (The choice of β is so that $\beta = h_C/h$).

1. For $1 \le i \le 2t$:
   (a) Given node-weighing A_{i-1}, compute a witnessed expander decomposition C_i and $(\mathcal{N}_i, \mathcal{R}_i, \Pi_{\mathcal{R}_i \to G})$ with parameters $(h_{\mathrm{cov}}, \phi, \beta, s_{\mathrm{dcp}})$ using Corollary 3.17.
      i. C_i is a h_C-moving cut, and
      ii. $(\mathcal{N}_i, \mathcal{R}_i, \Pi_{\mathcal{R}_i \to G})$ is a $(h_{\mathrm{cov}}, t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$-witness of $A_{i-1} + \beta \deg_{C_i}$ in $G - C_i$.
   (b) Set $B_i \leftarrow A_{i-1} + \beta \deg_{C_i}$ and $A_i \leftarrow \beta \deg_{C_i}$.
   (c) Set $E'_i \leftarrow E(\mathcal{R}_i) := \bigcup_{S \in \mathcal{N}_i} R^S$ scaled the capacity down by $t\kappa_\Pi$.
2. Return G' where $E(G') = \cup_i E'_i$ and $\ell_{G'}(e) = h_\Pi$ for all $e \in E(G')$.

Proposition 4.4. *We have the following:*

1. *For all i and v, $A_i(v)$ and $B_i(v)$ are non-negative multiples of $\frac{1}{h}$.*
2. *For all $i \ge 0$, we have $|A_i| \le |A|/N^{i/t}$. In particular, $|A_{2t}| = 0$.*
3. *$|E(G')| \le 2t \cdot \mathrm{size}_{\mathcal{R}}$ where $\mathrm{size}_{\mathcal{R}}$ upper bounds the total number of edges in the routers.*

Proof. (1): We assume that A_0 is integral. For $i \ge 1$, $C_i(e)$ is a non-negative multiple of $\frac{1}{h_C}$ for all e as C_i is a h_C-moving cut. By induction, $A_i(v)$ and $B_i(v)$ are non-negative multiples of $\frac{\beta}{h_C} = \frac{1}{h}$.

(2): For $i = 0$, this holds by the assumption. For $i \ge 1$, we have that $|C_i| \le \phi |A_{i-1}|$ by Corollary 3.17 and so

$$|A_i| \le \beta |\deg_{C_i}| = 2\beta |C_i| \le 2\beta\phi |A_{i-1}| = |A_{i-1}|/N^{1/t} \le |A|/N^{i/t}.$$

by the choice of ϕ and by induction hypothesis. Since we have $|A_{2t}| \le N^{1-2t/t} < \frac{1}{h}$, it follows that $|A_{2t}| = 0$ by (1).

(3): This is because there are at most $2t$ levels. □

We show that any demand in G can be routed in G' with small congestion, length, and steps. This is the key technical lemma and we defer the proof to Section 4.2.

Lemma 4.5 (Forward Mapping)**.** *Let F be a flow in G where D_F respects A such that $\mathrm{cong}_F = 1$ and $\mathrm{leng}_F \le h$. There is a flow F' routing D_F in G' with $\mathrm{cong}_{F'} \le t\kappa_\Pi = N^{O(1/t)}\mathrm{poly}(t \log N)$ and $\mathrm{step}_{F'} \le O(t \cdot t_{\mathcal{R}}) = O(t^2)$.*

Next, we show an embedding from G' into G.

Lemma 4.6 (Backward Mapping)**.** *There exists an embedding $\Pi_{G' \to G}$ from G' into G with length slack 1 and congestion 1.*

Proof. For each level i, there is an embedding $\Pi_{\mathcal{R}_i \to G}$ from $\mathcal{R}_i$ into G with congestion κ_Π and length h_Π. Recall that $E(G') = \cup_{i=1}^{2t} E(\mathcal{R}_i)$ where the capacity is scaled down by $t\kappa_\Pi$. Define the embedding $\Pi_{G' \to G} = \sum_i \Pi_{\mathcal{R}_i \to G}/(\kappa_\Pi t)$ scaled down by $t\kappa_\Pi$.

The length slack of $\Pi_{G' \to G}$ is 1 because the length of $\Pi_{\mathcal{R}_i \to G}$ is h_Π for every i but we have $\ell_{G'}(e) = h_\Pi$ for all $e \in E(G')$. The congestion of $\Pi_{G' \to G}$ is 1 because the congestion of $\sum_i \Pi_{\mathcal{R}_i \to G}$ is at most $\kappa_\Pi t$ but we scaled down by the flow by $\kappa_\Pi t$. □

Now, we are ready to prove Theorem 4.2.

Proof of Theorem 4.2.

Proof. Let $G' = \textsc{Emulator}(G, t, h)$ be the output of Algorithm 1. By Lemma 4.5 and Lemma 4.6 immediately implies that G' is an h-length-constrained $O(t^2)$-step emulator for G with congestion $t\kappa_\Pi = \text{poly}(t \log N) N^{O(1/t)}/\phi = \text{poly}(t \log N) N^{O(1/t)}$ because $\phi^{-1} = 2\beta N^{1/t} = \tilde{O}(t^2 N^{O(1/t)})$. The length slack is $s = t^2$ because edges in G' have length $h_\Pi = t^2 h$ by construction. The bound on $|E(G')|$ follows Proposition 4.4 (Item 3 and Corollary 3.17. □

4.2 Proof of Lemma 4.5: Forward Mapping

Strategy. Our strategy is to construct the flow F' that routes D_F in G' incrementally. More concretely, let $D_0 = D_F$. For each $i \geq 1$, we will construct a flow F'_i that partially routes D_{i-1} in G' using only edges from E'_i so that the remaining demand is D_i. After $i > 2t$, we have $D_i = 0$, i.e., there is no remaining demand. By combining and concatenating these flows F'_i for all $i \geq 1$, we will obtain F' routing D_F in G' with the desired properties.

We will maintain the following invariant, for all $i \geq 0$,

1. D_i is A_i-respecting, and
2. D_i is routable in G with congestion 1 and length h.

Let us check that the invariant holds for $i = 0$. First, D_0 respects A_0 because D_F respects A by assumption. Second, D_0 is routable in G with congestion 1 and length h because $\text{cong}_F = 1$ and $\text{leng}_F \leq h$ by assumption. For $i \geq 1$, assuming that the invariant holds for $i-1$, we will construct the flow F'_i that partially routes D_{i-1} so that the invariant holds for i. We will then argue why the invariant for all i implies that our final flow F' has the desired properties.

Construct F'_i. The high-level idea is that we try to send a packet from v to w for each demand pair (v, w) of D_{i-1}. If $\text{dist}_{G - C_i}(v, w) \leq h_{\text{cov}}$, then we will successfully route this packet via some router and be done with it. Otherwise, $\text{dist}_{G - C_i}(v, w) > h_{\text{cov}}$. In this case, for each (v, w)-flow path P, we will carefully identify a set of vertices $X_{v,P}$ and fractionally route the packet from v to $X_{v,P}$. Similarly, from another end, the packet w is routed to a set $X_{w,P}$ that we carefully define. We think of v "forward" the packet to $X_{v,P}$ and w "forward" the packet to $X_{w,P}$. The demand between $X_{v,P}$ and $X_{w,P}$ will induce the demand D_i in the next level. Below, we explain this high-level idea in detail.

For each demand pair (v, w) of D_{i-1} where $\text{dist}_{G - C_i}(v, w) \leq h_{\text{cov}}$, there must exist a cluster $S \in \text{cluster}(\mathcal{N}_i)$ where $v, w \in S$. We *assign* the pair (v, w) to such arbitrary cluster S. For each

cluster $S \in \text{cluster}(\mathcal{N}_i)$, we route in G' all demands $D_{i-1}(v,w)$ for all pairs (v,w) assigned to S using the edges of router R^S for $B_{i,S}$. Let F_S^{done} denote the flow in G' between vertices in S induced by the above routing. Let D_S^{done} be the demand routed by F_S^{done}.

Now, we take care of the remaining demand pair (v,w) of D_{i-1} where $\text{dist}_{G-C_i}(v,w) > h_{\text{cov}} = 4h$. Let F_{i-1} be a flow routing D_{i-1} in G with congestion 1 and length h, whose existence is guaranteed by the invariant. Consider any (v,w)-flow-path P of F_{i-1}. Since $\text{dist}_{G-C_i}(v,w) > 4h$ but $\ell_G(P) \le \text{leng}_{F_{i-1}} \le h$, C_i must increase the length P by least $3h$.

Let $E_{v,P}$ the minimal edge set in P closest to v such that the total length increase by C_i is at least h, i.e., $\sum_{e \in E_{v,P}} C_i(e) h_C \ge h$. Next, we define the vertex set $X_{v,P} \subseteq P$ as follows. For each $(x,y) \in E_{v,P}$ where x is closer to v, if $C_i(x,y) > 0$, we include x into set $X_{v,P}$. Let us denote $e_x = (x,y) \in E_{v,P}$ as the edge corresponding to $x \in X_{v,P}$. Observe that, for any $x \in X_{v,P}$,

$$\text{dist}_{G-C_i}(v,x) \le \ell_G(P) + h \le 2h \le h_{\text{cov}}$$

That is, $X_{v,P} \subseteq \text{ball}_{G-C_i}(v, h_{\text{cov}})$ and so there exists a cluster $S \in \text{cluster}(\mathcal{N}_i)$ containing both v and $X_{v,P}$. For each $x \in X_{v,P}$, we route flow in G' from v to x of value at most

$$F_{i-1}(P) \cdot \frac{C_i(e_x) h_C}{h}$$

via router R^S. Note that $v, x \in R^S$ which is a router for $S \cap (A_{i-1} + \beta \deg_{C_i})$ because $A_{i-1}(v) > 0$ and $\deg_{C_i}(x) > 0$. Since $\sum_{x \in X_{v,P}} C_i(e_x) h_C \ge h$ by definition, we can route flow from v to $X_{v,P}$ of total value $F_{i-1}(P)$. Symmetrically, we define $E_{w,P}$ and $X_{w,P}$. Observe that $X_{v,P}$ and $X_{w,P}$ are disjoint because they are defined based on vertices closest to v and w, respectively. We route flow in G' from w to $X_{w,P}$ of total value $F_{i-1}(P)$ via R^S where $S \in \text{cluster}(\mathcal{N}_i)$ is a cluster containing both w and $X_{w,P}$

For each cluster $S \in \text{cluster}(\mathcal{N}_i)$, let F_S^{forward} be the flow in G' induced by the routing described above, which routes from vertices positive demand in D_{i-1} to the vertices incident to the cut C_i. Let D_S^{forward} be the demand routed by F_S^{forward}.

Finally, we define the flow $F_i' = \sum_{S \in \text{cluster}(\mathcal{N}_i)} F_S^{\text{done}} + F_S^{\text{forward}}$ in G' that routes D_S^{done} and D_S^{forward} for all $S \in \text{cluster}(\mathcal{N}_i)$ in the manner described above.

D_S^{done} and D_S^{forward} are $B_{i,S}$-respecting. Here, we argue that both D_S^{done} and D_S^{forward} are $B_{i,S}$-respecting. This is useful because, by Lemma 3.11, it means that both D_S^{done} and D_S^{forward} can be routed in $R^S \subseteq G'$ using $t_{\mathcal{R}}$ steps and congestion 1.

It suffices to show that D_S^{done} and D_S^{forward} are B_i-respecting because their supports are only on the pairs of vertices inside S. This is easy to argue for D_S^{done}. We have $D_S^{\text{done}} \le D_{i-1}$ (entry-wise), D_{i-1} is A_{i-1}-respecting, and $A_{i-1} \le B_i$ (entry-wise). So D_S^{done} is B_i-respecting. Next, we analyze D_S^{forward}. On one hand, the total demand that each vertex v may send out is $\sum_x D_S^{\text{forward}}(v,x) \le \sum_x D_{i-1}(v,x) \le A_{i-1}(v)$ because D_{i-1} is A_{i-1}-respecting by the invariant. On the other hand, we claim that the total demand that each vertex x may receive is $\sum_v D_S^{\text{forward}}(v,x) \le \beta \deg_{C_i}(x)$. Since $B_i = A_{i-1} + \beta \deg_{C_i}$, we also have D_S^{forward} is B_i-respecting.

Now, we prove the claim. The key observation is $\sum_v D_S^{\text{forward}}(v,x)$ is at most

$$\sum_{e:\text{incident to } x} F_{i-1}(e) \frac{C_i(e) h_C}{h}.$$

This is because, by construction of the flow F_S^{forward}, x may only receive the demand of quantity at most $F_{i-1}(P)\frac{C_i(e)h_C}{h}$ whenever $x \in P$ and $e \in P$ is incident to x. But we have

$$\sum_{e:\text{incident to } x} F_{i-1}(e)C_i(e)\frac{h_C}{h} \leq \sum_{e:\text{incident to } x} u_G(e)C_i(e)\beta = \beta \deg_{C_i}(x)$$

because F_{i-1} has congestion 1 in G and so $F_{i-1}(e) \leq u_G(e)$. Also, $\beta = \frac{h_C}{h}$ by definition. This finishes the claim.

Define D_i and Prove the Invariant. We define D_i as the remaining demand after routing F'_i. Observe that the remaining demand is as follows. For each demand pair (v, w) of D_{i-1} where $\text{dist}_{G-C_i}(v, w) > h_{\text{cov}}$ and each (v, w)-flow-path P of F_{i-1}, we need route flow of value from $X_{v,P}$ to $X_{w,P}$ of total value $F_{i-1}(P)$. Then, D_i sums up these demand.

Now, we argue that D_i satisfies the invariant. Consider the congestion and length required for routing D_i in G. Observe that D_i can be routed through subpaths of the flow-paths of F_{i-1} and F_{i-1} is routable with congestion 1 and length h in G. Therefore, D_i is routable in G with the same congestion and length bound.

Next, we show that D_i is A_i-respecting. For $x \in \text{supp}(A_i)$, observe that the total demand that x sends out in D_i is the same as the total demand x receives in $\sum_S D_S^{\text{forward}}$. But we showed that this is at most $\sum_{e:\text{incident to } x} F_{i-1}(e)\frac{C_i(e)h_C}{h} \leq \beta \deg_{C_i}(x) = A_i(x)$ by the definition of A_i. This completes the proof why the invariant holds.

Construct F' and Bound its Quality. The flow F' is obtained by combining and concatenating the flows that route D_S^{done} and D_S^{forward} overall $S \in \text{cluster}(\mathcal{N}_i)$ for all level i in a natural way so that F' routes D_F. That is, in level i, each demand of D_{i-1} is routed either successfully routed in G' via some router in $t_{\mathcal{R}}$ steps, or forwards to a new demand in D_i by routing in G' via some router in $t_{\mathcal{R}}$ steps as well. As $i \leq 2t$, the maximum step of F' is $O(t \cdot t_{\mathcal{R}})$.

Now, we bound the congestion on G'. F' simultaneously routes, for all i, D_S^{done} and D_S^{forward} for each $S \in \text{cluster}(\mathcal{N}_i)$ on R^S. Since D_S^{done} and D_S^{forward} are $B_{i,S}$-respecting, the congestion for routing both D_S^{done} and D_S^{forward} on R^S is at most 1. But each router is edge-disjoint from each other, so the congestion for routing F' on $\cup_i E(\mathcal{R}_i) = \cup_i \cup_{S \in \mathcal{N}_i} R^S$ is at most 1. Since $E(G') = \cup_i E(\mathcal{R}_i)$ scaled down the capacity by $t\kappa_\Pi$, the congestion for routing F' in G' is then $t\kappa_\Pi$.

To summarize, we have successfully constructed a flow F' routing D_F in G' with $\text{cong}_{F'} \leq t\kappa_\Pi$ and $\text{step}_{F'} \leq O(t \cdot t_{\mathcal{R}})$ as desired.

4.3 Reduction to Witnessed Expander Decomposition

Observe that we can state Algorithm 1 as a reduction from length-constrained low-step emulators to witnessed expander decomposition, since the algorithm simply compute witnessed expander decomposition $O(t)$ times. This can be formalized as follows.

Corollary 4.7. *Suppose there is an algorithm $\mathcal{A}$ that, given an arbitrary node-weighting A_{i-1} of graph G, computes a witnessed expander decomposition C_i and $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$ with parameters $(h_{\text{cov}}, \phi, \beta, s_{\text{dcp}})$ such that*

- C_i *is a* h_C*-moving cut*
- $(\mathcal{N}_i, \mathcal{R}_i, \Pi_{\mathcal{R}_i \to G})$ *is a* $(h_{\text{cov}}, t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$*-witness of* $A_{i-1} + \beta \deg_{C_i}$ *in* $G - C_i$.
- $\mathcal{R}_i$ *has total number of edges at most* $\text{size}_{\mathcal{R}}$ *and the embedding* $\Pi_{\mathcal{R}_i \to G}$ *has path count at most* path_Π.
- $h_{\text{cov}} \leftarrow 4h$, $s_{\text{dcp}} \leftarrow t$, $\phi \leftarrow 1/2\beta N^{1/t}$, *and* β *is chosen such that* $\beta = h_C/h$.

Then, there is an algorithm for computing an h*-length-constrained* $O(t \cdot t_{\mathcal{R}})$*-step emulator* G' *for* G *with length slack* $O(t \cdot t_{\mathcal{R}})$ *and congestion slack* $t\kappa_\Pi$. *The emulator contains* $2t \cdot \text{size}_{\mathcal{R}}$ *edges and the embedding* $\Pi_{G' \to G}$ *has path count* $2t \cdot \text{path}_\Pi$.

The algorithm makes $O(t)$ *calls to* $\mathcal{A}$ *and spend additional work of* $O(|E(G')|)$ *and depth* $O(\log n)$.

Theorem 4.2 is then obtained simply by plugging the existential result of witnessed expander decomposition with parameters $(h_{\text{cov}}, \phi, \beta, s_{\text{dcp}})$. From Corollary 3.17, we need to set $\beta = t^2$ so that $\beta = h_C/h$. Thus, we get

$$\begin{aligned} t_{\mathcal{R}} &= t \\ \kappa_\Pi &= \tilde{O}(N^{O(1/t)}/\phi) = N^{O(1/t)}\text{poly}(\log N) \\ \text{size}_{\mathcal{R}} &= n \cdot N^{O(1/t)}\text{poly}(\log N), \end{aligned}$$

which implies Theorem 4.2 by Corollary 4.7.

From this reduction, we also immediately obtain an algorithmic result (Theorem 4.3) by plugging in the algorithmic witnessed expander decomposition from Theorem 3.18 with parameters $(h_{\text{cov}}, \phi, \beta, s_{\text{dcp}})$ into Corollary 4.7. Define $t = 1/\epsilon$. We need to set $\beta = \exp(\text{poly}(1/\epsilon))$ so that $\beta = h_C/h$. Thus, we get

$$\begin{aligned} t_{\mathcal{R}} &= t \\ \kappa_\Pi &= N^{\text{poly}\epsilon}/\phi = N^{\text{poly}\epsilon} \\ \text{path}_\Pi, \text{size}_{\mathcal{R}} &= m \cdot N^{O(1/t)}\text{poly}(\log N), \end{aligned}$$

which implies Theorem 4.3.

5 Bootstrapping Length-Constrained Low-Step Emulators

In this section, we establish efficient algorithms and representations for length-constrained low-step emulators. At a high level, this requires overcoming two barriers, the first technical and the second conceptual.

1. Efficiency: The (h, s)-length (ϕ, κ)-expander decomposition algorithm has a polynomial-in-h dependency in the running time. Even in unit-length graphs, h can be as large as n, which is prohibitively slow. Nevertheless, this is a purely technical issue, as a fast decomposition algorithm for any h can still exist. In fact, we could obtain such an algorithm using the techniques developed in our paper.

2. Representation size: Even if there was a fast (h, s)-length (ϕ, κ)-expander decomposition algorithm for any value of h, the additional embedding $|\Pi_{\mathcal{R}\to G}|$ may have size at least hn, since we need to embed at least a linear number of paths as per Theorem 3.18, and each path of length h can consist of h many edges. Conceptually, this barrier appears unavoidable with explicit embeddings as required by $\Pi_{\mathcal{R}\to G}$. In this section, we bypass this issue by *stacking* emulators on top of each other in a hierarchical fashion. More precisely, each emulator may embed not only into the original graph, but into previously computed emulators. This way, an h-length path can be implicitly represented across multiple levels of stacking: an edge can embed into an emulator at a previous level, whose path in this emulator contains edges that embed further into previous emulators, and so on.

Theorem 5.1 (Bootstrapping)**.** *Let h, d, t be given parameters. Suppose there exists $\alpha > 1$ and an algorithm $\mathcal{A}$ that, given an m-edge n-vertex graph G and $h' \le (2t+3)h$, computes an h'-length-constrained t-step emulator G' of G with length slack s, congestion slack κ, number of edges at most $|E(G')| \le \alpha|E(G)|$, and path count of the embedding $|\text{path}(\Pi_{G'\to G})| \le \alpha|E(G)|$.*

Then, there is an algorithm that, given graph G and (h, d, t, h_0) as parameters where $h_0 \le h$, computes graphs $G'_0, G'_1, G'_2, \ldots, G'_d$ such that for each index i,

1. *G'_i is an h_0h^i-length-constrained t-step emulator of G with length slack $s^{i+1}(2t+3)^i$ and congestion slack $(2\kappa)^{i+1}$.*
2. *G'_i has at most $(2\alpha)^{i+1}m$ edges.*
3. *There is an embedding $\Pi_{G'_i\to G\cup G'_{i-1}}$ that embeds G'_i into $G\cup G'_{i-1}$ with length slack 1, congestion 1, path count $(2\alpha)^{i+1}m$, and maximum step at most $(2st+3s)h$.[2] Moreover, $\Pi_{G'_i\to G\cup G'_{i-1}}$ only routes through edges of G with length in the range $(h_0h^{i-1}, h_0h^i]$.*

The algorithm calls $\mathcal{A}$ on d many graphs, each with at most $O(d(2\alpha)^d m)$ edges, and, outside these calls, runs in $O(d(2\alpha)^d m)$ work and $\tilde{O}(d)$ depth.

Before proving Theorem 5.1, we explain why Item 3 is important; it is crucial in the lemma below.

Lemma 5.2. *For any $i \in [d]$, for any flow F' in G'_i, there is a flow F in G routing the same demand where $\text{cong}_F \le \text{cong}_{F'}$ and $\text{leng}_F \le \text{leng}_{F'}$. Given the edge representation $\text{flow}_{F'}$ of F', we can compute the edge representation flow_F of F in $O(hst\cdot(2\alpha)^{i+1}m)$ work and $\tilde{O}(i)$ depth.*

Proof. By scaling, we can assume that F' has congestion and length 1 in G'_i. We will construct the edge representation flow_F of F with congestion and length 1. Although the existence of F follows immediately because G'_i embeds into G with congestion 1 and length slack 1, below we will how to construct the flow F inductively level by level for the efficiency on constructing the edge representation flow_F of F.

Define $F'_i \leftarrow F'$. Let $\text{flow}_{F'_i} \leftarrow \text{flow}_{F'}$ be the edge representation of F'_i. We construct a flow $\widehat{F}_i$ in $G\cup G'_{i-1}$ as follows: for each directed edge $(v,w) \in \overleftrightarrow{E}(G'_i)$, $\widehat{F}_i$ routes $\text{flow}_{F'_i}(v,w)$ units of flow through the (v,w)-flow-paths of $\Pi_{G'_i\to G\cup G'_{i-1}}$. By Theorem 5.1(3), we have $|\text{path}(\Pi_{G'_i\to G\cup G'_{i-1}})| \le$

[2] We define $G'_{-1} = \emptyset$.

$(2\alpha)^{i+1}m$ and $\text{step}_{\Pi_{G'_i \to G \cup G'_{i-1}}} \le (2st+3s)h$. Thus, we can explicitly compute the path decomposition of $\widehat{F}_i$ as well as the its edge representation of $\text{flow}_{\widehat{F}_i}$ in $O(hst \cdot (2\alpha)^{i+1}m)$ work and $\tilde{O}(1)$ depth by explicitly summing the flow values overall flow paths of $\Pi_{G'_i \to G \cup G'_{i-1}}$.

Given $\widehat{F}_i$, we now define F_i and F'_{i-1} as the flow $\widehat{F}_i$ restricted to edges G and G'_{i-1}, respectively. More precisely, for each directed edge $(v,w) \in \overleftrightarrow{E}(G)$ where $\widehat{F}_i((v,w)) > 0$, F_i routes $\text{flow}_{\widehat{F}_i}(v,w)$ units of flows through a single edge (v,w). Similarly, for each directed edge $(v,w) \in \overleftrightarrow{E}(G'_{i-1})$ where $\widehat{F}_i((v,w)) > 0$, F'_{i-1} routes $\text{flow}_{\widehat{F}_i}(v,w)$ units of flows through (v,w). Since each flow path of F_i and F'_{i-1} routes through a single (directed) edge, the edge representations flow_{F_i} and $\text{flow}_{F'_{i-1}}$ can be trivially computed in linear work and $\tilde{O}(1)$ depth.

Next, we repeat the same argument on the edge representation $\text{flow}_{F'_{i-1}}$ of F'_{i-1} and until $i = 0$. Finally, we obtain the edge representations $\text{flow}_{F_i}, \text{flow}_{F_{i-1}}, \ldots, \text{flow}_{F_0}$ of $F_i, F_{i-1}, \ldots, F_0$ in G, respectively, such that, after concatenating the flow paths of $F_i, F_{i-1}, \ldots, F_0$, we can obtain the flow F that routes exactly the same demand as F'. We also have that the edge representation of $\text{flow}_F = \text{flow}_{F_i} + \text{flow}_{F_{i-1}} + \cdots + \text{flow}_{F_0}$. The total cost for constructing flow_F is then $O(hst \cdot (2\alpha)^{i+1}m)$ work and $\tilde{O}(i)$ depth as desired.

Note that we have $\text{leng}_F \le 1$ because $\Pi_{G'_i \to G \cup G'_{i-1}}$ has length slack 1 for all i. Moreover, $\text{cong}_F \le 1$ because $\Pi_{G'_i \to G \cup G'_{i-1}}$ has congestion 1 and each F_i only route through edges of G with length in the range $(h_0 h^{i-1}, h_0 h^i]$ by Theorem 5.1(3). □

By plugging the algorithm from Theorem 4.3 into Theorem 5.1 and Lemma 5.2, we obtain immediately the following, which will be used in the next section.

Corollary 5.3. *For any $\epsilon \in (\log^{-c}, 1)$ for some small enough constant c, there are parameters $t = \exp(\text{poly}1/\epsilon)$ and $\gamma = N^{\text{poly}\epsilon}$ such that there is a parallel algorithm that, given a m-edge graph G, and (h, d, ϵ, h_0) as parameters where $h_0 \le h$, computes graphs $G'_0, G'_1, G'_2, \ldots, G'_d$ such that for each index $i \le d$, G'_i is an $(h_0 h^i)$-length-constrained t-step emulator of G containing $m\gamma^i$ edges with length slack $O(t^2)^i$ and congestion slack γ^i. The algorithm has $m\gamma^d \text{poly}(h)$ work and $\tilde{O}(d \cdot \text{poly}(h))$ depth.*

For any i, given an edge representation of F' in G'_i, one can compute the edge representation flow_F of F in G that routes the same demand where $\text{cong}_F \le \text{cong}_{F'}$ and $\text{leng}_F \le \text{leng}_{F'}$ in $m\gamma^d h$ work and $\tilde{O}(d)$ depth.

The rest of this section is for proving Theorem 5.1.

5.1 Construction and Analysis

In this subsection, we prove three properties of Theorem 5.1 by induction on $i \ge 0$.

The base case $i = 0$ is straightforward. Let $G_{\le h_0}$ denote the graph containing only edges in G of length at most h_0. By the guarantee of $\mathcal{A}$, G'_0 is an h_0-length-constrained t-step emulator of $G_{\le h_0}$ with length slack s and congestion slack κ. By definition, G'_0 is also an h_0-length-constrained t-step emulator of G with the same ganrantees, because any flow F in G of length at most h_0 may route through only edges of length at most h_0. Moreover, $|E(G'_0)| \le \alpha m$ and $\Pi_{G'_0 \to G}$ has length slack 1, congestion 1, and path count $\alpha m \le 2\alpha^2 m$. The maximum step $\Pi_{G'_0 \to G}$ is $\text{step}_{\Pi_{G'_0 \to G}} \le \text{leng}_{\Pi_{G'_0 \to G}}$

Algorithm 2 EMULATORWITHBOOTSTRAPPING(G, h, d, t)

1. Let G'_0 be an h_0-length-constrained t-step emulator obtained by calling algorithm $\mathcal{A}$ on $G_{\le h_0}$, i.e., the graph containing only edges in G of length at most h_0.
2. For $1 \le i \le d$:
   (a) Let H_i be the unit-length graph constructed as follows:
      i. For each edge e in G'_{i-1}, add a corresponding unit-length edge e in H_i.
      ii. For each edge e in G with length in the range $(h_0 h^{i-1}, h_0 h^i]$, add an edge in H_i of length $\lceil \ell_G(e)/(h_0 h^{i-1}) \rceil$.
   (b) Let H'_i be a $(2t+3)h$-length-constrained t-step emulator obtained by calling algorithm $\mathcal{A}$ on H_i.
   (c) Let G'_i be the graph H'_i with all edges modified to have length $h_0 h^i \cdot s^{i+1}(2t+3)^i$.

because the edge length of G is integral. We have $\text{leng}_{\Pi_{G'_0 \to G}} \le s h_0 \le (2st+3s)h$ because the length of edges in G'_0 is $h_0 s$ and $\Pi_{G'_0 \to G}$ has length slack 1. So $\text{step}_{\Pi_{G'_0 \to G}} \le (2st+3s)h$. Finally, $\Pi_{G'_0 \to G}$ only routes through edges of G with length at most h_0, by definition of $G_{\le h_0}$.

For the rest of the proof, assume that the three properties hold for iteration $i-1$. The two lemmas below establish the analogues of Lemmas 4.5 and 4.6 from Section 4. We defer their proofs to Sections 5.2 and 5.3.

Lemma 5.4 (Backward Mapping). *The graph G'_i has at most $(2\alpha)^{i+1}m$ edges and there is an embedding $\Pi_{G'_i \to G \cup G'_{i-1}}$ with congestion 1, length slack 1, path count at most $(2\alpha)^{i+1}m$, and $\text{step}_{\Pi_{G'_i \to G \cup G'_{i-1}}} \le (2st+3s)h$. Also, there is an embedding $\Pi_{G'_i \to G}$ with congestion 1 and length slack 1.*

Lemma 5.5 (Forward Mapping). *Let F be a flow in G with $\text{leng}_F \le h_0 h^i$. There is a flow F' routing D_F in G'_i with $\text{cong}_{F'} \le (2\kappa)^{i+1} \cdot \text{cong}_F$ and $\text{step}_{F'} \le t$.*

Lemma 5.4 immediately implies properties (2) and (3) of Theorem 5.1 for iteration i. To see that property (1) is satisfied, we check each requirement in Definition 4.1:

1. By construction, edges in G'_i have the same length $h_0 h^i \cdot s^{i+1}(2t+3)^i$.
2. Given a flow F in G where $\text{leng}_F \le h_0 h^i$, Lemma 5.5 guarantees a flow F' in G' routing the same demand where $\text{cong}_{F'} \le (2\kappa)^i \cdot \text{cong}_F$ and $\text{step}_{F'} \le t$.
3. By Lemma 5.4, G'_i can be embedded into G with congestion 1 and length slack 1.

Finally, we show that the algorithm calls $\mathcal{A}$ on d many graphs, each with at most $O((2\alpha)^d m)$ edges, and runs in $O((2\alpha)^d m)$ work and $\tilde{O}(1)$ depth outside these calls. On each iteration $1 \le i \le d$, we call $\mathcal{A}$ on the graph H_i which consists of edges from G'_{i-1} and G (with their lengths modified), which is at most $O((2\alpha)^d m)$ edges in total. Outside of this call, the algorithm clearly runs in time linear in G'_{i-1} and G, which is $O((2\alpha)^d m)$ time. Over the d iterations, the total work outside calls to $\mathcal{A}$ is $O(d(2\alpha)^d m)$ and the total depth is $\tilde{O}(d)$.

5.2 Proof of Lemma 5.4: Backward Mapping

First, we show that the number of edges in G'_i is at most $(2\alpha)^{i+1}m$. This is because $|E(G'_i)| = |E(H'_i)| \le \alpha|E(H_i)|$ by the guarantee of $\mathcal{A}$, and $|E(H_i)| \le |E(G'_{i-1})| + |E(G)| \le (2\alpha)^i m + m$. So $|E(G'_i)| \le \alpha((2\alpha)^i + 1)m \le (2\alpha)^{i+1}m$. Our next goal is to construct the embedding $\Pi_{G'_i \to G \cup G'_{i-1}}$ and $\Pi_{G'_i \to G}$ with desired properties. To do this, we will show the following embedding: $\Pi_{G'_i \to H'_i}$, $\Pi_{H'_i \to H_i}$, $\Pi_{H_i \to G \cup G'_{i-1}}$, and $\Pi_{H_i \to G}$. We will obtain the goal by composing them. Recall that all G'_i have length $h_0 h^i \cdot s^{i+1}(2t+3)^i$ including the case when $i = 0$.

1. **From G'_i to H'_i:** Since G'_i is the graph H'_i with all edge lengths scaled up by factor $h_0 h^{i-1} s^i (2t+3)^{i-1}$, there is a trivial embedding $\Pi_{G'_i \to H'_i}$ with congestion 1 and length slack $1/(h_0 h^{i-1} s^i (2t+3)^{i-1})$.

2. **From H'_i to H_i:** By the guarantee of algorithm $\mathcal{A}$, it returns an embedding $\Pi_{H'_i \to H_i}$ with congestion 1, length slack 1. The path count $|\text{path}(\Pi_{H'_i \to H_i})|$ is at most $\alpha|E(H_i)| \le (2\alpha)^{i+1}m$. Next, we bound $\text{step}_{\Pi_{H'_i \to H_i}}$. We have $\text{step}_{\Pi_{H'_i \to H_i}} \le \text{leng}_{\Pi_{H'_i \to H_i}}$ because the edge length of H_i is integral. Also, flow path of $\Pi_{H'_i \to H_i}$ has length $\text{leng}_{\Pi_{H'_i \to H_i}} \le (2t+3)hs$ because each edge in H'_i has length $(2t+3)hs$ and the length slack of $\Pi_{H'_i \to H_i}$ is 1. So $\text{step}_{\Pi_{H'_i \to H_i}} \le (2t+3)hs$.

3. **From H_i to $G \cup G'_{i-1}$:** Since H_i is the graph G'_{i-1} scaled down by factor $h_0 h^{i-1} s^i (2t+3)^{i-1}$, together with edges in G with length in the range $(h_0 h^{i-1}, h_0 h^i]$ scaled down by factor at most $h_0 h^{i-1}$. So there is a trivial embedding $\Pi_{H_i \to G \cup G'_{i-1}}$ with congestion 1 and length slack $\max\{h_0 h^{i-1} s^i (2t+3)^{i-1}, h_0 h^{i-1}\} = h_0 h^{i-1} s^i (2t+3)^{i-1}$.

4. **From H_i to G:** We embed H_i further to G as follows. We split the trivial embedding $\Pi_{H_i \to G \cup G'_{i-1}}$ into an embedding Π_1 from H_i to G'_{i-1} and another embedding Π_2 from H_i to G that only congests edges of length more than $h_0 h^{i-1}$, each with congestion 1 and length slack at most $h_0 h^{i-1} s^i (2t+3)^{i-1}$. By induction, G'_{i-1} can be embedded into G with congestion 1 and length slack 1. Actually, we claim the stronger property that G'_{i-1} can be embedded with congestion 1 and length slack 1 into the subgraph of G consisting of all edges of length at most $h_0 h^i$. This is because the edges in G of length greater than h^i are ignored in the first $i-1$ levels of the construction. So the same inductive statement must hold on the graph with these edges taken out. Let $\Pi_{G'_{i-1} \to G}$ be this strengthened embedding. We compose Π_1 with $\Pi_{G'_{i-1} \to G}$ and then combine it with Π_2, we obtain an embedding $\Pi_{H_i \to G}$ with congestion 1 and length slack $h_0 h^{i-1} s^i (2t+3)^{i-1}$.

By composing the embedding $\Pi_{G'_i \to H'_i}$, $\Pi_{H'_i \to H_i}$, $\Pi_{H_i \to G \cup G'_{i-1}}$, we obtain $\Pi_{G'_i \to G \cup G'_{i-1}}$ with congestion 1 and length slack $1/(h_0 h^{i-1} s^i (2t+3)^{i-1}) \times 1 \times h_0 h^{i-1} s^i (2t+3)^{i-1} = 1$. Since both $\Pi_{G'_i \to H'_i}$ and $\Pi_{H_i \to G \cup G'_{i-1}}$ are trivial embedding, we have $|\text{path}(\Pi_{G'_i \to G \cup G'_{i-1}})| \le (2\alpha)^{i+1}m$ and $\text{step}_{\Pi_{G'_i \to G \cup G'_{i-1}}} \le (2t+3)hs$, inheriting the properties of $\Pi_{H'_i \to H_i}$. By the definition of H_i, we have $\Pi_{G'_i \to G \cup G'_{i-1}}$ only routes through edges of G with length in the range $(h_0 h^{i-1}, h_0 h^i]$.

The embedding $\Pi_{G'_i \to G}$ with congestion 1 and length slack 1 is obtained by composing $\Pi_{G'_i \to H'_i}$, $\Pi_{H'_i \to H_i}$, and $\Pi_{H_i \to G}$.

5.3 Proof of Lemma 5.5: Forward Mapping

Let F be a flow in G with $\text{leng}_F \le h_0 h^i$ from the lemma statement. Our goal is to show a flow F' routing D_F in G'_i with $\text{cong}_{F'} \le (2\kappa)^i \cdot \text{cong}_F$ and $\text{step}_{F'} \le t$.

Construct flow F^* on H_i. We first construct a flow F^* that routes D_F in H_i with $\text{leng}_{F^*} \le (2t+3)h$ and $\text{cong}_{F^*} \le ((2\kappa)^i+1)\cdot\text{cong}_F$. We start by decomposing the flow-paths in F as follows.

For each flow-path P in F, we first break down the path into at most $2h$ segments $P_1, P_2, \ldots$ such that each path P_i either has length at most $h_0 h^{i-1}$ or consists of a single edge. To do so, initialize $P' \leftarrow P$ and $i \leftarrow 1$, and while P' is non-empty, let path P_i be the longest prefix of P' of length at most $h_0 h^{i-1}$, or the first edge of P' if its length is already greater than $h_0 h^{i-1}$; then, remove the edges of P_i from P' and increment i by 1. If P_i is the longest prefix of P' of length at most $h_0 h^{i-1}$, and if there is an edge e after P_i in P', then the combined length of P_i and e is greater than $h_0 h^{i-1}$, and furthermore, e must be removed on the next iteration. It follows that every two iterations decreases the length of P' by at least $h_0 h^{i-1}$, and since P has length at most $h_0 h^i$, there are at most $2h$ many paths.

For each path P_i of length at most $h_0 h^{i-1}$, add it to a new flow F_1, and for the remaining paths P_i (consisting of single edges of length greater than $h_0 h^{i-1}$), add it to a new flow F_2. Let F_1 and F_2 be the final flows after repeating this procedure for all flow-paths P in F. By construction, we have $F = F_1 + F_2$, $\text{cong}_{F_1} \le \text{cong}_F$, $\text{cong}_{F_2} \le \text{cong}_F$, $\text{leng}_{F_1} \le h_0 h^{i-1}$, and $\text{step}_{F_2} = 1$, and moreover, each flow-path P in F decomposes into at most $2h$ flow-paths in F_1 and F_2.

By induction, property (1) guarantees that G'_{i-1} is an $h_0 h^{i-1}$-length-constrained t-step emulator for G with congestion slack $(2\kappa)^i$. Since $\text{leng}_{F_1} \le h_0 h^{i-1}$, there exists a flow F'_1 in G'_{i-1} routing demand D_{F_1} where $\text{cong}_{F'_1} \le (2\kappa)^i \cdot \text{cong}_{F_1} \le (2\kappa)^i \cdot \text{cong}_F$ and $\text{step}_{F'_1} \le t$. Since edges H_i have unit length, there is a corresponding flow F_1^* in H_i where $\text{cong}_{F_1^*} \le (2\kappa)^i \cdot \text{cong}_F$ and $\text{leng}_{F_1^*} \le t$.

By construction, each path in F_2 is a single edge e of length $\ell_G(e) \in (h_0 h^{i-1}, h_0 h^i]$, so there is a corresponding edge in H_i of length $\lceil \ell_G(e)/h_0 h^{i-1} \rceil \le \ell_G(x)/(h_0 h^{i-1}) + 1$. Let F_2^* be the flow in H_i that routes each single edge in F_2 through its corresponding edge in H_i. By construction, $\text{cong}_{F_2^*} = \text{cong}_{F_2} \le \text{cong}_F$.

Finally, we concatenate flows F_1^* and F_2^* in H_i as follows. For each flow-path P in F, consider the decomposition into at most $2h$ segments $P_1, P_2, \ldots$. For each path P_i of length at most $h_0 h^{i-1}$, take a corresponding flow in F_1^* of length at most t, and for each single-edge path P_i of length greater than $h_0 h^{i-1}$, take the flow in F_2^* along its corresponding edge in H_i. In both cases, the flow in H_i has length at most $\max\{t, \ell_G(P_i)/(h_0 h^{i-1})+1\}$. Concatenating these flows over all i produces a flow for path P of length at most $\sum_i \max\{t, \ell_G(P_i)/(h_0 h^{i-1})+1\} \le 2ht + h_0 h^i / h_0 h^{i-1} + 2h = (2t+3)h$. Over all flow-paths P, the final flow F^* in H_i satisfies $\text{leng}_{F^*} \le (2t+3)h$ and $\text{cong}_{F^*} \le \text{cong}_{F_1^*} + \text{cong}_{F_2^*} \le ((2\kappa)^i + 1) \cdot \text{cong}_F$.

Use emulator H'_i. Since H'_i is a $(2t+3)h$-length-constrained t-step emulator of H_i with congestion slack κ, there is a flow $F^\dagger$ in H'_i routing $D_{F^*} = D_F$ with $\text{cong}_{F^\dagger} \le \kappa \cdot \text{cong}_{F^*} \le \kappa \cdot ((2\kappa)^i+1) \cdot \text{cong}_F \le (2\kappa)^{i+1} \cdot \text{cong}_F$ and $\text{step}_{F^\dagger} \le t$. Finally, since G'_i is simply H'_i with edge length increased, the flow $F^\dagger$ in H'_i translates to a flow F' in G'_i with $\text{cong}_{F'} \le (2\kappa)^i \cdot \text{cong}_F$ and $\text{step}_{F'} \le t$ as promised.

6 Low-Step Emulators

In this section, we define and construct emulators similar to the length-constrained low-step emulators, but they preserve information about general flows instead of length-constrained flows.

To do this, we need a notion of *path-mapping* between flows. For any two flows F in G and F' in G' routing the same demand, observe that there exist path-decomposition of F and F' denoted by $\mathcal{P}$ and $\mathcal{P}'$, respectively, and a bijection $\pi : \mathcal{P} \to \mathcal{P}'$ such that for every $P \in \mathcal{P}$ and $P' = \pi(P)$, we have $\text{val}(F'(P')) = \text{val}(F(P))$. We call π a *path-mapping from F to F'*. We say that π has length slack s if $\ell_{G'}(\pi(P)) \leq s \cdot \ell_G(P)$.

Intuitively, if F can be mapped to F' via a path-mapping of length slack s, then F' is "as short as" F up to a factor of s. Now we are ready to define our main object.

Definition 6.1 (Low-Step Emulators)**.** *Given a graph G and a node-weighting A in G, we say that G' is a t-*step emulator *of A with length slack s and congestion slack κ if*

1. *Given a flow F in G routing an A-respecting demand, there is a flow F' in G' routing the same demand where $\text{cong}_{F'} \leq \kappa \cdot \text{cong}_F$ and $\text{step}_{F'} \leq t$. Moreover, there exists a path-mapping from F to F' with length slack s.*
2. *G' can be embedded into G with congestion 1 and length slack 1.*

Remark 6.2. *Let G' is a t-step emulator of A in G. Then, G' is simultaneous an h-length-constrained t-step emulator of A in G for all h, except that G' does not satisfies Condition 1 on length uniformity.*

The main theorems of this section are the construction of low-step emulators.

Theorem 6.3 (Existential)**.** *Given any graph G, a node weighing A and parameter t, there exists a t-step emulator G' for A in G with length slack $O(t)$, congestion slack $\text{poly}(t \log N) N^{O(1/\sqrt{t})}$, and $|E(G')| \leq n \cdot N^{O(1/\sqrt{t})} \text{poly}(\log N)$.*

Next, we show an algorithmic version of the above theorem when $A = \deg_G$ and the number of emulator edges is close to m, instead of n.

Theorem 6.4 (Algorithmic)**.** *Given any m-edge graph G and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, let $t = \exp(\text{poly}(1/\epsilon))$. There is a parallel algorithm $\textsc{LowStepEmu}(G, \epsilon)$ that constructs a t-step emulator G' for A in G with length slack $\exp(\text{poly}(1/\epsilon))$, congestion slack $N^{\text{poly}\epsilon}$, and $|E(G')| \leq mN^{\text{poly}\epsilon}$.*

The algorithms takes $mN^{\text{poly}\epsilon}$ work and $N^{\text{poly}\epsilon}$ depth. Given an edge representation $\text{flow}_{F'}$ of flow F' in G', there is an algorithm $\textsc{FlowMap}(G', \text{flow}_{F'})$ that computes an edge representation flow_F of flow F in G routing the same demand where $\text{cong}_F \leq \text{cong}_{F'}$ and $\text{leng}_F \leq \text{leng}_{F'}$ using $mN^{\text{poly}\epsilon}$ work and $\tilde{O}(1/\text{poly}\epsilon)$ depth.

It is a natural question to ask if we can obtain above emulator for general A whose number of edges is close to $|\text{supp}(A)|$. We believe this to be possible, but leave this for future work.

In the rest of this section, we prove Theorem 6.3 and Theorem 6.4. We will use the following basic lemma showing how to construct a t-step emulator given 2^i-length-constrained t-step emulator for each i.

Lemma 6.5. *Let A be a a node-weighting of G. For $i \in \{1, \ldots, d = \log N\}$, let G'_i be a 2^i-length-constrained t-step emulator of A with length slack s and congestion slack κ. Let $G' = \biguplus_i G'_i$ with capacity scaled down by d. Then G' is a t-step emulator of A with length slack $2st$ and congestion κd.*

Proof. We will argue the following:

1. Given a flow F in G routing an A-respecting demand with congestion 1, there is a flow F' in G' routing D_F where $\text{cong}_{F'} \le \kappa d$ and $\text{step}_{F'} \le t$. Moreover, there exist a path-mapping between F and F' with length slack $2st$.
2. G' can be embedded into G with congestion 1 and length slack 1.

For the first point, given F, we decompose $F = F_1 + \cdots + F_d$ so that F_i contains all flow paths P of F where $2^{i-1} \le \ell(P) < 2^i$ for $i \ge 1$. For each flow F_i in G, by Definition 4.1, there is a flow F'_i routing D_{F_i} in G'_i with $\text{cong}_{F'_i} \le \kappa$, $\text{step}_{F'_i} \le t$, and $\text{leng}_{F'_i} \le st \cdot 2^i$. Define $F' = F'_1 + \cdots + F'_d$ as a flow in G'. We now bound the congestion, step, and length slack of F', respectively. We have $\text{cong}_{G',F'} \le d \cdot \max_i \text{cong}_{G'_i,F'_i} \le \kappa d$ because G' is a disjoint union of G'_i after scaling down the capacity by d. Also, we have $\text{step}_{F'} \le t$ as $\text{step}_{F'_i} \le t$ for all i. To bound the length slack, since F_i and F'_i route the same demand, there exists a path mapping π_i from F_i to F'_i. Since all flow paths of F_i and F'_i have length at least 2^{i-1} and less than $st \cdot 2^i$ respectively, the length slack of π_i is at most $2st$. From $\pi_1, \ldots, \pi_d$, there exists a natural path-mapping π from F to F' with length slack $2st$. This completes the proof of the first point.

For the second point, for each i, let $\Pi_{G'_i \to G}$ be the embedding from G'_i into G with congestion 1 and length slack 1. Observe that $\Pi_{G' \to G} = \frac{1}{d} \biguplus_i \Pi_{G'_i \to G}$ is an embedding of G' (which is $\biguplus_i G'_i$ after scaling down the capacity by d) into G with congestion $d/d = 1$ and length slack 1. □

Existential Emulators: Proof of Theorem 6.3. From Theorem 4.2, for any i, there exists a 2^i-length-constrained t-step emulator G'_i for A in G with with length slack $s = O(t)$ and congestion slack $\kappa = \text{poly}(t \log N) N^{O(1/\sqrt{t})}$ where $|E(G'_i)| \le nN^{O(1/\sqrt{t})}\text{poly}(\log N)$.

By plugging $G'_1, \ldots, G'_{\log N}$ into Lemma 6.5, we obtain a t-step emulator G' for A in G with length slack $2st = O(t^2)$ and congestion slack $\kappa d = \text{poly}(t \log N) N^{O(1/\sqrt{t})}$. Since $G' = \biguplus_i G'_i$ with capacity scaled down, the bound of $|E(G'_i)|$ follow.

Algorithmic Emulators: Proof of Theorem 6.4. Given $\epsilon \in (\log^{-c}, 1)$ for some small enough constant c, let $t = \exp(\text{poly}(1/\epsilon))$ and $\gamma = N^{\epsilon^{c_0}}$ be the parameters from Corollary 5.3 for some constant $c_0 > 0$. Set $\epsilon' \leftarrow \epsilon^{c_0/2}$ and $h \leftarrow N^{\epsilon'}$ as the stacking parameter. We round up h so that it is a power of 2. For every $h_0 = 2^j$ where $h_0 \le h$, we do the following. Let $d' = \log_h N = 1/\epsilon'$. Therefore, $\gamma^{d'} \le N^{(\epsilon^{c_0})/\epsilon^{c_0/2}} = N^{\epsilon^{c_0/2}} = N^{\text{poly}\epsilon}$. We will exploit this inequality.

Construct $G'_{j,0}, G'_{j,1}, \ldots, G'_{j,d'}$ via Corollary 5.3 such that for each index k, $G'_{j,k}$ is an $(h_0 h^k)$-length-constrained t-step emulator of G. Each emulator $G'_{j,k}$ contains $|E(G'_{j,k})| \le m\gamma^{d'} = m \cdot N^{\text{poly}\epsilon}$ edges and has with length slack at most $O(t^2)^{d'} = \exp(\text{poly}(1/\epsilon) \cdot (1/\epsilon')) = \exp(\text{poly}(1/\epsilon))$ and congestion slack $\kappa \le \gamma^{d'} \le N^{\text{poly}\epsilon}$.

Therefore, we obtained 2^i-length-constrained t-step emulator G'_i of G containing $m \cdot N^{\mathrm{poly}\epsilon}$ edges with length slack $s = \exp(\mathrm{poly}(1/\epsilon))$ and congestion slack $\kappa = N^{\mathrm{poly}\epsilon}$ for all $i \le \log N$. By plugging these emulators G'_i into Lemma 6.5, we obtain a t-step emulator G' of G with length slack $2st = \exp(\mathrm{poly}(1/\epsilon))$ and congestion slack $\kappa \log N = N^{\mathrm{poly}\epsilon}$ where $G' = \biguplus_i G'_i$ with capacity scaled down by $\log N$. Clearly, we have $|E(G')| \le m \cdot N^{\mathrm{poly}\epsilon}$.

Let us analyze the construction time of G'. We call Corollary 5.3 $O(\log h) = O(1/\mathrm{poly}\epsilon)$ times, each of which takes $m\gamma^{d'}\mathrm{poly}(h) = mN^{\mathrm{poly}\epsilon}$ work and $\tilde{O}(d \cdot \mathrm{poly}(h)) = N^{\mathrm{poly}\epsilon}$ depth.

Finally, given an edge representation $\mathrm{flow}_{F'}$ of flow F' in G', we show how to compute the edge representation of the corresponding flow F in G. Given $\mathrm{flow}_{F'} : \overleftrightarrow{E}(G') \to \mathbb{R}_{\ge 0}$, we define $\mathrm{flow}_{F'_i}$ as $\mathrm{flow}_{F'}$ restricted to $\overleftrightarrow{E}(G'_i)$ that represents a flow F'_i in G'_i where $\mathrm{leng}_{F'_i} \le \mathrm{leng}_{F'}$ and $\mathrm{cong}_{F'_i} \le \mathrm{cong}_{F'}/\log N$ (because $G' = \biguplus_i G'_i$ with capacity scaled down by $\log N$). By Corollary 5.3, we can compute the edge representation flow_{F_i} of flow F_i in G where $D_{F_i} = D_{F'_i}$, $\mathrm{leng}_{F_i} \le \mathrm{leng}_{F'}$ and $\mathrm{cong}_{F_i} \le \mathrm{cong}_{F'}/\log N$ in $m\gamma^{d'}h = mN^{\mathrm{poly}\epsilon}$ work and $\tilde{O}(d) = \tilde{O}(1/\mathrm{poly}\epsilon)$ depth. By concatenating all F_i, we get the flow F in G where $D_F = D_{F'}$, $\mathrm{leng}_F \le \mathrm{leng}_{F'}$ and $\mathrm{cong}_F \le \sum_i \mathrm{cong}_{F_i}/\log N \le 1$. The edge representation flow_F of F can be defined as $\mathrm{flow}_F = \sum_i \mathrm{flow}_{F_i}$. Thus, we can return flow_F in $mN^{\mathrm{poly}\epsilon}$ work and $\tilde{O}(1/\mathrm{poly}\epsilon)$ depth.

7 Routing on an Expansion Witness

In this section, we prove the following.

Theorem 7.1 (Routing on a Router)**.** *Given a t-step γ-router R for a node weighting A, an A-respecting demand D and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, for parameters $\lambda_{rr}(\epsilon) = \exp(\mathrm{poly}(1/\epsilon))$ and $\kappa_{rr}(\epsilon) = N^{\mathrm{poly}(\epsilon)}$, one can compute a flow F routing D with*

- *congestion $\gamma\kappa_{rr}(\epsilon)$ and length $t\lambda_{\mathrm{rr}}(\epsilon)$, and*
- *support size $|\mathrm{supp}(F)| \le (|E(R)| + |\mathrm{supp}(D)|)N^{\mathrm{poly}\epsilon}$,*

with $(|E(R)| + |\mathrm{supp}(D)|) \cdot \mathrm{poly}(t)N^{\mathrm{poly}\epsilon}$ work and $\mathrm{poly}(t)N^{\mathrm{poly}\epsilon}$ depth.

Since an expansion witness covers neighbourhoods with routers, the above result for routing on a router gives as a corollary the following result for routing on a witness.

Corollary 7.2 (Routing on a Witness)**.** *Given a $(h, t_{\mathcal{R}}, t_\Pi, \kappa_\Pi)$-witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$ of A in G, an A-respecting demand D such that for all $(a, b) \in \mathrm{supp}(D)$ there exists $S \in \mathcal{S} \in \mathcal{N}$ such that $a, b \in S$, and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, one can compute a flow F routing D with*

- *length $h_\Pi t_{\mathcal{R}} \lambda_{rr}(\epsilon))$ and congestion $\kappa_\Pi \kappa_{rr}(\epsilon)$, and*
- *support size $|\mathrm{supp}(F)| \le (|\mathrm{path}(\Pi_{\mathcal{R}\to G})| + |\mathrm{supp}(D)|)N^{\mathrm{poly}\epsilon}$,*

with $(|\mathrm{supp}(D)| + |\mathrm{path}(\Pi_{\mathcal{R}\to G})|) \cdot \mathrm{poly}(t_{\mathcal{R}})N^{\mathrm{poly}(\epsilon)}$ work and $\mathrm{poly}(t_{\mathcal{R}})N^{\mathrm{poly}(\epsilon)}$ depth.

7.1 Section Preliminaries

Our algorithm is based on the *cut-matching game*, especially in the low hop-length regime developed in [HHG22].

Cut-Matching Game. For a node weighting A, a cut-matching game with r rounds produces a sequence of capacitated unit-length graphs $G^{(0)}, \ldots, G^{(r)}$ on $\mathrm{supp}(A)$, where $G^{(0)}$ is the empty graph. In round i, the cut-player selects based on $G^{(0)}, G^{(1)}, \ldots, G^{(i)}$ a pair of node weightings $(A^{(i)}, B^{(i)})$, where $A^{(i)}$ and $B^{(i)}$ are A-respecting, have disjoint support, and have equal size (i.e. $|A^{(i)}| = |B^{(i)}|$). The matching player then produces a capacitated unit-length matching graph $\widetilde{G}^{(i)}$, with edges between the supports of $A^{(i)}$ and $B^{(i)}$, and capacities such that $\deg_{\widetilde{G}^{(i)}} = A^{(i)} + B^{(i)}$. The produced graphs are then added to the current graph: $G^{(i+1)} := G^{(i)} + \widetilde{G}^{(i)}$.

Matching Strategy. For a node weighting A, a (r, t, η, Δ)-cut strategy for the cut matching game with r rounds produces, regardless of the matching player, a graph $G^{(r)}$ which is a t-step η-congestion router for A with $\deg_{G^{(r)}} \prec \Delta A$.

We use the following cut-matching strategy of [HHG22].

Theorem 7.3 (Good Cut Strategy, Theorem 4.2 of [HHG22])**.** *For every node weighting A and every $t \leq \log N$, there exists a (r, t, η, Δ)-cut strategy with $r, \eta, \Delta \leq N^{\mathrm{poly}(1/t)}$. Suppose that each graph $G^{(i)}$ produced by the matching player contains at most m' edges. Then, such a cut strategy can be computed in time $\tilde{O}(\mathrm{poly}(t) \cdot (T(m') + m'))$, where $T(m')$ is the time needed for computing an $(h, 2^t)$-length $(\phi, N^{\mathrm{poly}(1/t)})$-expander decomposition on a capacitated unit-length m'-edge graph.*

Using the expander decomposition algorithm of Theorem 3.18 with $\epsilon = 1/t$ (and disregarding linkedness), we immediately obtain the following.

Corollary 7.4. *For every node weighting A and every $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, we can compute a (r, t, η, Δ)-cut strategy with $t = 1/\epsilon$ and $r, \eta, \Delta \leq N^{\mathrm{poly}\epsilon}$ in depth $\tilde{O}(1) \cdot \mathrm{poly}(t) N^{\mathrm{poly}\epsilon}$ and work $\tilde{O}(m' + |\mathrm{supp}(A)|) \cdot \mathrm{poly}(t) N^{\mathrm{poly}\epsilon}$, where m' is the maximum number of edges in any graph produced by the matching player.*

Note that what we use here has two differences to the result stated in [HHG22]:

- The cut matching game is on a node weighting, instead of a vertex set.
- The matching player can return arbitrary complete flows between the two vertex sets, instead of being restricted to return a perfect matching.

Obtaining the first generalization is simple: first, round down the node weighting A into powers of two, and bucket equal powers of two, forming node weightings $A_1, \ldots, A_b$ (for $b \leq \log N$) such that $A_i(v) \in \{0, 2^i\}$, exactly one of $A_i(v)$ is nonzero for any v, and $A/2 \leq \sum_i A_i \leq A$. Then, the node weighting A_i of maximum value $|A_i|$ satisfies $|A_i| \geq \frac{1}{b} \sum_{i'} |A_{i'}|$. Next, we run a cut matching game on vertex set $|\mathrm{supp}(A_i)|$, with step bound $t - 2$. Finally, the remaining vertices need to be connected to the router constructed on $|\mathrm{supp}(A_i)|$. This can be done through $b - 1$ cuts, each of which has A_j, $j \neq i$ as one side, and a subdemand $A'_i \leq A_i$ of size $|A'_i| = |A_j|$ as the other side. Now, any demand can be routed with congestion b times higher (which can be absorbed into the $N^{\mathrm{poly}\epsilon}$-factor) through paths of length $(t - 2) + 2 = t$.

The second generalization is possible with a capacitated length-bounded expander decomposition algorithm. The algorithm or potential analysis of the cut-matching game needs no changes.

Length-Constrained Multicommodity Flow. We will use the following lemma for h-length k-commodity flow, obtained by applying a result of [HHT24] to the specific case of routers. Note that the *additive* k in support size (as opposed to multiplicative k as in the algorithms of [HHS23]) is critical for our application. For derivation of Lemma 7.5, see Appendix A.

Lemma 7.5. *Let $G = (V, E)$ be a t-step γ-router for node weighting A, and $\{(A_j, A'_j)\}_{j\in[k]}$ node weighting pairs such that $\sum_j A_j + A'_j$ is A-respecting and $|A_j| = |A'_j|$ for all j. Then, one can compute a flow $F = \sum_j F_j$ where F_j is a complete A_j to A'_j -flow for each $j \in [k]$ with*

- *length $2t$ and congestion $\tilde{O}(\gamma)$, and*
- *support size $|\mathrm{supp}(F)| \le \tilde{O}(|E(G)| + k + \sum_j |\mathrm{supp}(A_j + A'_j)|)$,*

with $\tilde{O}((|E(G)| + \sum_j |\mathrm{supp}(A_j + A'_j)|) \cdot k \cdot \mathrm{poly}(t))$ work and $\tilde{O}(k \cdot \mathrm{poly}(t))$ depth.

7.2 Algorithm for Routing on a Router

In this section, we present Algorithm 3, which routes a flow on a *router*. The correctness of the algorithm is proven in Section 7.3. The simple extension to routing on a witness is performed in Section 7.4.

In addition to the length-bounded cut-matching game strategy and flow algorithm presented in the section preliminaries, Algorithm 3 needs two simple functions SplitFlow and ConcatFlow for manipulating flows.

SplitFlow splits a complete flow F from a node weighting $A = \sum_{i\in k} A_i$ to A' into complete flows F_i from node weightings A_i to A'_i for $\sum_i F_i = F$ and $\sum_i A'_i = A'$:

Lemma 7.6. *Let G be a graph, $A = \sum_{i\in k} A_i$ and A' be node weightings, and F be a complete flow from A to A'. Then, there is a deterministic algorithm* $\mathrm{SplitFlow}(F, \{A_i\}_{i\in k})$ *that returns (flow, node weighting) pairs $\{(F_i, A'_i)\}$ such that*

- *$\sum_i F_i = F$ and $\sum_i A'_i = A'$,*
- *F_i is a complete flow from A_i to A'_i, and*
- *$\sum_i |\mathrm{supp}(A'_i)| \le \sum_i |\mathrm{supp}(F_i)| \le \sum_i |\mathrm{supp}(A_i)| + |\mathrm{supp}(F)|$*

The algorithm has work $\tilde{O}(|\mathrm{supp}(F)|)$ and depth $\tilde{O}(1)$.

Proof. For every vertex $v \in \mathrm{supp}(A)$, let $I := \{i : A_i(v) > 0\}$ be the set of node weightings with nonzero weight on v and $a = |I|$, and let $P_1, \ldots, P_b \in \mathrm{supp}(F)$ be the flow paths from v. Fix an arbitrary order of the a node weightings and b paths. Then, form $a + b - 1$ pairs (i, j), where a pair is formed if the prefix sums from node weighting $i - 1$ to i and paths $j - 1$ to j overlap. For every pair, add flow path P_j to flow F_i with value equal to the overlap. As sorting can be done with depth $\tilde{O}(1)$, this can be done with depth $\tilde{O}(1)$. □

ConcatFlow *concatenates* a complete flow F from node weighting A to node weighting A' with a complete flow F' from node weighting A' to node weighting A'', forming a complete flow F^{res} from A to A''.

Lemma 7.7. *Let G be a graph, A, A' and A'' be node weightings with $|A| = |A'| = |A''|$, F a complete A-to-A'-flow, and F' a complete A'-to-A''-flow. Then, there is a deterministic algorithm* $\text{ConcatFlow}(F, F')$ *that returns a complete A-to-A''-flow F^{res} such that*

- *Each flow path in F^{res} is a concatenation of a flow path in F with a flow path in F', and the total flow value of flow paths using a path $P \in F$ or $P' \in F'$ is $F(P)$ or $F(P')$ respectively.*
- $|\text{supp}(F^{\text{res}})| \leq |\text{supp}(F)| + |\text{supp}(F')|$.

The algorithm has work $\tilde{O}(|\text{supp}(F)| + |\text{supp}(F')|)$ and depth $\tilde{O}(1)$.

Proof. For every vertex $v \in \text{supp}(A')$, let $P_1, \ldots, P_a \in \text{supp}(F)$ be the flow paths of F to v, and $P'_1, \ldots, P'_b$ the flow paths of F' from v. We have $\sum_i F(P_i) = \sum_j F'(P_j)$ as F is a complete flow to A' and F' is a complete flow from A'.

Now, as with Lemma 7.6, pair paths P_i with paths P_j if the prefix sums of flow value overlap, to form $a + b - 1$ pairs (i, j). For every such pair, add a path P that consists of P_i concatenated with P_j of value equal to the overlap. □

Brief overview of Algorithm 3. The algorithm is recursive, with the goal of splitting the instance into k smaller instances of roughly the same total size (where the "size" of an instance equals the number of edges in the router plus the number of pairs in the demand). Suppose the graph contains more than one vertex (otherwise, we are in a trivial base case). Then, the algorithm performs the following steps:

1. Split the vertex set of the router into k equal size vertex subsets.
2. Embed a router in each of the vertex subsets, by running a cut-matching game in each subset in parallel.
3. Compute flows sending demand from each vertex subset to the correct vertex subset, so that it only remains to route flow within each vertex subset.
4. Recursively route the resulting demand within each vertex subset using the embedded router.

Each recursive instance will be on a vertex set that is a factor k smaller than the current one, for a recursion depth of $\log_k |V(R)|$. Thus, if the total size of the recursive instances is at most N^ϵ times the size of the current instance, the total size of the instances at the bottommost level of the recursion is $N^{\epsilon \cdot \log_k |V(R)|}$ times the initial size – for $k = |V(R)|^\epsilon$, this is $N^{\text{poly}\epsilon}$. Notably, we must ensure that the total size of the recursive instances is not k times the size of the current instance.

In step 2, the routers we recurse on are constructed through parallel cut-matching games: we run an independent cut-matching game in each of the k vertex subsets. Each round i, we receive from each game $j \in [k]$ a partition $(A_j^{(i)}, B_j^{(i)})$ of the node weighting of its vertex subset. To construct the matching, we call Lemma 7.5 *once*, with k commodities, one for each pair $(A_j^{(i)}, B_j^{(i)})$. Thus, the number of flow paths produced is $\tilde{O}(|E(G)| + k + \sum_j |\text{supp}(A_j^{(i)} + B_j^{(i)})|) \leq \tilde{O}(|E(G)| + k)$ instead of $\tilde{O}(k \cdot |E(G)|)$; as each flow path will correspond to an edge in the constructed routers, this avoids multiplying the total size of the recursive instances by k. The cut-matching games need to run for $N^{\text{poly}\epsilon}$ rounds, but this is an acceptable blowup in instance size.

For step 3, we want to route the part of the demand that starts and ends at different vertex subsets (say, A_j and $A_{j'}$) from A_j to $A_{j'}$, so that it only remains to route flow within each subset. To do this, we create a k^2-commodity flow instance, where each commodity corresponds to a pair of source-vertex-subset, destination-vertex-subset, with the node weightings equal to the amount of flow each vertex of the subsets needs to send or receive from the other subset. We call Lemma 7.5 once to solve this flow instance; again, the additive k^2 in support size is critical, as this goes to the total support size of the demands in the recursive instance.

Step 4 is then simple. It remains to route flow within each vertex subset, which we can do using the routers embedded in step 2. After computing these flows, we map them back to the original graph using the embeddings of the routers, and concatenate the flows of step 3 with those produced recursively.

For notational simplicity, the algorithm represents the demands with node weightings: the demand is a set of node weightings $\{S_w\}_{w \in V}$, with the correspondence $S_w(v) = D(v, w)$.

7.3 Analysis of the Algorithm

Proof of correctness of Algorithm 3 is split into multiple simple lemmas. Lemma 7.8 certifies the validity of the recursive instance and bounds its size, Lemma 7.9 bounds the work and depth of a single call, excluding recursive calls, and Lemma 7.10 bounds the congestion, length and support size of the produced flow. After proving the lemmas, we combine them for the proof of Theorem 7.1.

Lemma 7.8. *For each recursive call made by Algorithm 3,*

- $G_{j'}$ *is a* t'*-step* η*-router for* $A_{j'}$*, and*
- $\sum_{w \in V_{j'}} S'_w$ *and* $|S'_w|$ *(restricted to* $V_{j'}$*) are* $A_{j'}$*-respecting,*

and the total size of recursive instances satisfies

$$\sum_{j'} \left(|E(G_{j'})| + \sum_{w \in V_{j'}} |\mathrm{supp}(S'_w)| \right) \leq \tilde{O} \left(k^2 + |E(G)| + \sum_{w \in V} |\mathrm{supp}(S_w)| \right) N^{\mathrm{poly}\epsilon}.$$

Proof. For the first claim, the graphs $G_{j'}$ are produced through a (r, t', η, Δ)-cut strategy for the node weightings $A_{j'}$, and are thus t'-step η-routers for $A_{j'}$.

For the second claim, for all j, j', the sum $\sum_{w \in V_{j'}} S'_{w,j}$ is $B_{j,j'}$-respecting, as $F^{\mathrm{match}}_{j,j'}$ is a complete $A_{j,j'}$, $B_{j,j'}$-flow, and Lemma 7.6 partitions the destination node weighting. We have $B_{j,j'}(w) := |S_{w,j}|$ for $w \in V_{j'}$, thus $\sum_j B_{j,j'}(w) = |S_w|$ for $w \in V_{j'}$. Since $|S_w|$ is A-respecting and $A_{j'}$ is A restricted to $V_{j'}$, $|S_w|$ is $A_{j'}$-respecting for $w \in V_{j'}$. Finally, by Lemma 7.6, $|S'_w| = |S_w|$.

It remains to bound the total size of the recursive instances. We show the following two bounds, which combine to the desired bound:

- $\sum_w |\mathrm{supp}(S'_w)| \leq \tilde{O}(k^2 + E(G) + \sum_w |\mathrm{supp}(S_w)|)$.
- $\sum_{j'} |E(G_{j'})| \leq \tilde{O}(|E(G)| + k) N^{\mathrm{poly}\epsilon}$.

For the first bound,

Algorithm 3 Router Routing Algorithm RouteRouter$(G, t, \gamma, \epsilon, A, \{S_w\}_{w\in V}, k)$

Input: t-step γ-router $G = (V, E)$ for a node weighting A with edge capacities u, demand node weightings S_w ($D(v, w) = S_w(v)$) such that $\sum_{w\in V} S_w$ and $|S_w|$ are A-respecting, and a recursion parameter $k \geq 1$.
Output: Multicommodity flow $F = \sum_w F_w$ satisfying the demand (i.e. $\text{val}((F_w)_{(v,w)}) = S_w(v)$).

0. Base case: If $|V| = 1$, output the empty flow as there is no demand to satisfy.

1. Partition vertices into k parts: let $V_1, \ldots, V_k$ be a partition of V such that $\big||V_j| - |V_{j'}|\big| \leq 1$ for all $j, j' \in [k]$, and let $A_j, S_{w,j}$ be the node weightings A, S_w restricted to V_j.

2. Construct routers G_j for each A_j with embeddings Π_j: run k cut-matching games in parallel, one on each A_j, using a (r, t', η, Δ)-cut strategy of Corollary 7.4 with parameters $t' = 1/\epsilon$ and $r, \eta, \Delta \leq N^{\text{poly}\epsilon}$:

   (a) For each round $i = 1, 2, \ldots, r$ sequentially:

      i. For each $j \in [k]$, let $A_j^{(i)}, B_j^{(i)} \leq A_j$ be the ith node weightings produced by the cut strategy on A_j.

      ii. Let $F^{(i)} = \sum_j F_j^{(i)}$ be a k-commodity flow for node weighting pairs $\{(A_j^{(i)}, B_j^{(i)})\}_{j\in[k]}$ computed using Lemma 7.5.

      iii. For each flow-path $P \in F_j^{(i)}$ with endpoints v, w, add an edge $e = (v, w)$ of capacity $F_j^{(i)}(P)$ to the matching graph $\tilde{G}_j^{(i)}$, with embedding $\Pi_j(e) = P$.

   (b) For each $j \in [k]$, let $G_j = G_j^{(r)}$ be the constructed router and Π_j its embedding.

3. Compute flows between pairs $V_j, V_{j'}$:

   (a) For each $j, j' \in [k]$, let $A_{j,j'}(v) := \sum_{w\in V_{j'}} S_{w,j}(v)$ and $B_{j,j'}(w) := |S_{w,j}|I[w \in V_{j'}]$.

   (b) Let $F^{\text{match}} = \sum_{j,j'} F_{j,j'}^{\text{match}}$ be a k^2-commodity flow for node weighting pairs $\{(A_{j,j'}, B_{j,j'})\}_{j,j'\in[k]}$ computed using Lemma 7.5.

   (c) For all $j, j' \in [k]$, let $\{(F_{w,j}^{\text{match}}, S'_{w,j})\}_{w\in V_{j'}} := \text{SplitFlow}(F_{j,j'}^{\text{match}}, \{S_{w,j}\}_{w\in V_{j'}})$.

   (d) Let $S'_w := \sum_{j\in[k]} S'_{w,j}$.

4. Recurse to route $\{S'_w\}_{w\in V_{j'}}$ inside each $G_{j'}$:

   (a) For all $j' \in [k]$,

      i. Let $F_{j'}^{\text{rec}} := \sum_{w\in V_{j'}} F_w^{\text{rec}} := \text{RouteRouter}(G_{j'}, t', \eta, \epsilon, A_{j'}, \{S'_w\}_{w\in V_{j'}}, k)$.

      ii. Let $F_w^{\text{tail}} := \Pi_{j'}(F_w^{\text{rec}})$ for $w \in V_{j'}$.

      iii. Let $\{(F_{w,j}^{\text{tail}}, \cdot)\}_{j\in[k]} := \text{SplitFlow}(F_w^{\text{tail}}, \{S'_{w,j}\}_{j\in[k]}))$ for $w \in V_{j'}$.

   (b) Return $F = \sum_w F_w = \sum_{j,w} \text{ConcatFlow}(F_{w,j}^{\text{match}}, F_{w,j}^{\text{tail}})$

- By Lemma 7.6, $\sum_w |\text{supp}(S'_w)| \le \sum_{w,j} |\text{supp}(S'_{w,j})| \le \sum_w |\text{supp}(S_w)| + |\text{path}(F^{\text{match}})|$.
- By Lemma 7.5, $|\text{supp}(F^{\text{match}})| \le \tilde{O}(k^2 + E(G) + \sum_{j,j'} |\text{supp}(A_{j,j'} + B_{j,j'})|) = \tilde{O}(E(G) + k^2 + \sum_w |\text{supp}(S_w)|)$.
- Thus, $\sum_w |\text{supp}(S'_w)| \le \tilde{O}(k^2 + E(G) + \sum_w |\text{supp}(S_w)|)$.

For the second bound,

- By Lemma 7.5, $\sum_j |\text{supp}(F_j^{(i)})| = |\text{supp}(F^{(i)})| \le \tilde{O}(E(G) + k + \sum_j |\text{supp}(A_j^{(i)} + B_j^{(i)})|) = \tilde{O}(E(G) + k)$.
- The cut strategy (Corollary 7.4) used has $r \le N^{\text{poly}\epsilon}$ rounds, thus $\sum_j |E(G_j)| \le \tilde{O}(|E(G)| + k)N^{\text{poly}\epsilon}$.

□

Lemma 7.9. *In a call to Algorithm 3, excluding the recursive calls, the work done is* $\tilde{O}((|E(G)| + \sum_w |\text{supp}(S_w)|) \cdot k^2 \cdot N^{\text{poly}\epsilon} \cdot \text{poly}(t))$ *and the depth is* $\tilde{O}(k^2 \cdot N^{\text{poly}\epsilon} \cdot \text{poly}(t))$.

Proof. Excluding the recursive calls, work done outside calls to Lemma 7.5 or Corollary 7.4 is negligble. For those two,

- the cut-strategy of Corollary 7.4 can be computed with work $\tilde{O}(|E(G)| + k) \cdot \text{poly}(t)N^{\text{poly}\epsilon}$ and depth $\tilde{O}(1) \cdot \text{poly}(t)N^{\text{poly}\epsilon}$, as the maximum number of edges m' in a matching graph produced is $m' = \tilde{O}(|E(G)| + k)$. The total number of cut-strategies computed is k.
- computing the flows for the cut-matching games takes $\tilde{O}(|E(G)| \cdot k \cdot \text{poly}(t))$ work and $\tilde{O}(k \cdot \text{poly}(t))$ depth, and is done $r = N^{\text{poly}\epsilon}$ times. Computing the flows for matching vertex sets V_j, V'_j takes $\tilde{O}((|E(G)| + \sum_w |\text{supp}(S_w)|) \cdot k^2 \cdot \text{poly}(t))$ work and $\tilde{O}(k^2 \cdot \text{poly}(t))$ depth, and is done once.

Thus, the total work excluding recursive calls is $\tilde{O}((|E(G)| + \sum_w |\text{supp}(S_w)|) \cdot k^2 \cdot N^{\text{poly}\epsilon} \cdot \text{poly}(t))$, and the depth is $\tilde{O}(k^2 \cdot N^{\text{poly}\epsilon} \cdot \text{poly}(t))$. □

Lemma 7.10. *Suppose that each flow* $F^{\text{rec}}_{j'} := \sum_{w \in V_{j'}} F^{\text{rec}}_w$ *returned by the recursive calls from a call to Algorithm 3 satisfies the demand* $\{S'_w\}_{w \in V_{j'}}$ *(i.e.* $\text{val}((F^{\text{rec}}_w)_{v,w}) = S'_w(v)$*), with*

- *length at most* t^{rec},
- *congestion at most* κ^{rec}, *and*
- *support size at most* $s^{\text{rec}} \cdot (|E(G_{j'})| + \sum_{w \in V_{j'}} |\text{supp}(S'_w)|)$.

Then, the flow $F = \sum_w F_w$ *returned satisfies the demand* $\{S_w\}_{w \in V}$ *with*

- *length at most* $t^{\text{rec}} \cdot O(t)$,
- *congestion at most* $\kappa^{\text{rec}} \cdot \tilde{O}(\gamma N^{\text{poly}\epsilon})$, *and*

- *support size at most* $\left(s^{\text{rec}} \cdot \tilde{O}(N^{\text{poly}\epsilon})\right) \cdot (|E(G)| + \sum_w |\text{supp}(S_w)|) + \tilde{O}(k^2)$.

Proof. First, consider the flows $F^{\text{tail}}_w := \Pi_{j'}(F^{\text{rec}}_w)$. The length of any path in this flow is at most $2t \cdot t^{\text{rec}}$, as the length of each flow path computed by Lemma 7.5 is at most $2t$, and each edge in $G_{j'}$ maps to one such flow path. For this reason, we also have $|\text{supp}(F^{\text{tail}}_w)| = |\text{supp}(F^{\text{rec}}_w)|$.

For congestion, consider some edge e of $E(G)$. The total capacity of edges in the graphs $G_{j'}$ that map to a flow path containing e is at most the number of rounds r of the cut strategy times the maximum congestion of a flow produced by Lemma 7.5, which is $\tilde{O}(\gamma)$, and the maximum congestion of an edge $e' \in E(G_{j'})$ is the edge's capacity times κ^{rec}, thus the congestion of $\sum_w F^{\text{tail}}_w$ is at most $r \cdot \tilde{O}(\gamma) \cdot \kappa^{\text{rec}} = \tilde{O}(\gamma \cdot \kappa^{\text{rec}}) \cdot N^{\text{poly}\epsilon}$.

Next, consider the flows $F^{\text{tail}}_{w,j}$. The total congestion and dilation of these flows follows from $\sum_{w,j} F^{\text{tail}}_{w,j} = \sum_w F^{\text{tail}}_w$. For support size, by Lemma 7.6, we have

$$\sum_{w,j} |\text{supp}(F^{\text{tail}}_{w,j})| \leq \sum_w (\sum_j |\text{supp}(S'_{w,j})| + |\text{supp}(F^{\text{tail}}_w)|) \leq \sum_{w,j} |\text{supp}(S'_{w,j})| + \sum_w |\text{supp}(F^{\text{rec}}_w)|.$$

and, using the bound $\sum_{w,j} |\text{supp}(S'_{w,j})| \leq \tilde{O}(k^2 + E(G) + \sum_w |\text{supp}(S_w)|)$ from the proof of Lemma 7.8 and the assumption from the lemma,

$$\sum_{w,j} |\text{supp}(F^{\text{tail}}_{w,j})| \leq \left(s^{\text{rec}} \cdot \tilde{O}(N^{\text{poly}\epsilon})\right) \cdot (|E(G)| + \sum_w |\text{supp}(S_w)|) + \tilde{O}(k^2).$$

Now, consider the flows $F^{\text{match}}_{j,j'}$. As they are computed by a single call to Lemma 7.5, the congestion of $\sum_{j,j'} F^{\text{match}}_{j,j'}$ is at most $\tilde{O}(\gamma)$, the length of each flow path is at most $2t$, and

$$\sum_{j,j'} |\text{supp}(F^{\text{match}}_{j,j'})| \leq \tilde{O}(k^2 + E(G) + \sum_{j,j'} |\text{supp}(A_{j,j'} + B_{j,j'})|) = \tilde{O}(k^2 + E(G) + \sum_w |\text{supp}(S_w)|)$$

as argued in the proof of Lemma 7.8. As before, after splitting, the flow $\sum_{w,j} F^{\text{match}}_{w,j}$ retains the same congestion bound $\tilde{O}(\gamma)$, the length bound $2t$ and has the same support size: $\tilde{O}(k^2 + E(G) + \sum_w |\text{supp}(S_w)|) + \sum_{w,j} |\text{supp}(S_{w,j})| = \tilde{O}(k^2 + E(G) + \sum_w |\text{supp}(S_w)|)$.

The returned flow is the concatenation of the flows $F^{\text{match}}_{w,j}$ with the flows $F^{\text{tail}}_{w,j}$. Thus, by Lemma 7.7, the congestion is at most the sum of the two congestions, the length the sum of the two lengths, and the support size the sum of the two support sizes. In each case, the quantity of $F^{\text{tail}}_{w,j}$ is larger, giving the desired bounds.

It remains to show the flow satisfies the demand $\{S_w\}_{w \in V}$. This is simple: as the flow $F^{\text{match}}_{w,j}$ is a complete $S_{w,j}$-to-$S'_{w,j}$-flow, and $F^{\text{tail}}_{w,j}$ is a complete $S'_{w,j}$-to-w-flow, their concatenation is a complete $S_{w,j}$-to-w-flow. The sum of such flows over j is a complete S_w-to-w-flow, as desired. □

We are ready to prove Theorem 7.1.

Theorem 7.1 (Routing on a Router)**.** *Given a t-step γ-router R for a node weighting A, an A-respecting demand D and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, for parameters $\lambda_{rr}(\epsilon) = \exp(\text{poly}(1/\epsilon))$ and $\kappa_{rr}(\epsilon) = N^{\text{poly}(\epsilon)}$, one can compute a flow F routing D with*

- *congestion* $\gamma\kappa_{rr}(\epsilon)$ *and length* $t\lambda_{\mathrm{rr}}(\epsilon)$*, and*
- *support size* $|\mathrm{supp}(F)| \le (|E(R)| + |\mathrm{supp}(D)|)N^{\mathrm{poly}\epsilon}$*,*

with $(|E(R)| + |\mathrm{supp}(D)|) \cdot \mathrm{poly}(t)N^{\mathrm{poly}\epsilon}$ *work and* $\mathrm{poly}(t)N^{\mathrm{poly}\epsilon}$ *depth.*

Proof. We produce a flow with the properties claimed by the theorem by calling Algorithm 3 with the graph R and node weighting A (which is a t-step γ-router), $\{S_w\}_{w\in V}$ defined by $S_w(v) = D(v, w)$, $k = |V(R)|^\epsilon$ (which results in recursion depth $d := \lceil \log_k |V(R)| \rceil = \lceil 1/\epsilon \rceil$, and error parameter $\epsilon' = \epsilon^{1/c'}$ for a sufficiently large constant c', such that

- $\tilde{O}(N^{\mathrm{poly}\epsilon'})^d \le N^{\mathrm{poly}\epsilon}$, and
- $O(1/\epsilon')^d \le \exp(\mathrm{poly}(1/\epsilon))$.

Note that this is possible as $\epsilon \ge \log^{-c} N$, thus $\tilde{O}(N^{\mathrm{poly}\epsilon'}) = N^{\mathrm{poly}\epsilon'}$ for a different polynomial.

Now, having set the parameters, consider the jth level of recursion, for $0 \le j \le d$. As the recursion always splits into k instances, there are k^j recursive instances at this level. Let siz_j denote the total size (sum of $|E(G')| + \sum_w |\mathrm{supp}(S'_w)|$) of instances at the jth level of recursion (with $\mathrm{siz}_0 = |E(R)| + \sum_w |\mathrm{supp}(S_w)|$. We have

$$\mathrm{siz}_{j+1} \le \left(\mathrm{siz}_j + k^j \cdot k^2\right) \tilde{O}(N^{\mathrm{poly}\epsilon'}),$$

thus $\mathrm{siz}_j \le (\mathrm{siz}_0 + k^{j+1}) \cdot \tilde{O}(N^{\mathrm{poly}\epsilon'})^j$.

At every recursion level except the topmost, the graph G' is a t'-step η-router for $t' = 1/\epsilon'$ and $\eta \le N^{\mathrm{poly}\epsilon'}$. Let t_j^{rec} be the maximum length of a flow returned from the jth recursion level and κ_j^{rec} the maximum congestion, as in Lemma 7.10. Then, we have

- $t_{j-1}^{\mathrm{rec}} \le t_j^{\mathrm{rec}} \cdot O(1/\epsilon')$, and
- $\kappa_{j-1}^{\mathrm{rec}} \le \kappa_j^{\mathrm{rec}} \cdot \tilde{O}(N^{\mathrm{poly}\epsilon'})$.

For support size, the total support size of the flows returned from the second-bottommost level is at most $\mathrm{siz}_{d-1} \cdot \tilde{O}(N^{\mathrm{poly}\epsilon'}) + k^{d-1} \cdot \tilde{O}(k^2)$. Thus, the final returned flow has

- length at most $t \cdot O(1/\epsilon')^d \le t \cdot \exp(\mathrm{poly}(1/\epsilon))$,
- congestion at most $\gamma \cdot \tilde{O}(N^{\mathrm{poly}\epsilon'})^d \le \gamma \cdot N^{\mathrm{poly}\epsilon}$, and
- support size at most $\left(\mathrm{siz}_{d-1} + k^{d+1}\right) \cdot \tilde{O}(N^{\mathrm{poly}\epsilon'})^d \le (|E(R)| + |\mathrm{supp}(D)|)N^{\mathrm{poly}\epsilon}$.

Finally, we analyze the work and depth. By Lemma 7.9, the total work of the algorithm is

$$\mathrm{siz}_d \cdot \mathrm{poly}(t) = (\mathrm{siz}_0 + k^{d+1}) \cdot \tilde{O}(N^{\mathrm{poly}\epsilon'})^d \cdot \mathrm{poly}(t) \le (|E(R)| + |\mathrm{supp}(D)|) \cdot N^{\mathrm{poly}\epsilon} \cdot \mathrm{poly}(t)$$

and its depth is $\tilde{O}(k^3 \cdot N^{\mathrm{poly}\epsilon'} \cdot \mathrm{poly}(t)) \le N^{\mathrm{poly}\epsilon} \cdot \mathrm{poly}(t)$ (as the size of the recursive instance does not affect its depth, and the recursion depth is k). ∎

7.4 Routing on a Witness

Recall the definition of an expansion witness:

Definition 3.14 (Expansion Witness)**.** *Let G be a graph and A a node weighting. A $(h, t_{\mathcal{R}}, h_\Pi, \kappa_\Pi)$-*witness *of A in G is a tuple $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$.*

- *$\mathcal{N}$ is a neighborhood cover with covering radius h.*
- *$\mathcal{R}$ is a collection of routers. For each cluster $S \in \mathcal{N}$, there exists a $(t_{\mathcal{R}}, 1)$-router $R^S \in \mathcal{R}$ on vertex set S for node-weighting $S \cap A$.*
- *$\Pi_{\mathcal{R}\to G}$ is an embedding of all routers in $\mathcal{R}$ to G. $\Pi_{\mathcal{R}\to G}$ has length h_Π and congestion κ_Π.*

Our algorithm satisfying Corollary 7.2 is simple: for every demand pair, choose an arbitrary neighbourhood cover cluster containing both vertices of the pair. Then, call routing in a router within the embedded router of each cluster to route that cluster's demand.

Corollary 7.2 (Routing on a Witness)**.** *Given a $(h, t_{\mathcal{R}}, t_\Pi, \kappa_\Pi)$-witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$ of A in G, an A-respecting demand D such that for all $(a, b) \in \mathrm{supp}(D)$ there exists $S \in \mathcal{S} \in \mathcal{N}$ such that $a, b \in S$, and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, one can compute a flow F routing D with*

- *length $h_\Pi t_{\mathcal{R}} \lambda_{rr}(\epsilon))$ and congestion $\kappa_\Pi \kappa_{rr}(\epsilon)$, and*
- *support size $|\mathrm{supp}(F)| \le (|\mathrm{path}(\Pi_{\mathcal{R}\to G})| + |\mathrm{supp}(D)|)N^{\mathrm{poly}\epsilon}$,*

with $(|\mathrm{supp}(D)| + |\mathrm{path}(\Pi_{\mathcal{R}\to G})|) \cdot \mathrm{poly}(t_{\mathcal{R}})N^{\mathrm{poly}(\epsilon)}$ work and $\mathrm{poly}(t_{\mathcal{R}})N^{\mathrm{poly}(\epsilon)}$ depth.

Proof. First, to simplify the algorithm, for every edge e in a router R^S of a cluster $S \in \mathcal{S} \in \mathcal{R}$, we split the edge into multiple edges, each of which $\Pi_{\mathcal{R}\to G}$ maps to a path, not a flow. After this, the total size of the routers is $|\mathrm{path}(\Pi_{\mathcal{R}\to G})|$.

Now, let $D_S, S \in \mathcal{S} \in \mathcal{N}$ be demands such that D_S is restricted to cluster S, $D_S(a, b) \in D(a, b)$, and $\sum_S D_S = D$. For each cluster S, let F_S^{router} be a flow routing D_S on R^S computed by Theorem 7.1 with error parameter ϵ. Then, F_S^{router} has length $t_{\mathcal{R}} \cdot \lambda_{\mathrm{rr}}(\epsilon)$, congestion $1 \cdot \kappa_{\mathrm{rr}}(\epsilon)$ and support size $(|E(R^S)| + |\mathrm{supp}(D_S)|) \cdot N^{\mathrm{poly}\epsilon}$. This takes $(|E(R^S)| + |\mathrm{supp}(D_S)|) \cdot N^{\mathrm{poly}\epsilon} \cdot \mathrm{poly}(t_{\mathcal{R}})$ work and has depth $N^{\mathrm{poly}\epsilon} \cdot \mathrm{poly}(t_{\mathcal{R}})$; over all the clusters S, the total support size is $(|\mathrm{supp}(D)| + |\mathrm{path}(\Pi_{\mathcal{R}\to G})|) \cdot N^{\mathrm{poly}\epsilon}$ and the work is $(|\mathrm{supp}(D)| + |\mathrm{path}(\Pi_{\mathcal{R}\to G})|) \cdot N^{\mathrm{poly}\epsilon} \cdot \mathrm{poly}(t_{\mathcal{R}})$.

Next, we map the flows back to the original graph using $\Pi_{\mathcal{R}\to G}$. Let $F_S := \Pi_{\mathcal{R}\to G}(F_S^{\mathrm{router}})$ be the flow on the router mapped into G by the embedding of the expansion witness, and $F = \sum_S F_S$. Then, F has length $h_\Pi \cdot t_{\mathcal{R}} \cdot \lambda_{\mathrm{rr}}(\epsilon)$, congestion $\kappa_{\mathrm{rr}}(\epsilon) \cdot \kappa_\Pi$ and support size $|\mathrm{supp}(F)| = \sum_{S\in\mathcal{S}\in\mathcal{R}} |F_S^{\mathrm{router}}|$, as desired. The work and depth of this stage of the algorithm are negligible. □

8 Low-Step Multi-commodity Flow

In this section, we give $O(t)$-step k-commodity flow algorithms whose the running time are $|E| \cdot \mathrm{poly}(t)N^{\mathrm{poly}\epsilon}$. We emphasize that the running time is independent from k, in contrast the algorithms from [HHS23]. Later in Section 10, we will further remove the $\mathrm{poly}(t)$ dependency.

Below, a flow F is said to *partially route* a demand D if $D_F \le D$ (pointwise), where D_F is the demand routed by the flow F. The definition of the *total length* and *average length* of a flow are defined as follows:

$$\text{totlen}(F) := \sum_P F(P)\ell(P) = \sum_e F(e)\ell(e),$$
$$\text{avglen}(F) := \frac{\text{totlen}(F)}{\text{val}(F)}.$$

The following is the main result of this section. We can control both the maximum step t and total length bound T of the flow. The bound of the total length is very useful, as we will see in Section 9, as it us to "boost" the congestion slack κ to match the length slack $s \ll \kappa$.

Theorem 8.1 (Low-step Non-concurrent Flow)**.** *Let G be a graph with integral edge lengths $\ell \ge 1$ and capacities $u \ge 1$, D an integral demand, $t \ge 1$ a step bound, $T \ge 1$ a total length bound, and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c a tradeoff parameter. Then,* LowStepNonConcFlow(G, D, t, T, ϵ) *(Algorithm 6) returns a multicommodity flow F partially routing D, such that*

1. *F has maximum step length ts, total length Ts and congestion κ for step slack $s = \exp(\text{poly}1/\epsilon)$ and congestion slack $\kappa = N^{\text{poly}(\epsilon)}$. The support size of F is $(|E| + \text{supp}(D))N^{\text{poly}\epsilon}$.*
2. *Let F^* be the maximum-value multicommodity flow partially routing D of step length t, total length T and congestion 1. Then, $\text{val}(F) \ge \text{val}(F^*)$.*

The algorithm has depth $\text{poly}(t)N^{\text{poly}(\epsilon)}$ and work $(|E| + \text{supp}(D)) \cdot \text{poly}(t)N^{\text{poly}(\epsilon)}$.

The organization of this section is as follows. In Section 8.1, we first give a *weak* cutmatch algorithm, which we need later. In Section 8.2, we give an algorithm for computing multi-commodity flows with bounded maximum length. We will use this key subroutine to prove Theorem 8.1 in Section 8.3.

8.1 Weak Cutmatch for Many Commodities

For the flow algorithms, we will need a *weak* cutmatch algorithm WeakCutmatch. It has a similar guarantee as the cutmatch algorithm from [HHS23], but, in contrast to [HHS23], our running time is independent of the number of commodities. This comes at the cost of slack in both length and congestion and a weaker bound on the size $|C|$ of the cut: the cut has size at most a ϕ-fraction of the size of the total demand, instead of just the un-routed part of the demand.

Lemma 8.2. *Let G be a graph with integral edge lengths $\ell \ge 1$ and capacities $u \ge 1$, D an integral demand, $h \ge 1$ a maximum length bound, ϕ a sparsity parameter and $\epsilon \in (1/\log N, 1)$ a tradeoff parameter. Then,* WeakCutmatch$(G, D, h, \phi, \epsilon)$ *returns a multicommodity flow, h-length moving cut pair (F, C) such that*

1. *F partially routes D. Moreover, $\text{val}(F_{(a,b)}) \in \{0, D(a, b)\}$. That is, for every vertex pair, either none of the demand or all of the demand is routed by F.*

Algorithm 4 Weak Cutmatch: WeakCutmatch$(G, D, h, \phi, \epsilon)$

Input: Graph G with integral edge lengths $\ell \geq 1$ and capacities $u \geq 1$, integral demand D, maximum length bound $h \geq 1$, sparsity parameter ϕ, tradeoff parameter ϵ.

Output: A pair (F, C) of a multicommodity flow partially routing D and a h-length moving cut. It is guaranteed that F has length hs and congestion $\frac{\kappa}{\phi}$ for $s = \exp(\text{poly}(1/\epsilon))$ and $\kappa = N^{\text{poly}(\epsilon)}$, that $|C| \leq \phi \cdot |D|$, and that the demand between any vertices h-close in $G - C$ is routed by F.

1. Let $s' := \exp(\text{poly}(1/\epsilon))$ and $\kappa' := N^{\text{poly}(\epsilon)}$, with appropriate asymptotics.
2. Let C and $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to\mathcal{G}})$ be a pair of a h-length moving cut of size $\phi|D|$ and a $(h, \lceil 1/\epsilon \rceil, hs', 2\kappa'/\phi)$ expansion witness for $G - C$, computed by Theorem 3.18 on G and node weighting load(D) with parameters $(h, \phi/2, 0, \lceil 1/\epsilon \rceil)$.
3. Let $D'(a, b) = D(a, b)\mathbb{I}[\exists S \in \mathcal{S} \in \mathcal{N} : a, b \in S]$.
4. Let F be an flow routing D' of length hs'^3, congestion κ'^2 and support size $(|E| + |D'|)N^{\text{poly}\epsilon}$ computed by Corollary 7.2 on graph $G - C$, node weighting load(D) and witness $(\mathcal{N}, \mathcal{R}, \Pi_{\mathcal{R}\to G})$.
5. Return (F, C), with length slack $s = s'^3$ and congestion slack $\kappa = 2\kappa'^2$.

---

2. *F has length hs and congestion $\frac{\kappa}{\phi}$ for length slack $s = \exp(\text{poly}(1/\epsilon))$ and congestion slack $\kappa = N^{\text{poly}(\epsilon)}$. The support size of F is $(|E| + \text{supp}(D))N^{\text{poly}\epsilon}$.*
3. *For any (a, b), if $\text{dist}_{G-C}(a, b) \leq h$, then $\text{val}(F_{(a,b)}) = D(a, b)$. That is, F routes all the demand between any vertex pair not h-separated by the cut C.*
4. *C has size at most $\phi \cdot |D|$.*

The algorithm has depth $\text{poly}(h)N^{\text{poly}(\epsilon)}$ *and work* $(|E| + |\text{supp}(D)|) \cdot \text{poly}(h)N^{\text{poly}(\epsilon)}$.

The algorithm for Lemma 8.2 is simple. We first compute a witnessed length-constrained ϕ-expander decomposition C for load(D) of size $|C| \leq \phi|D|$. Since $G - C$ has an expansion witness, all demand pairs (a, b) that appear together in some cluster $S \in \mathcal{S} \in \mathcal{N}$ that are still close in $G - C$ can be fully routed using routing in a witnessed graph (Corollary 7.2) with a length-constrained multicommodity flow of congestion $\approx 1/\phi$.

Proof of Lemma 8.2. We show each of the claims.

- **Property 1.** Corollary 7.2 guarantees the returned flow routes exactly D'. As $D'(a, b) \in \{0, D(a, b)\}$, for every vertex pair, either none of the demand or all of the demand is routed.
- **Property 2.** The flow has length $hs = hs'^3$ and congestion $\kappa/\phi = 2\kappa'^2/\phi$ with appropriately chosen s' and κ' as Corollary 7.2 produces a flow of length $h \cdot t_{\mathcal{R}} t_{\Pi} \lambda_{rr}(\epsilon)$ and congestion $\kappa_{\Pi}\kappa_{rr}(\epsilon)$. For support size, Theorem 3.18 guarantees the embedding $\Pi_{\mathcal{R}\to G}$ has path count $|E|N^{\text{poly}(\epsilon)}$, thus the call to Corollary 7.2 produces a flow of support size $(|E|N^{\text{poly}(\epsilon)} + |\text{supp}(D)|)N^{\text{poly}(\epsilon)} = (|E| + |\text{supp}(D)|)N^{\text{poly}(\epsilon)}$.

- **Property 4.** If $\text{dist}_{G-C}(a,b) \le h$, then $a, b \in S$ for some $S \in \mathcal{S} \in \mathcal{N}$ as $\mathcal{N}$ is a neighbourhood cover of $G - C$ with covering radius h.
- **Property 3.** Theorem 3.18 guarantees that $|C| \le (\phi/2)|\text{load}(D)| = \phi|D|$.
- **Work and depth.** The algorithm's work consists of a call to Theorem 3.18 and a call to Corollary 7.2, both of which have depth at most $\text{poly}(h)N^{\text{poly}(\epsilon)}$ and work $(|E| + \text{supp}(D)) \cdot \text{poly}(h)N^{\text{poly}(\epsilon)}$, thus the algorithm has this work and depth.

□

8.2 Maximum Length-Constrained Non-Concurrent Flow

In this section, we give an algorithm for computing multi-commodity flows with bounded maximum length. When we use later it as a subroutine, we the input demand will be $\frac{1}{n^2}$-fractional. This is why the statement handles $\frac{1}{n^2}$-fractional demands instead of just integral demands.

Lemma 8.3. *Let G be a graph with integral edge lengths $\ell \ge 1$ and capacities $u \ge 1$, D a $\frac{1}{n^2}$-fractional demand, $h \ge 1$ a maximum length bound, and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c a tradeoff parameter. Then,* MaxLenNonConcFlow(G, D, h, ϵ) *(Algorithm 5) returns a $\frac{1}{n^2}$-fractional multicommodity flow F routing a subdemand D' of D such that*

1. *F has maximum length hs and congestion κ for $s = \exp(\text{poly}1/\epsilon)$ and $\kappa = N^{\text{poly}(\epsilon)}$. The support size of F is $(|E| + \text{supp}(D))N^{\text{poly}\epsilon}$.*
2. *Let F^* be the maximum-value multicommodity flow partially routing D of maximum length h and congestion 1. Then, $\text{val}(F) \ge \text{val}(F^*)$.*

The algorithm has depth $\text{poly}(h)N^{\text{poly}(\epsilon)}$ and work $(|E| + \text{supp}(D)) \cdot \text{poly}(h)N^{\text{poly}(\epsilon)}$.

Our high-level strategy is as follows. We try to repeatedly apply WeakCutmatch to send flow as much as possible. We will set its sparsity parameter to be small enough so that the cuts from WeakCutmatch have small total size. Since WeakCutmatch guarantees that the demands pairs that are not cut, i.e., those still close after applying the cut, must have been fully routed, this mean that we have send a lot of flow. The complete algorithm description of Lemma 8.3 to carry out this strategy is shown in Algorithm 5.

Algorithm Explanation. We motivate the algorithm in more details here. First, we scale up the demand and capacity of the input graph by $\gamma = n^2$ from the beginning to allow us to solely work on integral demands. Our returned flow F_{res} directly satisfies Property 1 as the sum of logarithmically many flows returned from WeakCutmatch plus a tiny, short flow. However, showing the second property, i.e. that $\text{val}(F_{\text{res}}) \ge \text{val}(F^*)$, is non-trivial.

The algorithm has two main loops. For the outer for-loop, we will make progress as follows. After the iteration i of the outer loop, let F_i^* denote the max-value flow with small maximum length and congestion partially routing remaining D'. We will route a flow F (and add it F_{res}) to with value $\text{val}(F) \ge \text{val}(F_i^*)/2$ as long as $\text{val}(F_i^*) \ge 2n^2$. This implies that, either either $\text{val}(F^*) - \text{val}(F_{\text{res}})$ halves, or we have $\text{val}(F^*) - \text{val}(F_{\text{res}}) \le 2n^2$ for every iteration. If the latter happens, then we will

Algorithm 5 Length-Bounded NC Multicommodity Flow: MaxLenNonConcFlow(G, D, h, ϵ)

Input: Graph G with integral edge lengths $\ell \geq 1$ and capacities $u \geq 1$, $\frac{1}{n^2}$-fractional demand D, maximum length bound $h \geq 1$, tradeoff parameter ϵ.

Output: A multicommodity flow F of length hs and congestion κ routing a $\frac{1}{n^2}$-fractional subdemand of D for length slack $s = \exp(\text{poly}1/\epsilon)$ and congestion slack $\kappa = N^{\text{poly}(\epsilon)}$. For every multicommodity flow F^* partially routing D of maximum length h and congestion 1, it is guaranteed that $\text{val}(F) \geq \text{val}(F^*)$.

1. Let $\gamma := n^2$ and $\kappa' := 16\gamma \cdot \lceil \log \gamma |D| \rceil$.
2. Let $F_{\text{res}} \leftarrow 0$, $D' \leftarrow \gamma D$, and G' be G with capacities γu.
3. For $i \in \{1, 2, \ldots, \lceil \log |D| \rceil\}$:
   (a) Let $C \leftarrow 0$, $F \leftarrow 0$ and $S_{\text{close}} \leftarrow \text{supp}(D')$.
   (b) For $p \in \{1, 2, \ldots, \lceil \log \gamma |D| \rceil\}$:
      i. Let $D'_{\text{cap}}(a, b) := \min(D'(a, b), 2^p) \cdot \mathbb{I}[(a, b) \in S_{\text{close}}]$.
      ii. Let $(C', F') = \text{WeakCutmatch}(G' - C, D'_{\text{cap}}, 2h, 1/\kappa', \epsilon)$.
      iii. Set $C \leftarrow C + C'$ and $F \leftarrow F + F'$.
      iv. Set $D'(a, b) \leftarrow D'(a, b) - \text{val}(F'_{(a,b)})$ for each $(a, b) \in \text{supp}(D')$.
      v. Set $S_{\text{close}} \leftarrow S_{\text{close}} \cap \{(a, b) : \text{val}(F'_{(a,b)}) > 0\}$
   (c) $F_{\text{res}} \leftarrow F_{\text{res}} + F$.
4. Let $F'_{\text{final}} = \text{WeakCutmatch}(G' - C, D', 2h, 1/N^2, \epsilon)$.
5. Let F_{final} be a subflow of F'_{final} with integral $D_{F_{\text{final}}}$ of value $\text{val}(F_{\text{final}}) = \min(\text{val}(F'_{\text{final}}), 2n^2)$ routing a subdemand of D'.
6. Let $F_{\text{res}} \leftarrow F_{\text{res}} + F_{\text{final}}$.
7. Return $\frac{1}{\gamma} F_{\text{res}}$.

add F_{final} to F_{res} at the end of value $\min(\text{val}(F^*) - \text{val}(F_{\text{res}}), 2n^2)$. Note that F_{final} can be computed trivially as we do not need to worry about the congestion as we have scaled up the capacity of the graph by n^2 from the beginning.

The inner for-loop tries to construct F where $\text{val}(F) \geq \text{val}(F_i^*)/2$ assuming $\text{val}(F_i^*) \geq 2n^2$. Our strategy is to maintain a subdemand D'_{cap} of D' such that $|D'_{cap}| \leq 2\text{val}(F_i^*)$. Our definition of D'_{cap} satisfies this when $p = 1$ since $\text{val}(F_i^*) \geq 2n^2$. Also, $|D'_{cap}|$ can grow by only a factor of 2 per iteration of the inner loop.

We will use WeakCutmatch to cut or route D'_{cap}. The interesting case is when $\text{val}(F_i^*) \leq |D'_{cap}| \leq 2\text{val}(F_i^*)$. If WeakCutmatch routes more than half of D'_{cap}, then we have routed at least $\text{val}(F_i^*)/2$ and we have achieved the goal. Otherwise, WeakCutmatch cuts/separates more than half of D'_{cap} which reduces $|D'_{cap}|$ in the next iteration and maintain the invariant $|D'_{cap}| \leq 2\text{val}(F_i^*)$.

The bound of $|D'_{cap}|$ is useful because it means that each cut from WeakCutmatch has size

bounded by $\phi|D'_{cap}| = O(\phi \text{val}(F_i^*))$ where ϕ is a parameter we can choose. So the total size size after $\tilde{O}(1)$ iterations is small compared to $\text{val}(F_i^*)$. So all cuts found did not "separates" too many demand pairs routed by F_i^*. Since WeakCutmatch guarantees that the demand pairs that are not separated must be are completely routed. This implies that the total flow we have routed during the inner for loop is at least $\text{val}(F_i^*)/2$ again.

For the proof of the theorem, we will use the following lemmas:

Lemma 8.4. *Let G be a graph with edge capacities u and lengths ℓ. Let F be a h-length multi-commodity flow and C a $2h$-length moving cut. Let F' be the subflow of F that is $2h$-separated in $G - C$, i.e., for all paths P, $F'(P) := F(P)\mathbb{I}[l_{G-C}(P) > 2h]$. Then,*

$$\text{val}(F') \leq 2|C| \cdot \text{cong}_F.$$

Proof. Let L be the total length of the flow F in G. The total length of F in $G - C$ is at least $L + h \cdot \text{val}(F')$, as the length of the subflow F' has increased from at most h to at least $2h$. On the other hand, the total length of F in $G - C$ is at most $L + \sum_e F(e) \cdot 2h \cdot C(e) \leq L + \text{cong}_F \cdot 2h \sum_e u(e)C(e) = L + \text{cong}_F \cdot 2h|C|$. This implies that gives

$$L + h \cdot \text{val}(F') \leq \sum_P F(P)\ell_{G-C}(P) \leq L + 2h \cdot |C| \cdot \text{cong}_F.$$

Cancelling L and dividing by h gives $\text{val}(F') \leq 2|C| \cdot \text{cong}_F$, as desired. □

Lemma 8.5. *Let G' be a n-vertex graph with edge capacities $u \geq n^2$ and lengths ℓ, h a maximum length bound, $\gamma \geq 1$ a congestion bound and D an integral demand. Then, there exists a flow F^* of maximum length h and congestion 2γ that routes an **integral** subdemand of D, such that any flow F^{**} of maximum length h and congestion γ that routes a subdemand of D satisfies $\text{val}(F^*) \geq \text{val}(F^{**})$.*

Proof. Let F^{**} be the maximum-value flow of length h and congestion γ routing a subdemand of D. Let F^* be the flow created by rounding up the flow value between every vertex pair up to the next integer. This increases the total flow by at most n^2, and thus the total flow over any edge by at most $|\text{supp}(D)| \leq n^2$. Thus, as the capacity of every edge is at least n^2, the congestion goes up by at most $1 \leq \gamma$, while the flow value does not decrease. The routed demand still remains a subdemand of D, as D is integral. □

Lemma 8.6. *At the end of each iteration of the for-loop on line 3, for every (a, b), either $D'(a, b) = 0$ (the flow fully satisfies the (a, b)-demand) or $\text{dist}_{G'-C}(a, b) > 2h$ (the pair is $2h$-separated).*

Proof. Consider the iteration of the for-loop on line 3b where $p = \lceil \log \gamma|D| \rceil$. Then, $2^p \geq \gamma|D|$, and in particular $D'(a, b) \leq 2^p$ holds for every vertex pair (a, b). There are four possible situations before the updates to C, F, D' and S_{close} of the iteration:

- $D'(a, b) = 0$,
- $(a, b) \in S_{\text{close}}$ and $\text{val}(F'_{(a,b)}) = D'_{\text{cap}}(a, b)$,
- $(a, b) \in S_{\text{close}}$ and $\text{val}(F'_{(a,b)}) = 0$, and

- $(a,b) \notin S_{\text{close}}$.

In the first case we are trivially done. In the second case, we are done as $D'(a,b) = D'_{\text{cap}}(a,b)$. In the third case, since the cutmatch algorithm guarantees that for (a,b) with $\text{dist}_{G'-C}(a,b) \leq 2h$ we have $\text{val}(F'_{(a,b)}) = D'_{\text{cap}}(a,b)$, the pair must be $2h$-separated. Similarly, for (a,b) to have left S_{close} in a previous iteration, their distance must have been greater than $2h$. Thus, as distances cannot decrease, their distance remains greater than $2h$. □

We are ready to prove Lemma 8.3.

Proof. The work and depth bounds are clear, as the algorithm's work and depth are dominated by $\tilde{O}(1)$ calls to Algorithm 4.

Property 1. The cutmatch algorithm only produces integral flows, thus the returned flow is $\frac{1}{n^2}$-fractional. The flow F_{res} before adding F_{final} is the sum of $\tilde{O}(1)$ flows, each produced by Algorithm 4 with maximum length bound $2h$ and sparsity parameter $\frac{1}{\kappa'} = O(1/\gamma \log N)$. Algorithm 4 thus guarantees that the flow has length $2h \cdot s'' = h \cdot \exp(\text{poly}1/\epsilon)$ and congestion $\kappa'\kappa'' = \gamma N^{\text{poly}(\epsilon)}$, for $s'' = \exp(\text{poly}1/\epsilon)$ and $\kappa'' = N^{\text{poly}(\epsilon)}$. The flow F_{final} has length $2h \cdot s''$ and congestion at most its value of 2γ, thus adding it to F_{res} does not change the asymptotic length and congestion, and the returned flow has congestion $N^{\text{poly}(\epsilon)}$, as desired. Finally, the support size is $(|E| + \text{supp}(D))N^{\text{poly}\epsilon}$ as F is the sum of $\tilde{O}(1)$ flows of support size $(|E| + \text{supp}(D))N^{\text{poly}\epsilon}$.

Property 2. The algorithm starts by scaling up all capacities and demands by $\gamma = n^2$. Then, $u \geq n^2$, thus by Lemma 8.5, the maximum-value length-h congestion-2γ flow F^* routing an *integral* subdemand of γD has value at least the value of any maximum-length-h congestion-γ flow routing a possibly fractional subdemand of γD. It thus suffices to show that $\text{val}(F_{\text{res}}) \geq \text{val}(F^*)$ at the end of the algorithm.

Fix a iteration i of the outer for-loop on line 3. Let D'_i be the remaining demand D' at the start of iteration i. Let F^*_i be the maximum-value length-h and congestion-2γ flow routing an integral subdemand of D'_i. Consider the demand $D^{\Delta}_i = \min\{D'_i, D_{F^*}\}$ where D_{F^*} is routed by F^*. Since D^{Δ}_i is a subdemand of D_{F^*}, it is routable by a length-h congestion-2γ flow. So $\text{val}(F^*_i) \geq |D^{\Delta}_i| \geq \text{val}(F^*) - \text{val}(F_{res})$ for F_{res} at the beginning of iteration i.

If $\text{val}(F^*_i) \geq 2n^2$, we will show that the flow F constructed at the end of iteration i satisfies $\text{val}(F) \geq \text{val}(F^*_i)/2$. Therefore, at the end of iteration i when we set $F_{\text{res}} \leftarrow F_{\text{res}} + F$, either $\text{val}(F^*) - \text{val}(F_{\text{res}})$ halves, or we have $\text{val}(F^*) - \text{val}(F_{\text{res}}) \leq 2n^2$. As the difference is initially at most $\gamma|D| = n^2|D|$ and at the end of the algorithm we add a flow F_{final} of value $2n^2$ to the returned flow, we have that after $\lceil \log |D| \rceil$ iterations and after line 6, we have $\text{val}(F_{\text{res}}) \geq \text{val}(F^*)$.

Now, assume that $\text{val}(F^*_i) \geq 2n^2$. We show that the flow F constructed satisfies $\text{val}(F) \geq \text{val}(F^*_i)/2$. To show this, by Lemma 8.6, we have that the produced cut C $2h$-separates all of the demand not routed by F. Thus, it is sufficient to show that at most half of F^*_i is $2h$-separated by C.

By Lemma 8.4, any length-$2h$ cut C $2h$-separates at most $4\gamma|C|$ of F^*_i, as F^*_i has congestion at most 2γ. Thus, as long as $|C| \leq \text{val}(F^*_i)/8\gamma$, C separates at most half of F^*_i. The size of the

length-$2h$ cut C' produced on line 3(b)ii of the algorithm is at most

$$\frac{1}{\kappa'}|D'_{\text{cap}}| = \frac{1}{8\gamma}\frac{1}{\lceil \log \gamma |D| \rceil} \cdot \left(\frac{1}{2}|D'_{\text{cap}}|\right),$$

by the guarantee of WeakCutmatch from Lemma 8.2. Thus, assuming that $|D'_{\text{cap}}| \leq 2\text{val}(F_i^*)$ at every point during iteration i, we have that the size of the final length-$2h$ cut C produced is at most $\frac{\text{val}(F_i^*)}{8\gamma}$. So, at most half of F_i^* can be $2h$-separated by C, as desired.

Next, we prove the following result, which gives either $|D'_{\text{cap}}| \leq 2\text{val}(F_i^*)$ at every point during iteration i or directly that $\text{val}(F) \geq \frac{1}{2}\text{val}(F_i^*)$, by induction: for every $p \in \{0, 1, \ldots, \lceil \log \gamma |D| \rceil\}$, either $\text{val}(F) \geq \frac{1}{2}\text{val}(F_i^*)$, or $|D'_{\text{cap},p+1}| \leq 2\text{val}(F_i^*)$, where $D'_{\text{cap},p}$ is the demand defined at line 3(b)i in the iteration of the loop on line 3b for a particular p.

The base case follows from $|D'_{\text{cap},p}| \leq 2^p|\text{supp}(D')| \leq 2n^2 \leq 2\text{val}(F_i^*)$ when $p = 1$. Assume the claim holds for $p' < p$. If for the previous iteration $\text{val}(F) \geq \frac{1}{2}\text{val}(F_i^*)$, we are done, as the value of F cannot decrease and the value of F_i^* is invariant. Thus, we may assume that the other condition holds.

Observe that $|D'_{\text{cap},p+1}| \leq 2|D'_{\text{cap},p}|$. Thus, if $|D'_{\text{cap},p}| \leq \text{val}(F_i^*)$, we are done. Assume the contrary, and consider the pair (C', F') returned by WeakCutmatch. By Lemma 8.2, either the moving cut C' drops pairs from S_{close} contributing at least a $\frac{1}{2}$-fraction of $|D'_{\text{cap},p}|$ or the flow F' has value $\text{val}(F') \geq \frac{1}{2}|D'_{\text{cap},p}|$. In the latter case, we are done, as now $\text{val}(F) \geq \text{val}(F') \geq \frac{1}{2}|D'_{\text{cap},p}| \geq \frac{1}{2}\text{val}(F_i^*)$. In the former case, we are done, as the demand pairs dropped from S_{close} will not contribute to $|D'_{\text{cap},p}|$ the next iteration and all iterations after, and the demand value for the other pairs is at most doubled, and thus $|D'_{\text{cap},p+1}| \leq 2(\frac{1}{2}|D'_{\text{cap},p}|) \leq 2\text{val}(F_i^*)$. □

8.3 Low-Step Total-Length-Constrained Non-Concurrent Flow

In this section, we prove Theorem 8.1. The algorithm needs to round edge lengths to go from a step bound and a length bound to just a length bound. The following fact shows the correctness of the approach:

Fact 8.7. *Let G be a graph with positive edge lengths ℓ, h be a length bound, and t be a step bound. Define the length function $\ell'(e) = \left\lceil \frac{t \cdot \ell(e)}{h} \right\rceil$. Then, for any path p, we have*

$$\max\left(|p|, \frac{t \cdot \ell(p)}{h}\right) \leq \ell'(p) \leq |p| + \frac{t \cdot \ell(p)}{h}.$$

In particular,

- *if $\ell(p) \leq h$ and $|p| \leq t$, then $\ell'(p) \leq 2t$, and*
- *if $\ell'(p) \leq 2t$, then $\ell(p) \leq 2h$ and $|p| \leq 2t$.*

Proof. We have $\ell'(p) = \sum_{e \in p} \left\lceil \frac{t \cdot \ell(e)}{h} \right\rceil$. Clearly, $\max\left(|p|, \frac{t \cdot \ell(p)}{h}\right) \leq \sum_{e \in p} \left\lceil \frac{t \cdot \ell(e)}{h} \right\rceil \leq |p| + \frac{t \cdot \ell(p)}{h}$. □

Now, we describe the algorithm of Theorem 8.1 in Algorithm 6.

Algorithm 6 Low-Step Non-Concurrent Flow: LowStepNonConcFlow(G, D, t, T, ϵ)

Input: Graph G with integral edge lengths $\ell \geq 1$ and capacities $u \geq 1$, integral demand D, step bound $t \geq 1$, total length bound $T \geq 1$, tradeoff parameter ϵ.

Output: A multicommodity flow F partially routing D of step length ts, total length Ts and congestion κ for $s = \exp(\text{poly}1/\epsilon)$ and $\kappa = N^{\text{poly}(\epsilon)}$. For every multicommodity flow F^* partially routing D of step length t, total length T and congestion 1, it is guaranteed that $\text{val}(F) \geq \text{val}(F^*)$.

1. Let $s' = \exp(\text{poly}(1/\epsilon))$ be the length slack of Algorithm 5 on tradeoff parameter ϵ.
2. Let $T' = 5s'T$.
3. Let $F \leftarrow 0$ and $D' \leftarrow D$.
4. For $p \in \{0, 1, \ldots, \lceil \log nN \rceil\}$:
   (a) Let $h = 2^p$.
   (b) Let $\ell'(e) = \left\lceil \frac{t \cdot \ell(e)}{h} \right\rceil$ and G' be G with edge lengths ℓ'.
   (c) Let $F^{\text{aug}} = \text{MaxLenNonConcFlow}(G', D', 2t, \epsilon)$.
   (d) Let λ be the maximum value in $[0, 1]$ such that $\text{totlen}(F + \lambda F^{\text{aug}}) \leq T'$.
   (e) Set $F \leftarrow F + \lambda F^{\text{aug}}$.
   (f) If $\lambda < 1$, return F.
   (g) For all $(a, b) \in \text{supp}(D')$, set $D'(a, b) \leftarrow D'(a, b) - \text{val}(F^{\text{aug}}_{(a,b)})$.
5. Return F.

Proof. The work and depth are dominated by the calls to MaxLenNonConcFlow, of which there are $\tilde{O}(1)$-many. Note that since flow returned by MaxLenNonConcFlow routes a $\frac{1}{n^2}$-fractional subdemand, D' is always $\frac{1}{n^2}$-fractional and is a valid input for MaxLenNonConcFlow. Now, we prove the two properties of the returned flow F.

Property 1. Let $s = \exp(\text{poly}1/\epsilon)$ and $\kappa = N^{\text{poly}(\epsilon)}$ be the length slack and congestion slack respectively of MaxLenNonConcFlow from Lemma 8.3. By Fact 8.7, the step bound of F is at most $2ts'$ because F has maximum ℓ'-length at most $2ts'$. The total length bound of F is at most $T' = 3s'b$ as explicitly enforced by line 4d. The flow F has congestion $\kappa\lceil \log nN \rceil$ as there are at most $1 + \lceil \log nN \rceil$ iterations. The support size bound $\text{supp}(F) = (|E| + \text{supp}(D))N^{\text{poly}\epsilon}$ follows directly from Lemma 8.3.

Property 2. Let F^* be the maximum-value multicommodity flow partially routing D using step t, total length T and congestion 1. Our goal is to show that $\text{val}(F) \geq \text{val}(F^*)$.

For $p \in \{0, 1, \ldots, \lceil \log nN \rceil\}$, let F^*_p be the sub-flow of F^* with path lengths in ℓ at most 2^p. Note that $F^* = F^*_{\lceil \log nN \rceil}$ because simple paths have length at most nN as $\ell(e) \leq N$. Let D^*_p be the demand routed by F^*_p. Let $F^{*\text{aug}}_p$ be the flow that augment F^*_{p-1} to F^*_p, i.e., $F^*_p = F^*_{p-1} + F^{*\text{aug}}_p$. Let F^{aug}_p be the flow produced by Algorithm 5 from Lemma 8.3 on line 4c when $h = 2^p$. Let $F_p = F_{p-1} + F^{\text{aug}}_p$ where $F_{-1} = 0$. That is, F^{aug}_p augments F_{p-1} to F_p.

First, we show that $\mathrm{val}(F_p) \geq \mathrm{val}(F_p^*)$ for all p. Consider $D'_p = D' - D_{F_{p-1}}$. That is, D'_p is the remaining demand D' at the start of the iteration of the loop when $h = 2^p$. Consider the demand $D_p^\Delta = \min(D'_p, D_p^*)$. By the definition of F_p^*, there is a step-t ℓ-length-h congestion-1 flow routing D_p^Δ. So, D_p^Δ can be routed by a flow with ℓ'-length $2t$ and congestion 1 by Fact 8.7. Therefore, by the guarantee of MaxLenNonConcFlow from Lemma 8.3, the flow F_p^{aug} produced on line 4c has value at least $|D_p^\Delta| \geq \mathrm{val}(F_p^*) - \mathrm{val}(F_{p-1})$. Hence, we have $\mathrm{val}(F_p) = \mathrm{val}(F_p^{\mathrm{aug}}) + \mathrm{val}(F_{p-1}) \geq \mathrm{val}(F_p^*)$.

Suppose F is returned on line 5. Then, $F = F_{\lceil \log nN \rceil}$ and so we have $\mathrm{val}(F) = \mathrm{val}(F_{\lceil \log nN \rceil}) \geq \mathrm{val}(F^*_{\lceil \log nN \rceil}) = \mathrm{val}(F^*)$.

From now, we assume that F is returned on line 4f at the iteration p. Observe that $F = F_{p-1} + \lambda F_p^{\mathrm{aug}}$ and $\mathrm{totlen}(F) = T'$. The key claim is the following, which will be proved at the end.

Claim 8.8. *For all p, there is a subflow $\widehat{F}_p$ of F_p such that $\mathrm{val}(\widehat{F}_p) = \mathrm{val}(F_p^*)$ and $\mathrm{totlen}(\widehat{F}_p) \leq 4s' \cdot \mathrm{totlen}(F_p^*)$.*

We will use the above claim only for F_{p-1}. Now, suppose for contradiction that $\mathrm{val}(F) < \mathrm{val}(F^*)$. We will analyze $\mathrm{totlen}(F^*) = \mathrm{totlen}(F^* - F^*_{p-1}) + \mathrm{totlen}(F^*_{p-1})$. Let us analyze the two term as follows. First, we have

$$\begin{aligned}
\mathrm{totlen}(F^* - F^*_{p-1}) &= \mathrm{avglen}(F^* - F^*_{p-1})(\mathrm{val}(F^*) - \mathrm{val}(F^*_{p-1})) \\
&\geq \frac{1}{4s'}\mathrm{avglen}(F - \widehat{F}_{p-1})(\mathrm{val}(F^*) - \mathrm{val}(F) + \mathrm{val}(F) - \mathrm{val}(\widehat{F}_{p-1})) \\
&= \frac{1}{4s'}\left(\mathrm{avglen}(F - \widehat{F}_p)(\mathrm{val}(F^*) - \mathrm{val}(F)) + \mathrm{totlen}(F - \widehat{F}_{p-1})\right) \\
&> \frac{1}{4s'}\mathrm{totlen}(F - \widehat{F}_{p-1})
\end{aligned}$$

where the first inequality follows from (1) the minimum ℓ-length of $F^* - F_p^*$ is at least 2^{p-1}, (2) the maximum ℓ-length of $F - \widehat{F}_{p-1}$ is at most $2^{p+1}s'$ because the maximum ℓ'-length of F is $2ts'$ and by Fact 8.7, and (3) $\mathrm{val}(\widehat{F}_{p-1}) = \mathrm{val}(F^*_{p-1})$ by Claim 8.8. The last inequality is by our assumption that $\mathrm{val}(F^*) - \mathrm{val}(F) > 0$. Second, by Claim 8.8, we directly have

$$\mathrm{totlen}(F^*_{p-1}) \geq \frac{1}{4s'} \cdot \mathrm{totlen}(\widehat{F}_{p-1}).$$

Combining the two inequalities, we get a contradiction because

$$\begin{aligned}
T &\geq \mathrm{totlen}(F^*) \\
&= \mathrm{totlen}(F^* - F_p^*) + \mathrm{totlen}(F_p^*) \\
&> \mathrm{totlen}(F)/4s' \\
&> T
\end{aligned}$$

where the last equality is because $\mathrm{totlen}(F) = T' = 5s'T$. This concludes the proof that $\mathrm{val}(F) \geq \mathrm{val}(F^*)$ when returned on line 4f. □

Finally, we prove Claim 8.8.

Proof of Claim 8.8. We prove by induction. The base case when $p = -1$ is trivial because $\mathrm{val}(F^*_{-1}) = 0$ as every edge has length at least 1 and $F_{-1} = 0$. Now, assume the claim holds for all $p' < p$.

Since $\mathrm{val}(F_p) \geq \mathrm{val}(F^*_p)$ and $\mathrm{val}(\widehat{F}_{p-1}) = \mathrm{val}(F^*_{p-1})$ by induction, we have that $\mathrm{val}(F_p - \widehat{F}_{p-1}) \geq \mathrm{val}(F^{*\mathrm{aug}}_p)$.[3] So, there exists $\gamma \in [0,1]$ such that $\gamma\mathrm{val}(F_p - \widehat{F}_{p-1}) = \mathrm{val}(F^{*\mathrm{aug}}_p)$. We define $\widehat{F}^{\mathrm{aug}}_p = \gamma(F_p - \widehat{F}_{p-1})$ and set $\widehat{F}_p = \widehat{F}_{p-1} + \widehat{F}^{\mathrm{aug}}_p$.

Let us verify that $\widehat{F}_p$ satisfies the two properties. For the first property, we have

$$\mathrm{val}(\widehat{F}_p) = \mathrm{val}(\widehat{F}_{p-1}) + \mathrm{val}(\widehat{F}^{\mathrm{aug}}_p) = \mathrm{val}(F^*_{p-1}) + \mathrm{val}(F^{*\mathrm{aug}}_p) = \mathrm{val}(F^*_p)$$

by induction and by the definition of $\widehat{F}^{\mathrm{aug}}_p$. For the second property, observe that maximum ℓ'-length of $\widehat{F}^{\mathrm{aug}}_p$, which is a subflow of F_p, is at most $2ts'$, and so the maximum ℓ-length of $\widehat{F}^{\mathrm{aug}}_p$ is at most $(2ts') \cdot h/t = 2^{p+1}s'$ by Fact 8.7. But the minimum ℓ-length of $F^{*\mathrm{aug}}_p$ is at least 2^{p-1}. Hence, $\mathrm{avglen}(\widehat{F}^{\mathrm{aug}}_p) \leq 4s'\mathrm{avglen}(F^{*\mathrm{aug}}_p)$ and so $\mathrm{totlen}(\widehat{F}^{\mathrm{aug}}_p) \leq 4s'\mathrm{totlen}(F^{*\mathrm{aug}}_p)$ because the value of the two flows are same. Thus, by induction, we get

$$\mathrm{totlen}(\widehat{F}_p) = \mathrm{totlen}(\widehat{F}_{p-1}) + \mathrm{totlen}(\widehat{F}^{\mathrm{aug}}_p) \leq 4s' \cdot \mathrm{totlen}(F^*_{p-1}) + 4s' \cdot \mathrm{totlen}(F^{*\mathrm{aug}}_p) = 4s' \cdot \mathrm{totlen}(F^*_p).$$

This completes the inductive step of the claim. □

9 Flow Boosting

In this section, we show how to *boost* a flow algorithm that achieves length slack s and congestion slack κ to an algorithm that achieves length slack s and congestion slack $(1+\epsilon)s$ with an additional running time overhead of $\mathrm{poly}(\kappa/\epsilon)$. Since we are primarily interested in the regime when $s = \exp(\mathrm{poly}(1/\epsilon))$ and $\kappa = n^{\mathrm{poly}\epsilon}$, boosting effectively reduces the congestion slack down to the length slack.

In order to bypass the $O(mk)$ flow-path decomposition barrier, our algorithms must output an *implicit* flow, which we formalize as a flow *oracle*.

Definition 9.1 (Flow Oracle)**.** *A flow oracle $\mathcal{O}_F$ for a multi-commodity flow F on a graph G is a data structure supporting the following query:*

- *Given a subset S of pairs of vertices of G, return the edge representation flow_{F_S} of F_S, where $F_S := \sum_{(a,b)\in S} F_{(a,b)}$ is the subflow of F between the vertex pairs $(a,b) \in S$.*

The oracle has query work Q_w and query depth Q_d if every query S takes at most Q_w work and has depth at most Q_d.

9.1 Flow Boosting Template

We begin with a generic flow boosting *template* that does not depend on the specifics of the flow problem, and instead works for any convex set $\mathcal{F}$ of satisfying flows. For illustration, the reader can imagine that $\mathcal{F}$ is the set of concurrent or non-concurrent flows for a given demand.

[3] Note that $F_p - \widehat{F}_{p-1}$ is well-defined because $\widehat{F}_{p-1}$ is a subflow of F_{p-1} which is a subflow of F_p.

Theorem 9.2 (Flow Boosting Template)**.** *Let $G = (V, E, u, b)$ be a graph with capacity function u and cost function b, and let $B \geq 0$ be the cost budget. Let $\mathcal{F}$ be any convex set of flows in G containing at least one capacity-respecting flow, and let $\epsilon, s, \kappa \geq 0$ be parameters. Suppose an algorithm is given an oracle $\mathcal{O}$ that, given any integral edge length function $\ell : E \to \{1, 2, \ldots, O(m^{1/\epsilon}N/\epsilon)\}$, computes the edge representation of a (not necessarily capacity-respecting) flow $F \in \mathcal{F}$ such that*

- Length slack s: $\sum_{e\in E} F(e) \cdot \ell(e) \leq s \cdot \sum_{e\in E} F^*(e) \cdot \ell(e)$ *for any flow $F^* \in \mathcal{F}$ that is also capacity-respecting, and*
- Congestion slack κ: $F(e) \leq \kappa u(e)$ *for all $e \in E$, i.e., $F(e)/\kappa$ is capacity-respecting.*

Then, there is a deterministic algorithm that makes $O(\kappa\epsilon^{-2} \log^2 n)$ calls to oracle $\mathcal{O}$ and outputs the edge representation of a flow $\bar{F} \in \mathcal{F}$ and scalar $\lambda \geq 0$ such that

1. Feasibility: *The flow $\lambda\bar{F}$ is capacity-respecting with cost at most B, and*
2. Approximation factor $\approx s$: *Let λ^* be the maximum value such that there exists flow $F^* \in \mathcal{F}$ where $\lambda^* F^*$ is capacity-respecting with cost at most B. Then, $\lambda \geq \frac{1-O(\epsilon)}{s}\lambda^*$.*

Moreover, the flow $\bar{F}$ is a convex combination of the flows returned by oracle $\mathcal{O}$, and this convex combination can be output as well.

Furthermore, if the oracle $\mathcal{O}$ also outputs a flow oracle with query work Q_w and query depth Q_d, then the algorithm can also output a flow oracle for $\bar{F}$ with query work $\tilde{O}(\kappa\epsilon^{-2}Q_w)$ and query depth $\tilde{O}(Q_d)$.

The algorithm takes $\tilde{O}(\kappa\epsilon^{-2}m)$ work and $\tilde{O}(\kappa\epsilon^{-2})$ time outside of the oracle calls.

For the rest of this subsection, we prove Theorem 9.2. The proof closely follows Sections 5 and 6 of [GK07], so we claim no novelty here. We first impose the assumption that $\lambda^* \geq 1$ for λ^* as defined in Condition 2 of Theorem 9.2.

Let $\mathcal{K}$ be the set of capacity-respecting flows in G, and consider the following flow LP of the graph G. We have a variable $x(F) \geq 0$ for each $F \in \mathcal{F} \cap \mathcal{K}$ indicating that we send flow F scaled by $x(F)$. To avoid clutter, we also define $b(F) = \sum_{e\in E} F(e) \cdot b(e)$ as the cost of the flow F.

$$\begin{aligned}
\max \quad & \sum_{F\in\mathcal{F}\cap\mathcal{K}} x(F) \\
\text{s.t.} \quad & \sum_{F\in\mathcal{F}\cap\mathcal{K}} F(e) \cdot x(F) \leq u(e) && \forall e \in E \\
& \sum_{F\in\mathcal{F}\cap\mathcal{K}} b(F) \cdot x(F) \leq B \\
& x \geq 0
\end{aligned}$$

Let β be the optimal value of this LP. Note that since $\mathcal{F} \cap \mathcal{K}$ is convex, there is an optimal solution with $x(F^*) = \beta$ for some $F^* \in \mathcal{F}$ and $x(F) = 0$ elsewhere. It follows that $\beta = \lambda^*$.

The dual LP has a length $\ell(e) \geq 0$ for each edge $e \in E$ as well as a length $\phi \geq 0$ of the cost

constraint.

$$\begin{aligned} \min \quad & \sum_{e\in E} u(e)\cdot \ell(e) + B\cdot\phi && =: D(\ell,\phi) \\ \text{s.t.} \quad & \sum_{e\in E} F(e)\cdot(\ell(e)+b(e)\phi) \geq 1 && \forall F\in\mathcal{F}\cap\mathcal{K} \\ & \ell\geq 0,\ \phi\geq 0 \end{aligned}$$

Let $D(\ell,\phi)=\sum_{e\in E} u(e)\ell(e)+B\cdot\phi$ be the objective value of the dual LP. Define $\alpha(\ell,\phi)$ as the minimum length of a flow $F\in\mathcal{F}\cap\mathcal{K}$ under length function $\ell+\phi b$:

$$\alpha(\ell,\phi)=\min_{F\in\mathcal{F}\cap\mathcal{K}}\sum_{e\in E} F(e)\cdot(\ell(e)+b(e)\phi).$$

Then by scaling, we can restate the dual LP as finding a length function ℓ minimizing $D(\ell,\phi)/\alpha(\ell,\phi)$. By LP duality, the minimum is β, the optimal value of the primal LP, which we recall also equals $\lambda^*\geq 1$.

The algorithm initializes length functions $\ell^{(0)}(e)=\delta/u(e)$ and $\phi^{(0)}=\delta/B$ for parameter $\delta=m^{-1/\epsilon}$, and proceeds for a number of iterations. For each iteration i, the algorithm wishes to call the oracle $\mathcal{O}$ on length function $\ell^{(i-1)}+\phi^{(i-1)}b$, but the length function $\ell^{(i-1)}+\phi^{(i-1)}b$ is not integral. However, we will ensure that they are always in the range $[\delta/N, O(1)]$. So the algorithm first multiplies each length by $O(N/(\delta\epsilon))=O(m^{1/\epsilon}N/\epsilon)$ and then rounds the weights to integers so that each length is scaled by roughly the same factor up to $(1+\epsilon)$. The algorithm calls the oracle on these scaled, integral weights to obtain a flow $F^{(i)}$. On the original, unscaled graph, the flow satisfies the following two properties:

1. Length slack $(1+\epsilon)s$: $\sum_{e\in E} F^{(i)}(e)\cdot(\ell^{(i-1)}(e)+b(e)\phi^{(i-1)})\leq(1+\epsilon)s\cdot\alpha(\ell^{(i-1)},\phi^{(i-1)})$, and
2. Congestion slack κ: $F(e)\leq\kappa u(e)$ for all $e\in E$, i.e., $F/\kappa\in\mathcal{K}$.

Define $z^{(i)}=\min\{1,\ B/b(F^{(i)})\}$ so that $b(z^{(i)}F^{(i)})\leq B$, i.e., the cost of the scaled flow $z^{(i)}F^{(i)}$ is within the budget B. The lengths are then modified as

$$\ell^{(i)}(e)=\ell^{(i-1)}(e)\left(1+\frac{\epsilon}{\kappa}\cdot\frac{z^{(i)}F^{(i)}(e)}{u(e)}\right)\quad\text{and}\quad\phi^{(i)}=\phi^{(i-1)}\left(1+\frac{\epsilon}{\kappa}\cdot\frac{b(z^{(i)}F^{(i)})}{B}\right).$$

This concludes the description of a single iteration. The algorithm terminates upon reaching the first iteration t for which $D(t)\geq 1$ and outputs

$$\bar{F}=\frac{z^{(1)}F^{(1)}+z^{(2)}F^{(2)}+\cdots+z^{(t-1)}F^{(t-1)}}{z^{(1)}+z^{(2)}+\cdots+z^{(t-1)}}\quad\text{and}\quad\lambda=\frac{z^{(1)}+z^{(2)}+\cdots+z^{(t-1)}}{\kappa\log_{1+\epsilon}1/\delta}.$$

Analysis. We will analyze the values of $D(\ell^{(i)},\phi^{(i)})$ and $\alpha(\ell^{(i)},\phi^{(i)})$ only for the lengths $\ell^{(i)},\phi^{(i)}$. To avoid clutter, we denote $D(i)=D(\ell^{(i)},\phi^{(i)})$ and $\alpha(i)=\alpha(\ell^{(i)},\phi^{(i)})$. For each iteration i, we

have

$$\begin{aligned}
D(i) &= \sum_{e\in E} u(e)\cdot \ell^{(i)}(e) + B\cdot\phi^{(i)} \\
&= \sum_{e\in E} u(e)\cdot \ell^{(i-1)}(e)\left(1+\frac{\epsilon}{\kappa}\cdot\frac{z^{(i)}F^{(i)}(e)}{u(e)}\right) + B\cdot\phi^{(i-1)}\left(1+\frac{\epsilon}{\kappa}\cdot\frac{b(z^{(i)}F^{(i)})}{B}\right) \\
&= D(i-1) + \frac{\epsilon}{\kappa}\cdot\sum_{e\in E} z^{(i)}F^{(i)}(e)\cdot\ell^{(i-1)}(e) + \frac{\epsilon}{\kappa}\cdot z^{(i)}\cdot b(F^{(i)})\cdot\phi^{(i-1)} \\
&= D(i-1) + \frac{\epsilon}{\kappa}\cdot z^{(i)}\left(\sum_{e\in E} F^{(i)}(e)\cdot(\ell^{(i-1)}(e) + b(e)\phi^{(i-1)})\right) \\
&\le D(i-1) + \frac{\epsilon}{\kappa}\cdot z^{(i)}\cdot(1+\epsilon)s\cdot\alpha(i-1).
\end{aligned}$$

Since $D(i-1)/\alpha(i-1)\ge\beta$ by definition of β, we have

$$D(i)\le D(i-1)+\frac{\epsilon}{\kappa}\cdot z^{(i)}\cdot(1+\epsilon)s\cdot\frac{D(i-1)}{\beta}.$$

Define $\epsilon'=\epsilon(1+\epsilon)s/\kappa$ so that

$$D(i)\le\left(1+\frac{\epsilon(1+\epsilon)sz^{(i)}}{\kappa\beta}\right)D(i-1)=\left(1+\frac{\epsilon' z^{(i)}}{\beta}\right)D(i-1).$$

Since $D(0)=m\delta$ we have for $i\ge 1$

$$\begin{aligned}
D(i) &\le \left(\prod_{j\le i}(1+\epsilon' z^{(j)}/\beta)\right)m\delta = \left(1+\frac{\epsilon' z^{(i)}}{\beta}\right)m\delta\prod_{j\le i-1}\left(1+\frac{\epsilon' z^{(j)}}{\beta}\right) \\
&\le (1+\epsilon')m\delta\exp\left(\frac{\epsilon'\sum_{j\le i-1} z^{(j)}}{\beta}\right),
\end{aligned}$$

where the last inequality uses our assumption that $\beta\ge 1$ and the fact that $z^{(j)}\le 1$ by definition. To avoid clutter, define $z^{(\le i)}=\sum_{j\le i} z^{(j)}$ for all i.

The procedure stops at the first iteration t for which $D(t)\ge 1$. Therefore,

$$1\le D(t)\le(1+\epsilon')m\delta\exp\left(\frac{\epsilon' z^{(\le t-1)}}{\beta}\right),$$

which implies

$$\frac{\beta}{z^{(\le t-1)}}\le\frac{\epsilon'}{\ln\frac{1}{(1+\epsilon')m\delta}}. \tag{1}$$

Claim 9.3. *The scaled down flow* $\frac{1}{\kappa\log_{1+\epsilon}1/\delta}(z^{(1)}F^{(1)}+z^{(2)}F^{(2)}+\cdots+z^{(t-1)}F^{(t-1)})$ *is capacity-respecting with cost at most* B.

Proof. To show it is capacity-respecting, consider an edge e. On each iteration, we route $z^{(i)}F^{(i)}(e) \le \kappa u(e)$ units of flow through e and increase its length by a factor $(1 + \frac{\epsilon}{\kappa} \cdot \frac{z^{(i)}F^{(i)}(e)}{u(e)}) \le 1 + \epsilon$. So for every $\kappa u(e)$ units of flow routed through e over the iterations, we increase its length by at least a factor $1 + \epsilon$. Initially, its length is $\delta/u(e)$, and after $t-1$ iterations, since $D(t-1) < 1$, the length of e satisfies $\ell^{(t-1)}(e) \le D(t-1)/u(e) < 1/u(e)$. Therefore the total amount of flow through e in the first $t-1$ phases is strictly less than $\kappa \log_{1+\epsilon} \frac{1/u(e)}{\delta/u(e)} = \kappa \log_{1+\epsilon} 1/\delta$ times its capacity. Scaling the flow down by $\kappa \log_{1+\epsilon} 1/\delta$, we obtain a capacity-respecting flow.

To show that the scaled down flow has cost at most B, similarly observe that on each iteration, we route a flow of cost $b(z^{(i)}F^{(i)}) \le B$ and increase the length $\phi^{(i)}$ by a factor $(1 + \frac{\epsilon}{\kappa} \cdot \frac{b(z^{(i)}F^{(i)})}{B}) \le 1 + \epsilon/\kappa$ over the previous length $\phi^{(i-1)}$. So for every B cost of flow routed, we increase the length $\phi^{(i)}$ by at least a factor $1 + \epsilon/\kappa$. And for every κB cost of flow routed, the length increases by at least a factor $(1+\epsilon/\kappa)^\kappa \ge 1+\epsilon$. Initially, $\phi^{(0)} = \delta/B$, and after $t-1$ iterations, since $D(t-1) < 1$, the length satisfies $\phi^{(t-1)} \le D(t-1)/B < 1/B$. Therefore the total cost of flow in the first $t-1$ phases is strictly less than $\kappa B \log_{1+\epsilon} \frac{1/B}{\delta/B} = \kappa B \log_{1+\epsilon} 1/\delta$. Scaling the flow down by $\kappa \log_{1+\epsilon} 1/\delta$, we obtain a flow with cost at most B. □

Recall that $\bar{F} = \frac{z^{(1)}F^{(1)} + z^{(2)}F^{(2)} + \ldots + z^{(t-1)}F^{(t-1)}}{z^{(\le t-1)}}$ and $\lambda = \frac{z^{(\le t-1)}}{\kappa \log_{1+\epsilon} 1/\delta}$, so $\lambda\bar{F}$ is capacity-respecting with cost at most B, fulfilling Condition 1. To establish Condition 2, we use Equation (1) and the fact that $\delta = m^{1/\epsilon}$ to obtain

$$\frac{\lambda}{\lambda^*} = \frac{z^{(\le t-1)}}{\kappa \log_{1+\epsilon} 1/\delta} \cdot \frac{1}{\beta} \ge \frac{\ln \frac{1}{(1+\epsilon')m\delta}}{\epsilon' \cdot \kappa \log_{1+\epsilon} 1/\delta} = \frac{\ln \frac{1}{(1+\epsilon')m\delta}}{\epsilon s \log_{1+\epsilon} 1/\delta} \ge \frac{1 - O(\epsilon)}{s}.$$

Running time. Recall from above that $\lambda\bar{F}$ is capacity-respecting with cost at most B, so $\lambda \le \beta$. Since $\lambda = \frac{z^{(\le t-1)}}{\kappa \log_{1+\epsilon} 1/\delta}$, we obtain $z^{(\le t-1)} \le \beta\kappa \log_{1+\epsilon} 1/\delta$. On each iteration $i \le t-1$, either $z^{(i)} = 1$ or $z^{(i)} < 1$, and the latter case implies that $b(z^{(i)}F^{(i)}) = B$, which means $\phi^{(i)} = \phi^{(i-1)}(1 + \epsilon/\kappa)$. Initially, $\phi^{(0)} = \delta/B$, and after $t-1$ iterations, since $D(t-1) < 1$, we have $\phi^{(t-1)} \le D(t-1)/B < 1/B$. So the event $\phi^{(i)} = \phi^{(i-1)}(1 + \epsilon/\kappa)$ can happen at most $\log_{1+\epsilon/\kappa} 1/\delta$ times. It follows that $z^{(i)} < 1$ for at most $\log_{1+\epsilon/\kappa} 1/\delta$ values of $i \le t-1$. Since $z^{(\le t-1)} \le \beta\kappa \log_{1+\epsilon} 1/\delta$, we have $z^{(i)} = 1$ for at most $\beta\kappa \log_{1+\epsilon} 1/\delta$ many values of $i \le t-1$. Therefore, the number of iterations t is at most

$$\log_{1+\epsilon/\kappa} 1/\delta + \beta\kappa \log_{1+\epsilon} 1/\delta + 1 = O(\beta\kappa\epsilon^{-2} \log m).$$

By scaling all edge capacities and costs by various powers of two, we can ensure that $\beta \in [1, 2]$ on at least one guess, so the number of iterations is $O(\kappa \log_{1+\epsilon} 1/\delta) = O(\kappa\epsilon^{-2} \log m)$. Doing so also ensures that $\lambda^* = \beta \ge 1$ as we had previously assumed. For incorrect guesses, we terminate the algorithm above after $O(\kappa\epsilon^{-2} \log m)$ iterations to not waste further computation. Among all guesses, we take the one with maximum λ that satisfies feasibility (Condition 1). Since there are $O(\log n)$ relevant powers of two, the running time picks up an overhead of $O(\log n)$.

Flow oracle. Finally, if oracle $\mathcal{O}$ outputs a flow oracle, then the algorithm can return the following flow oracle for the output flow F: on input subset S of pairs of vertices of G, query the flow oracles

of the flows $F^{(1)}, F^{(2)}, \ldots, F^{(t-1)}$ to obtain flows $F_S^{(1)}, F_S^{(2)}, \ldots, F_S^{(t-1)}$, respectively, and output

$$F_S = \frac{z^{(1)}F_S^{(1)} + z^{(2)}F_S^{(2)} + \cdots + z^{(t-1)}F_S^{(t-1)}}{z^{(1)} + z^{(2)} + \cdots + z^{(t-1)}}.$$

Querying each flow oracle $F_S^{(i)}$ takes work Q_w and depth Q_d and can be done in parallel. Since there are $t-1 \le O(\kappa\epsilon^{-2}\log m)$ many flow oracles, the total work is $\tilde{O}(\kappa\epsilon^{-2}Q_w)$ and the total depth is $\tilde{O}(Q_d)$.

9.2 Flow Boosting: Thresholded Instantiation

By choosing an appropriate convex set $\mathcal{F}$ of flows, we directly obtain boosting theorems for computing a flow of a given target value τ that partially routes a given flow demand D. Note that this formulation includes both min-cost concurrent and non-concurrent multicommodity flow: for concurrent we set $\tau = |D|$ forcing the flow to route the entire demand D, and for non-concurrent we set $\tau = 1$ which is less than or equal to the value of any nonzero demand pair (when the demand is integral).

Theorem 9.4 (Thresholded Flow Boosting)**.** *Let $G = (V, E, u, b)$ be a graph with capacity function u and cost function b. Let $B \ge 0$ be the cost budget. Let $D : V \times V \to \mathbb{R}_{\ge 0}$ be a flow demand, and let $\epsilon, s, \kappa \ge 0$ be parameters. Let τ be the target flow value parameter such that there exists a capacity-respecting flow partially routing D of value at least τ.*

Suppose an algorithm is given an oracle $\mathcal{O}$ that, given any integral edge length function $\ell : E \to \{1, 2, \ldots, O(m^{1/\epsilon}N/\epsilon)\}$, computes the edge representation of a flow F of value $\mathrm{val}(F) \ge \tau$ partially routing D such that

- Length slack s: $\sum_{e\in E} F(e)\cdot\ell(e) \le s\cdot\sum_{e\in E} F^*(e)\cdot\ell(e)$ *for any capacity-respecting flow F^* of value $\mathrm{val}(F^*) \ge \tau$ partially routing D, and*
- Congestion slack κ: $F(e) \le \kappa u(e)$ *for all $e \in E$, i.e., $F(e)/\kappa$ is capacity-respecting.*

Then, there is a deterministic algorithm that makes $O(\kappa\epsilon^{-2}\log^2 n)$ calls to oracle $\mathcal{O}$ and outputs the edge representation of a flow $\bar{F}$ of value $\mathrm{val}(\bar{F}) \ge \tau$ partially routing D and scalar $\lambda \ge 0$ such that

1. Feasibility: *The flow $\lambda\bar{F}$ is capacity-respecting with cost at most B, and*
2. Approximation factor $\approx s$: *Let λ^* be the maximum value such that there exists a flow F^* of value $\mathrm{val}(F^*) \ge \tau$ partially routing D where $\lambda^* F^*$ is capacity-respecting with cost at most B. Then, $\lambda \ge \frac{1-O(\epsilon)}{s}\lambda^*$.*

Moreover, the flow $\bar{F}$ is a convex combination of the flows returned by oracle $\mathcal{O}$, and this convex combination can be output as well.

Furthermore, if the oracle $\mathcal{O}$ also outputs a flow oracle with query work Q_w and query depth Q_d, then the algorithm can also output a flow oracle for $\bar{F}$ with query work $\tilde{O}(\kappa\epsilon^{-2}Q_w)$ work and $\tilde{O}(Q_d)$ depth.

The algorithm takes $\tilde{O}(\kappa\epsilon^{-2}m)$ work and $\tilde{O}(\kappa\epsilon^{-2})$ time outside of the oracle calls.

Proof. Let $\mathcal{F}$ be the set of flows of value at least τ partially routing D. Apply Theorem 9.2. □

10 Constant-Approximate Multi-commodity Flow

This section concludes with our constant-approximate algorithms for k-commodity flow in $(m+k)^{1+\epsilon}$ time. The main results are stated in Theorem 10.5. Below, we first implement the oracle for flow boosting in Section 10.1 and then apply the oracle to the flow boosting framework in Section 10.2 to obtain our results.

10.1 Oracle for Flow Boosting

In this subsection we construct the oracle required for Theorem 9.4.

Theorem 10.1 (Oracle for Flow-Boosting)**.** *Let G be a graph with integral edge lengths $\ell \geq 1$ and capacities $u \geq 1$, D an integral $\deg_G$-respecting demand, τ an integral required flow amount such that there exists a value-τ capacity-respecting flow partially routing D and $\epsilon \in (\log^{-c} N, 1)$ for a small enough constant c a tradeoff parameter. Then,* UnboostedFlow(G, D, τ, ϵ) *(Algorithm 7) returns a flow oracle $\mathcal{O}_F$ for a flow F of value $\text{val}(F) \geq \tau$ partially routing D, with*

1. *Length slack s: $\sum_{e\in E} F(e)\cdot\ell(e) \leq s\cdot\sum_{e\in E} F^*(e)\cdot\ell(e)$ for any capacity-respecting flow F^* of value $\text{val}(F^*) \geq \tau$ partially routing D, and*
2. *Congestion Slack κ: $F(e) \leq \kappa u(e)$ for all $e \in E$, i.e., $F(e)/\kappa$ is capacity-respecting,*

for length slack $s = \exp(\text{poly}(1/\epsilon))$ and congestion slack $\kappa = N^{\text{poly}(\epsilon)}$.

The algorithm has depth and the flow oracle has query depth $N^{\text{poly}(\epsilon)}$, and the algorithm has work and the flow oracle query work $(|E| + \text{supp}(D))\cdot N^{\text{poly}(\epsilon)}$. The produced flow has support size $(|E| + \text{supp}(D))N^{\text{poly}(\epsilon)}$.

Proof of Theorem 10.1. Let F^* be the capacity-respecting flow of value at least τ routing a subdemand of D on G of minimum total length T^*. Theorem 6.4 guarantees that G' is a t-step emulator for $\deg_G$ with length slack $s' = \exp(\text{poly}(1/\epsilon))$ and congestion slack $\kappa' = N^{\text{poly}(\epsilon)}$. Since D is $\deg_G$-respecting, there is a flow $F^{*\prime}$ on G' routing the same demand as F^* of step-length t, congestion at most κ' and total length T^*s'. As G'' is simply G' with capacities scaled up by κ', the flow $F^{*\prime}$ on G'' is a capacity-respecting flow partially routing D of step length t, total length T^*s', and value at least τ.

Algorithm 6 guarantees that for input (G'', D, t, T, ϵ), for every capacity-respecting flow F^* partially routing D of step length t and total length T, the returned flow F has value $\text{val}(F) \geq \text{val}(F^*)$. Thus, notably when $T \geq T^*s'$, the returned flow has value at least τ. Let T' be the minimum value of T for which the returned flow F_{res} had value at least τ on line 5c. By the above argument, $T' \leq \lceil T^*s' \rceil$.

Algorithm 6 guarantees that F_{res} on G'' is a congestion-κ'' flow partially routing D of step length ts'', total length $T's''$ and value at least τ for the congestion slack $\kappa'' = N^{\text{poly}(\epsilon)}$ and length slack $s'' = \exp(\text{poly}(1/\epsilon))$ of the flow algorithm. The flow has the same step length, total length and value, but congestion $\kappa'\kappa'' = \kappa = N^{\text{poly}(\epsilon)}$ on G'. Since G' can be embedded into G with congestion 1 and length slack 1, the flow F that is F_{res} mapped from G' to G by the embedding is a congestion-$\kappa = \kappa'\kappa''$ flow partially routing D of value at least τ of total length $T^*s = \lceil T^*s' \rceil s'' = T's''$ for length

Algorithm 7 Fast Unboosted Flow: UNBOOSTEDFLOW(G, D, τ, ϵ)

Input: Graph G with integral edge lengths $\ell \geq 1$ and capacities $u \geq 1$, integral $\deg_G$-respecting demand D, integral required flow amount τ such that there exists a value-τ capacity-respecting flow partially routing D, tradeoff parameter ϵ.

Output: A flow oracle $\mathcal{O}_F$ for a flow F of value $\text{val}(F) \geq \tau$ partially routing D with length slack $s = \exp(\text{poly}(1/\epsilon))$, congestion slack $\kappa = N^{\text{poly}(\epsilon)}$ and support size $(|E| + \text{supp}(D))N^{\text{poly}(\epsilon)}$.

1. Let $t, G' = \text{LOWSTEPEMU}(G, \epsilon)$.
2. Let κ' be the congestion slack of G', and let $G'' = \kappa' G'$ be G' with capacities scaled up by κ'.
3. Let $T_{\text{low}} = 0$ and $T_{\text{high}} = N$.
4. Let $F_{\text{res}} = 0$.
5. While $T_{\text{low}} \leq T_{\text{high}}$:
   (a) Let $T = \left\lfloor \frac{T_{\text{low}} + T_{\text{high}}}{2} \right\rfloor$.
   (b) Let $F' = \text{LOWSTEPNONCONCFLOW}(G'', D, t, T, \epsilon)$.
   (c) If $\text{val}(F') \geq \tau$,
      - Set $T_{\text{high}} = T - 1$.
      - Set $F_{\text{res}} = F'$.
   (d) Else, set $T_{\text{low}} = T + 1$.
6. Return $\mathcal{O}_F = (G', F_{\text{res}})$ with query function $\mathcal{O}_F(S) = \text{FLOWMAP}(G', F'_S)$.

slack $s = \exp(\text{poly}(1/\epsilon))$ and congestion slack $\kappa = N^{\text{poly}(\epsilon)}$[4], as desired. As $|E(G')| = |E|N^{\text{poly}(\epsilon)}$ and the embedding from G' to G maps edges to paths, thus not increasing the flow support size, the flow support size is $(|E| + \text{supp}(D)) \cdot N^{\text{poly}(\epsilon)}$ as desired.

The algorithm consists of one call to LOWSTEPEMU and $\tilde{O}(1)$ calls to LOWSTEPNONCONCFLOW on a $|E|N^{\text{poly}(\epsilon)}$-edge graph, thus its work and depth are $(|E| + \text{supp}(D)) \cdot N^{\text{poly}(\epsilon)}$ and $N^{\text{poly}(\epsilon)}$ respectively. □

10.2 Constant-Approximate Min Cost Multi-Commodity Flow

The following theorem is a generalisation of concurrent and non-concurrent flow, which gives concurrent flow when $\tau = |D|$ and non-concurrent flow when $\tau = 1$.

Theorem 10.2 (Constant-Approximate Multi-Commodity Flow)**.** *Let $G = (V, E, u, b)$ be a connected graph with edge capacities $u \geq 1$ and costs $b \geq 0$. Let $B \geq 0$ be the cost budget. Let $D : V \times V \to \mathbb{N}$ be a integral flow demand, $\tau \in [|D|]$ a integral required flow amount and $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c be a tradeoff parameter. Then,* MCMCFLOW$(G, B, D, \tau, \epsilon)$ *returns a flow oracle $\mathcal{O}_F$ for a flow F of value* $\text{val}(F) \geq \tau$ *partially routing D and a value $\lambda \geq 0$,*

[4]Note that since τ is a integer and $\ell \geq 1$, $T^* \geq 1$ and taking ceil does not affect the asymptotic congestion slack

such that

- *Feasibility: the flow λF is capacity-respecting with cost at most B, and*
- *Approximation factor $1/\exp(\text{poly}(1/\epsilon))$: let λ^* be the maximum value such that there exists a flow F^* of value $\text{val}(F^*) \geq \tau$ partially routing D where $\lambda^* F^*$ is capacity-respecting with cost at most B. Then, $\lambda \geq \frac{1}{\exp(\text{poly}(1/\epsilon))}\lambda^*$.*

The algorithm has work and the flow oracle query work $(|E| + \text{supp}(D))N^{\text{poly}(\epsilon)}$, and the algorithm has depth and the flow oracle query depth $N^{\text{poly}(\epsilon)}$. The produced flow F has support size $(|E| + \text{supp}(D))N^{\text{poly}(\epsilon)}$.

Algorithm 8 Constant-Approximate MCMC Flow: MCMCFlow$(G, B, D, \tau, \epsilon)$

Input: Connected graph G with edge capacities $u \geq 1$ and costs $b \geq 0$, integral demand D, integral required flow amount τ, tradeoff parameter ϵ.
Output: A flow oracle $\mathcal{O}_F$ for a flow F of value $\text{val}(F) \geq \tau$ partially routing D and a value $\lambda \geq 0$, such that λF is capacity-respecting with cost at most B, and $\lambda \geq (1/\exp(\text{poly}(1/\epsilon)))\lambda^*$ for the optimal λ^*, F^* pair.

1. Let $\gamma = |D|$.
2. Let $\epsilon' = O(1)$ be small enough that $1 - O(\epsilon') \geq \frac{1}{2}$ in *Theorem* 9.4.
3. Let $\mathcal{O}_F$, λ be the flow-scalar pair returned by Theorem 9.4 on graph G with capacities γu, costs b, total cost budget γB, demand D, required flow amount τ, parameters $(\epsilon', \exp(\text{poly}(1/\epsilon)), N^{\text{poly}(\epsilon)})$ and flow oracle UnboostedFlow(G, D, τ, ϵ).
4. Return $\mathcal{O}_F$, λ/γ.

Proof of Theorem 10.2. For a flow F, parameter λ and value $\gamma \geq 0$, the following are equivalent:

- λF is capacity-respecting with capacities u and has cost at most B,
- $(\lambda\gamma)F$ is capacity-respecting with capacities γu and has cost at most γB,

but scaling all capacities and the cost bound by $\gamma = |D|$ guarantees that there exists a capacity-respecting flow partially routing D of value at least τ, which is required by Theorem 9.4.

UnboostedFlow(G, D, τ, ϵ) is a oracle function of the kind required for Theorem 9.4 for length slack $s = \exp(\text{poly}(1/\epsilon))$ and congestion slack $\kappa = N^{\text{poly}(\epsilon)}$. Note that D is $\deg_{G'}$-respecting for graph G' that is G with capacities γu as $u \geq 1$.

The flow F returned as a flow oracle $\mathcal{O}_F$ and value λ returned by Theorem 9.4 are guaranteed to satisfy the required properties: F partially routes D and has value at least τ, λF is capacity-respecting with cost at most B, and for the maximum value λ^* such that there exists a flow F^* of value $\text{val}(F^*) \geq \tau$ partially routing D where $\lambda^* F^*$ is capacity-respecting with cost at most B, $\lambda \geq \frac{1-O(\epsilon')}{s}\lambda^* \geq \frac{1}{2s}\lambda^* = \frac{1}{\exp(\text{poly}(1/\epsilon))}\lambda^*$.

The flow support size is at most the product of the support size $(|E| + |\text{supp}(D)|)N^{\text{poly}(\epsilon)}$ of flows returned by Theorem 10.1 and the number of oracle calls $O(\kappa\epsilon^{-2}\log^2 n) = N^{\text{poly}(\epsilon)}$, giving the desired bound of $(|E| + |\text{supp}(D)|)N^{\text{poly}(\epsilon)}$.

For work and depth, Theorem 9.4 makes $O(\kappa\epsilon'^{-2}\log^2 n) = N^{\text{poly}(\epsilon)}$ calls to the oracle, which has work $(|E|+|\text{supp}(D)|)N^{\text{poly}(\epsilon)}$ and depth $N^{\text{poly}(\epsilon)}$, and takes $\tilde{O}(\kappa\epsilon'^{-2}m) = |E|\cdot N^{\text{poly}(\epsilon)}$ work and has depth $\tilde{O}(\kappa\epsilon'^{-2}) = N^{\text{poly}(\epsilon)}$ outside the oracle calls. This gives total work $(|E| + |\text{supp}(D)|)N^{\text{poly}(\epsilon)}$ and depth $N^{\text{poly}(\epsilon)}$ as desired. □

10.3 Constant-Approximate Concurrent/Non-Concurrent Flow

Finally, we prove a formal version of Theorem 1.1, first giving formal definitions for concurrent and non-concurrent flow.

Definition 10.3 (Concurrent Flow Problem)**.** *Let G be a connected graph with edge capacities $u \geq 1$ and $D : V \times V \mapsto \mathbb{N}$ an integral demand. The concurrent flow problem asks to find a capacity-respecting flow F routing λD for maximum λ. An algorithm is a C-approximation for the concurrent flow problem if it always produces a capacity-respecting flow F routing λD, where $\lambda \geq C\lambda^*$ and λ^* is the maximum value for which there exists a capacity-respecting F^* routing $\lambda^* D$.*

In the concurrent flow problem with costs, each edge has a cost $b \geq 0$ and there is a total cost budget B: the produced flow F must additionally have total cost $\sum_P F(P)\sum_{e\in P} b(e) \leq B$. An algorithm is a C-approximation for the concurrent flow problem with costs if it always produces a capacity-respecting flow F of total cost at most B routing λD, where $\lambda \geq C\lambda^*$ and λ^* is the maximum value for which there exists a capacity-respecting F^* of total cost at most B routing $\lambda^* D$.

Definition 10.4 (Non-Concurrent Flow Problem)**.** *Let G be a connected graph with edge capacities $u \geq 1$ and S a set of vertex pairs. The non-concurrent flow problem asks to find a capacity-respecting flow F routing flow between vertex pairs in S, i.e. $\text{supp}(D_F) \subseteq S$ of maximum value. An algorithm is a C-approximation for the non-concurrent flow problem if it always produces a capacity-respecting flow F routing flow between vertex pairs in S of value $\text{val}(F) \geq C\text{val}(F^*)$, where F^* is the maximum-value capacity-respecting flow routing flow between vertex pairs in S.*

In the non-concurrent flow problem with costs, each edge has a cost $b \geq 0$ and there is a total cost budget B: the produced flow F must additionally have total cost $\sum_P F(P)\sum_{e\in P} b(e) \leq B$. An algorithm is a C-approximation for the concurrent flow problem with costs if it always produces a capacity-respecting flow F of total cost at most B routing flow between vertex pairs in S of value $\text{val}(F) \geq C\text{val}(F^*)$, where F^* is the maximum-value capacity-respecting flow of total cost at most B routing flow between vertex pairs in S.

Theorem 10.5 (Constant Approximate Concurrent/Non-Concurrent Flow)**.** *For every tradeoff parameter $\epsilon \in (\log^{-c} N, 1)$ for some sufficiently small constant c, for both concurrent and non-concurrent multi-commodity flow with costs, there exists a $(m+k)^{1+\text{poly}(\epsilon)}$-work $(m+k)^{\text{poly}(\epsilon)}$-depth $O(2^{-1/\epsilon})$-approximate algorithm that returns a flow oracle $\mathcal{O}_F$ for the flow F.*

Proof. We can select $\epsilon' = \text{poly}(\epsilon)$ such that $\exp(\text{poly}(1/\epsilon'))^{-1} = O(2^{-1/\epsilon})$. Both the concurrent flow and non-concurrent flow algorithm are direct consequences of applying Theorem 10.2 with this ϵ' and differing D, τ and returning $\lambda\mathcal{O}_F$:

- Concurrent Flow: apply Theorem 10.2 with $\tau = |D|$, return $\lambda\mathcal{O}_F$.

- Non-Concurrent Flow: apply Theorem 10.2 with $D(a,b) = \mathbb{I}[(a,b) \in S]$, $\tau = 1$, return $\lambda \mathcal{O}_F$.

□

References

[CK09] Julia Chuzhoy and Sanjeev Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *Journal of the ACM (JACM)*, 56(2):1–28, 2009.

[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. *arXiv preprint arXiv:2203.00671*, 2022.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *Journal of the ACM (JACM)*, 47(1):132–166, 2000.

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

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

[HHG22] Bernhard Haeupler, Jonas Huebotter, and Mohsen Ghaffari. A cut-matching game for constant-hop expanders. *arXiv preprint arXiv:2211.11726*, 2022.

[HHS23] Bernhard Haeupler, D. Ellis Hershkowitz, and Thatchaphol Saranurak. Maximum length-constrained flows and disjoint paths: Distributed, deterministic, and fast. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1371–1383. ACM, 2023.

[HHT23] Bernhard Haeupler, D Ellis Hershkowitz, and Zihan Tan. Parallel greedy spanners. *arXiv preprint arXiv:2304.08892*, 2023.

[HHT24] Bernhard Haeupler, D Ellis Hershkowitz, and Zihan Tan. New structures and algorithms for length-constrained expander decompositions. *arXiv preprint arXiv:2404.13446*, 2024.

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

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

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[Rac02] Harald Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.

[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*, pages 227–238. SIAM, 2014.

[She17] Jonah Sherman. Area-convexity, l_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 452–460, 2017.

[vdBZ23] Jan van den Brand and Daniel Zhang. Faster high accuracy multi-commodity flow from single-commodity techniques. In *FOCS*, 2023.

A Derivation of Lemma 7.5

In this appendix section, we give the derivation of Lemma 7.5 from Theorem A.1 (of [HHT24]), a low-support-size version of the flow algorithms of [HHS23]. The theorem gives an algorithm for efficiently computing a cutmatch, defined below.

Cutmatch. Given a graph $G = (V, E)$ with length ℓ, an h-length ϕ-sparse cutmatch of congestion γ between disjoint node weighting pairs $\{(A_j, A_j)\}_{j\in[k]}$ consists of, for each i, a partition of the node-weightings $A_j = M_j + U_j$ and $A'_j = M'_j + U'_j$ where M_j, M'_j and U_j, U'_j are the "matched" and "unmatched" parts respectively and $|M_j| = |M'_j|$ and

- A h-length flow $F = \sum_j F_j$ in G with lengths ℓ of congestion γ, such that, for each $j \in [k]$, F_j is a complete M_j to M'_j -flow.
- A h-length moving cut C in G, such that for all $j \in [k]$, $\text{supp}(U_j)$ and $\text{supp}(U'_j)$ are at least h-far in $G - C$, and C has size at most

$$|C| \leq \phi \cdot \left(\left(\sum_j |A_j| \right) - \text{val}(F) \right)$$

Theorem A.1 ([HHT24]). *Let $G = (V, E)$ be a graph on m edges with edge lengths $\ell \geq 1$ and capacities $u \geq 1$. Then, for any $h \geq 1$, $\phi \leq 1$, there is an algorithm that, given node-weighting pairs $\{(A_j, A'_j)\}_{j\in[k]}$, outputs a multi-commodity h-length ϕ-sparse cutmatch (F, C) of congestion γ where $\gamma = \tilde{O}\left(\frac{1}{\phi}\right)$. This algorithm has depth $\tilde{O}(k \cdot \text{poly}(h))$ and work $\tilde{O}(|(E(G)| + \sum_j |\text{supp}(A_j + A'_j)|) \cdot k \cdot \text{poly}(h))$. Moreover, $|\text{supp}(F)| \leq \tilde{O}(|E(G)| + k + \sum_j |\text{supp}(A_j + A'_j)|)$.*

To obtain Lemma 7.5, we use the fact that routers have no sparse cuts; selecting ϕ and h as twice the router's parameters, the multicommodity flow produced by the cutmatch must completely satisfy the demand.

Lemma 7.5. *Let $G = (V, E)$ be a t-step γ-router for node weighting A, and $\{(A_j, A'_j)\}_{j\in[k]}$ node weighting pairs such that $\sum_j A_j + A'_j$ is A-respecting and $|A_j| = |A'_j|$ for all j. Then, one can compute a flow $F = \sum_j F_j$ where F_j is a complete A_j to A'_j -flow for each $j \in [k]$ with*

- *length $2t$ and congestion $\tilde{O}(\gamma)$, and*
- *support size $|\mathrm{supp}(F)| \leq \tilde{O}(|E(G)| + k + \sum_j |\mathrm{supp}(A_j + A'_j)|)$,*

with $\tilde{O}((|E(G)| + \sum_j |\mathrm{supp}(A_j + A'_j)|) \cdot k \cdot \mathrm{poly}(t))$ work and $\tilde{O}(k \cdot \mathrm{poly}(t))$ depth.

Proof. We may assume without loss of generality that the supports of A_j and A'_j are disjoint for all j; if they are not, we can add a flow path of length 0 from that vertex to itself of value equal to the minimum of the two. Now, let (F, C) be a $2t$-length $\left(\frac{1}{2\gamma}\right)$-sparse cutmatch (F, C) between $\{(A_j, A'_j)\}_{j\in[k]}$. Then, for each $j \in [k]$, F_j is a complete A_j to A'_j -flow. Thus, calling Theorem A.1 with length $2t$ and sparsity $\left(\frac{1}{2\gamma}\right)$ suffices.

We prove the claim. Assume the contrary; then, $\sum_j |U_j| = \sum_j |U'_j| > 0$. For each j, let D_j be an arbitrary demand such that $\mathrm{load}(D_j) = U_j + U'_j$, and let $D = \sum_j D_j$. Then, D is A-respecting, and there exists a t-step γ-congestion flow on G satisfying D. Let F^* be that flow. We have

$$|C| = \sum_{e\in G} C(e)u(e) \leq \left(\frac{1}{2\gamma}\right)|D|,$$

but

- for every path $P \in F^*$, $\sum_{e\in P} C(e) > \frac{1}{2}$, as C is a length-$2t$ cut and every flow-path had its length increased by more than t, and
- $\frac{1}{\gamma}\sum_{P\in F^*:e\in P} F^*(P) \leq u(e)$, as F^* has congestion γ,

thus

$$\left(\frac{1}{2\gamma}\right)|D| \geq \sum_{e\in G} C(e)u(e) \geq \frac{1}{\gamma}\sum_{P\in F^*} F^*(P)\sum_{e\in P} C(e) > \frac{1}{2\gamma}|F^*|,$$

a contradiction. Thus, the unmatched part of the cutmatch and the cut are empty. □