# Maximum Length-Constrained Flows and Disjoint Paths: Distributed, Deterministic and Fast*

Bernhard Haeupler
Carnegie Mellon University &
ETH Zürich
haeupler@cs.cmu.edu

D Ellis Hershkowitz
Carnegie Mellon University &
ETH Zürich
dhershko@cs.cmu.edu

Thatchaphol Saranurak
University of Michigan
thsa@umich.edu

**Abstract**

Computing routing schemes that support both high throughput and low latency is one of the core challenges of network optimization. Such routes can be formalized as $h$-length flows which are defined as flows whose flow paths are restricted to have length at most $h$. Many well-studied algorithmic primitives—such as maximal and maximum length-constrained disjoint paths—are special cases of $h$-length flows. Likewise the optimal $h$-length flow is a fundamental quantity in network optimization, characterizing, up to poly-log factors, how quickly a network can accomplish numerous distributed primitives.

In this work, we give the first efficient algorithms for computing $(1-\epsilon)$-approximate $h$-length flows. We give deterministic algorithms that take $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}))$ parallel time and $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}) \cdot 2^{O(\sqrt{\log n})})$ distributed CONGEST time. We also give a CONGEST algorithm that succeeds with high probability and only takes $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}))$ time.

Using our $h$-length flow algorithms, we give the first efficient deterministic CONGEST algorithms for the maximal length-constrained disjoint paths problem—settling an open question of Chang and Saranurak (FOCS 2020)—as well as essentially-optimal parallel and distributed approximation algorithms for maximum length-constrained disjoint paths. The former greatly simplifies deterministic CONGEST algorithms for computing expander decompositions. We also use our techniques to give the first efficient $(1-\epsilon)$-approximation algorithms for bipartite $b$-matching in CONGEST. Lastly, using our flow algorithms, we give the first algorithms to efficiently compute $h$-length cutmatches, an object at the heart of recent advances in length-constrained expander decompositions.

*First two authors supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF1750808, a Sloan Research Fellowship, funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272) and the Swiss National Foundation (project grant 200021-184735). Second author also supported by the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

# Contents

# 1 Introduction

Throughput and latency are two of the most fundamental quantities in a communication network. Given node sets $S$ and $T$, throughput measures the rate at which bits can be delivered from $S$ to $T$ while the worst-case latency measures the maximum time it takes for a bit sent from $S$ to arrive at $T$. Thus, a natural question in network optimization is:

*How can we achieve high throughput while maintaining a low latency?*

If we imagine that each edge in a graph incurs some latency and edges in a graph can only support limited bandwidth, then achieving high throughput subject to a latency constraint reduces to finding a large collection of paths that are both short and non-overlapping. One of the simplest and most well-studied ways of formalizing this is the maximal edge-disjoint paths problems (henceforth we use $h$-length to mean length *at most* $h$).

> **Maximal Edge-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of $h$-length edge-disjoint $S$ to $T$ paths $\mathcal{P}$ such that any $h$-length $S$ to $T$ path shares an edge with at least one path in $\mathcal{P}$.

The simplicity of the maximal edge-disjoint paths problem has made it a crucial primitive in numerous algorithms. For example, algorithms for maximal edge-disjoint paths are used in approximating maximum matchings [30] and computing expander decompositions [14, 38]. While efficient randomized algorithms are known for maximal edge-disjoint paths in the CONGEST model of distributed computation [12, 30], no deterministic CONGEST algorithms are known. Indeed, the existence of such algorithms was stated as an open question by Chang and Saranurak [12].

Of course, a maximal collection of routing paths need not be near-optimal in terms of cardinality and so a natural extension of the above problem is its *maximum* version.

> **Maximum Edge-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of $h$-length edge-disjoint $S$ to $T$ paths.

While this problem and its variants have received considerable attention [7, 9, 27], it is unfortunately known to suffer from strong hardness results: the above problem has an $\Omega(h)$ integrality gap and is $\Omega(h)$-hard-to-approximate under standard complexity assumptions in the directed case [4, 19]. Indeed, as observed in several works [1, 21, 27], working in the presence of latency bounds in the form of a length constraint can make otherwise tractable problems computationally infeasible and render otherwise structured objects poorly behaved.

In large part, the above problems are common primitives because their solutions are special cases of a more general class of routing schemes that are central to distributed computing. Namely, they are special cases of length-constrained flows.

> **Maximum Length-Constrained Flow**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of $h$-length $S$ to $T$ paths $\mathcal{P}$ and a value $f_P \geq 0$ for $P \in \mathcal{P}$ where $\sum_{P \ni a} f_P \leq 1$ for every $a \in A$ and $\sum_P f_P$ is maximized.

In several formal senses, length-constrained flows are *the* problem that describes how to efficiently communicate in a network. Haeupler et al. [20] showed that, up to poly-log factors, the maximum length-constrained flow gives the minimum makespan of multiple unicasts in a network, even when

(network) coding is allowed. Even stronger, the "best" length-constrained flow gives, up to poly-log factors, the optimal running time of a CONGEST algorithm for numerous distributed optimization problems, including minimum spanning tree (MST), approximate min-cut and approximate shortest paths [22]. Despite the key role these flows play in distributed computing, there are currently no known distributed (or even parallel!) algorithms for computing them. The need for algorithms for length-constrained flows is further highlighted by the fact that many classic optimization problems (such as matchings) reduce to length-constrained flows with small values of $h$.

Thus, in summary a well-studied class of routing problems aims to capture both latency and throughput concerns. These problems are known to serve as important algorithmic primitives as well as complete characterizations of the distributed complexity of many problems. However, even the simplest of these problems—maximal edge-disjoint paths—lacks good deterministic CONGEST algorithms; even worse virtually nothing is known about parallel or distributed algorithms for the maximum version of this problem and its fractional generalization, length-constrained flows.

## 1.1 Our Contributions

We give the first efficient algorithms for computing these objects in several models of computation.

### 1.1.1 Algorithms for Length-Constrained Flows

Given a digraph with $n$ nodes and $m$ arcs, our main theorem shows how to *deterministically* compute $h$-length flows that are $(1-\varepsilon)$-approximate in $\tilde{O}(\text{poly}(h, \frac{1}{\varepsilon}))$ parallel time with $m$ processors and $\tilde{O}(\text{poly}(h, \frac{1}{\varepsilon}) \cdot 2^{O(\sqrt{\log n})})$ distributed CONGEST time. We additionally give a randomized CONGEST algorithm that succeeds with high probability and runs in time $\tilde{O}(\text{poly}(h, \frac{1}{\varepsilon}))$.[1] As an immediate consequence of our parallel algorithms we also get deterministic sequential algorithms running in $\tilde{O}(m \cdot \text{poly}(h, \frac{1}{\varepsilon}))$ time; to our knowledge no such algorithms were previously known.

Additionally, our algorithms satisfy three desirable properties.

1. **General Capacities, Lengths and Multi-Commodity:** Our algorithms work for general arc capacities (i.e. connection bandwidths), general lengths (i.e. connection latencies) and multi-commodity flow variants.

2. **Dual Solution:** Not only do our algorithms compute a primal solution for length-constrained flows but they also compute a certifying dual solution; a so-called moving cut, which is an object of algorithmic utility in its own right; see, e.g. [22].

3. **Optimal Integrality:** The flows we compute are "as integral as possible." In particular, for constant $\varepsilon > 0$ (and unit capacities) they are a convex combinations of $\tilde{O}(h)$ sets of arc-disjoint paths. No near-optimal $h$-length flow can be a convex combination of $o(h)$ such sets since, by an averaging argument, this would violate the aforementioned $\Omega(h)$ integrality gap.

We give a more formal description of our results for length-constrained flows in Section 3.

### 1.1.2 Applications of our Length-Constrained Flow Algorithms

We give several applications of our length-constrained flow algorithms.

[1] We use $\tilde{O}$ notation to suppress dependence on $\text{poly}(\log n)$ factors and we use "with high probability" to mean with probability at least $1 - \frac{1}{\text{poly}(n)}$ and.

**Maximal and Maximum Edge-Disjoint Paths** First, as an almost immediate corollary of our length-constrained flow algorithms, we derive the first deterministic CONGEST algorithms for maximal edge-disjoint paths and essentially-optimal parallel and distributed algorithms for the maximum edge-disjoint paths problem as well as for many variants of these problems. The former result settles the open question of Chang and Saranurak [12]. The latter result crucially relies on the optimal integrality of our length-constrained flows and matches known hardness-of-approximation results. See Section 12 for details.

**Simpler Distributed Expander Decompositions Deterministically.** As a consequence of our maximal edge-disjoint paths algorithms, we are able to greatly simplify known distributed algorithms for deterministically computing expander decompositions.

We refer the reader to Chang and Saranurak [12] for a more thorough overview of the area, but provide a brief synopsis here. An $(\epsilon, \phi)$ expander decomposition removes an $\epsilon$ fraction of edges from a graph so as to ensure that each remaining connected component has conductance at least $\phi$. Expander decompositions have led to many recent exciting breakthroughs, including in solving linear systems [39], unique games [2, 36, 40], minimum cut [26], and dynamic algorithms [33].

Chang and Saranurak [12] gave the first deterministic CONGEST algorithms for constructing expander decompositions. However, while much of expander decomposition construction reduces to maximal edge-disjoint paths, the authors observe:

> *In the deterministic setting, we are not aware of an algorithm that can [efficiently] solve [maximal disjoint paths]... [A solution to this problem would] simplify our deterministic expander decomposition and routing quite a bit. [12]*

As a result of the lack of such algorithms, the authors employ significant technical work-arounds.

Our deterministic CONGEST algorithms for maximal edge-disjoint paths when plugged into the results of Chang and Saranurak [12] greatly simplify deterministic distributed algorithms for expander decompositions. See Section 13 for further details.

**Bipartite $b$-Matching.** Using our length-constrained flow algorithms, we give the first efficient $(1-\varepsilon)$-approximations for bipartite $b$-matching in CONGEST. $b$-matching is a classical problem in combinatorial optimization which generalizes matching where we are given a graph $G = (V, E)$ and a function $b : V \to \mathbb{Z}_{>0}$. Our goal is to assign integer values to edges so that each vertex $v$ has at most $b(v)$ assigned value across its incident edges. $b$-matching and its variants have been extensively studied in distributed settings [5, 8, 16, 16, 17, 28]. A standard folklore reduction which replaces vertex $v$ with $b(v)$ non-adjacent copies and edge $e = \{u, v\}$ with a bipartite clique between the copies of $u$ and $v$ reduces $b$-matching to matching but requires overhead $\max_{\{u,v\}\in E} b(u) \cdot b(v)$ to run in CONGEST. Thus, the non-trivial goal here is a CONGEST algorithm whose running time does not depend on $b$. Currently, the best algorithm in CONGEST is a $(\frac{1}{2} - \varepsilon)$-approximation of Fischer [16] running in time $\tilde{O}(\text{poly}(\log \frac{1}{\varepsilon}))$.

Similarly to classical matching, it is easy to reduce bipartite $b$-matching to an $O(1)$-length flow problem. Thus, applying our algorithms for length-constrained flows and some of the flow rounding techniques we develop in this work allows us to give the first $(1-\varepsilon)$-approximation for $b$-matching in bipartite graphs running in CONGEST time $\tilde{O}(\text{poly}(\frac{1}{\varepsilon}) \cdot 2^{O(\sqrt{\log n})})$. Our algorithms are deterministic though similar results even for the randomized setting do not seem to be known. See Section 14 for further details.

**Length-Constrained Cutmatches.** Lastly, our results allow us to give the first efficient constructions of length-constrained cutmatches. Informally, an $h$-length cutmatch with congestion $\gamma$ is a collection of $h$-length $\gamma$-congestion paths between two vertex subsets along with a moving cut that shows that adding any more $h$-length paths to this set would incur congestion greater than $\gamma$. See Section 15 for details.

A recent work [23] uses our algorithms for length-constrained cutmatches to give the first efficient constructions of a length-constrained version of expander decompositions. This work, in turn, uses these constructions to, among other things, give CONGEST algorithms for many problems including MST, $(1+\epsilon)$-min-cut and $(1+\epsilon)$-shortest paths that are guaranteed to run in sub-linear rounds as long as such algorithms exist on the input network.

## 2 Notation and Conventions

Before moving on to a formal statement of length-constrained flows, moving cuts and our results we introduce some notation and conventions. Suppose we are given a digraph $D = (V, A)$.

**Digraph Notation.** We will associate three functions with the arcs of $D$. We clarify these here.

1. **Lengths:** We will let $\ell = \{\ell_a\}_a$ be the *lengths* of arcs in $A$. These lengths will be input to our problem and determine the lengths of paths when we are computing length-constrained flows. Throughout this work we imagine each $\ell_a$ is in $\mathbb{Z}_{>0}$. Informally, one may think of $\ell$ as giving link latencies. We will assume $\ell_a$ is always $\text{poly}(n)$.

2. **Capacities:** We will let $U = \{U_a\}_a$ be the capacities of arcs in $A$. These capacities will specify a maximum amount of flow (either length-constrained or not) that is allowed over each arc. Throughout this work we imagine each $U_a$ is in $\mathbb{Z}_{\geq 0}$ and we let $U_{\max}$ give $\max_a U_a$. We will assume $U_{\max}$ is $\text{poly}(n)$. Informally, one may think of $U$ as link bandwidths.

3. **Weights:** We will let $w = \{w_a\}_a$ stand for the weights of arcs in $A$. These weights will be given by our moving cut solutions. Throughout this work each $w_a$ will be in $\mathbb{R}_{>0}$.

In general we will treat a path $P = ((v_1, v_2), (v_2, v_3), \ldots)$ as series of consecutive arcs in $A$ (all oriented consistently towards one endpoint). For any one of these weighting functions $\phi \in \{\ell, U, w\}$, we will let $d_\phi(u, v)$ give the minimum value of a path in $D$ that connects $u$ and $v$ where the value of a path $P$ is $\phi(P) := \sum_{a \in P} \phi(a)$. That is, we think of $d_\phi(u, v)$ as the distance from $u$ to $v$ with respect to $\phi$. We will refer to paths which minimize $w$ as lightest paths (so as to distinguish them from e.g. shortest paths with respect to $\ell$).

We let $\delta^+(v) := \{a : a = (v, u)\}$ and $N^+(v) := \{u : (v, u) \in A\}$ give the out arcs and out neighborhoods of vertex $v$. $\delta^-(v) := \{a : a = (u, v)\}$ and $N^- := \{u : (u, v) \in A\}$ are defined symmetrically. We let $\mathcal{P}(u, v)$ be all simple paths between $u$ and $v$ and for $W, W' \subseteq V$, we let $\mathcal{P}(W, W') := \bigcup_{w \in W, w' \in W'} \mathcal{P}(w, w')$ give all paths between vertex subsets $W$ and $W'$.

Given sources $S \in V$ and sinks $T \in V$, we say that $D$ is an $S$-$T$ DAG if $\delta^-(v) = \emptyset$ iff $v \in S$ and $\delta^+(v) = \emptyset$ iff $v \in T$. We say that such an $S$-$T$ DAG is an $h$-layer DAG if the vertex set $V$ can be partitioned into $h + 1$ layers $S = V_1 \sqcup V_2 \sqcup \ldots \sqcup V_{h+1} = T$ where any arc $a = (u, v)$ is such that $u \in V_i$ and $v \in V_j$ for some $i$ and $j > i$. We say that $D$ has diameter at most $d$ if in the graph where we forget about arc directions in $D$ every pair of vertices is connected by a path of at most $d$

edges. Notice that the diameter of an $h$-layer $S$-$T$ DAG might be much larger than $h$, for example, when $S$ and $T$ are large sets of vertices.

For a (di)graph $D = (V, A)$ and a collection of subgraphs $\mathcal{H}$ of $D$, we let $D[\mathcal{H}]$ be the graph induced by the union of all elements of $\mathcal{H}$. $A[\mathcal{H}]$ is defined as all elements of $A$ which are contained in some element of $\mathcal{H}$.

**(Non-Length Constrained) Flow Notation and Conventions.** We will make extensive use of non-length constrained flows and so clarify our notation for such flows here.

Given a DAG $D = (V, A)$ with capacities $U$ we will let a flow $f$ be any assignment of non-negative values to arcs in $a$ where $f_a$ gives the value that $f$ assigns to $a$ and $f_a \leq U_a$ for every $a$. If it is ever the case that $f_a > U_a$ for some $a$, we will explicitly state that this "flow" does not respect capacities. We say that $f$ is an integral flow if it assigns an integer value to each arc. We let $f(A') := \sum_{a \in A'} f_a$ for any $A' \subseteq A$. We define the deficit of a vertex $v$ as $\text{deficit}(f, v) := |\sum_{a \in \delta^+(f,v)} f_a - \sum_{a \in \delta^-(v)} f_a|$. We will let $\text{supp}(f) := \{a : f_a > 0\}$ give the support of flow $f$.

Given desired sources $S$ and sinks $T$, we let $\text{deficit}(f) := \sum_{v \notin S \cup T} \text{deficit}(f, v)$ be the total amount of flow produced but not at $S$ plus the amount of flow consumed but not at $T$; likewise, we say that a flow $f$ is an $S$-$T$ flow if $f\text{deficit}(f) = 0$. We let $\text{val}(f) = \bigcup_{s \in S} f(\delta^+(s))$ be the amount of flow delivered by an $S$-$T$ flow $f$ and we say that $f$ is $\alpha$-approximate if $\text{val}(f) \geq \alpha \cdot \text{val}(f^*)$ where $f^*$ is the $S$-$T$ flow that maximizes val. We say that $f$ is $\alpha$-blocking for $\alpha \in [0, 1]$ if for every path from $S$ to $T$ there is some $a \in P$ where $f_a \geq \alpha \cdot U_a$. We say that a 1-blocking flow is blocking. We say that flow $f'$ is a subflow of $f$ if $f'_a \leq f_a$ for every $a$.

Given a maximum capacity of $U_{\max}$, we may assume that every flow $f$ is of the form $f = \sum_i f^{(i)}$ where $(f^{(i)})_a \in \{0, 2^{\log(U_{\max})-i}\}$ for every $a$ and $i$; that is, a given flow can always be decomposed into its values on each bit. We call $f^{(i)}$ the $i$th bit flow of $f$ and call the decomposition of $f$ into these flows be the bitwise decomposition of $f$.

**Length-Constrained Notation.** Given a length function $\ell$, vertices $u, v \in V$ and length constraint $h \geq 1$, we let $\mathcal{P}_h(u, v) := \{P \in \mathcal{P}(u, v) : \ell(P) \leq h\}$ be all paths between $u$ and $v$ which have length at most $h$. For vertex sets $W$ and $W'$, we let $\mathcal{P}_h(W, W') := \{P \in \mathcal{P}(W, W') : \ell(P) \leq h\}$. If $G$ also has weights $w$ then we let $d^{(h)}_w(u, v) := \min_{P \in \mathcal{P}_h(u,v)} w(P)$ give the minimum weight of a length at most $h$ path connecting $u$ and $v$. For vertex sets $W, W' \subseteq V$ we define $d^{(h)}_w(W, W') := \min_{P \in \mathcal{P}_h(W,W')} w(P)$ analogously. As mentioned an $h$-length path is a path of length at most $h$.

**Parallel and Distributed Models.** Throughout this work the parallel model of computation we will make use of is the EREW PRAM model [25]. Here we imagine that we are given some number of processors as well as shared random access memory; every memory cell can be read or written to by only one processor at a time.

The distributed model we will make use of is the CONGEST model, defined as follows [35]. The network is modeled as a graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Communication is conducted over discrete, synchronous rounds. During each round each node can send an $O(\log n)$-bit message along each of its incident edges. Every node has an arbitrary and unique ID of $O(\log n)$ bits, first only known to itself. The running time of a CONGEST algorithm is the number of rounds it uses. We will slightly abuse terminology and talk about running a CONGEST algorithm in digraph $D$; when we do so we mean that the algorithm runs in the (undirected) graph $G$ which is identical to $D$ but where we forget the directions of arcs. In this work, we will assume that if an

arc $a$ has capacity $U_a$ then we allow nodes to send $O(U_a \cdot \log n)$ bits over the corresponding edge, though none of our applications rely on this assumption.[2]

# 3 Length-Constrained Flows, Moving Cuts and Main Result

We proceed to more formally define a length-constrained flow, moving cuts and our main result which computes them. While we have defined length-constrained flows in Section 1 for unit capacities, it will be convenient for us to formally define length-constrained flows for general lengths and capacities in terms of a relevant linear program (LP). We do so now.

Suppose we are given a digraph $D = (V, A)$ with arc capacities $U$, lengths $\ell$ and specified source and sink vertices $S$ and $T$. A maximum $S$ to $T$ flow in $D$ in the classic sense can be defined as a collection of paths between $S$ and $T$ where each path receives some value and the total value incident to an edge does not exceed its capacity. This definition naturally extends to the length-constrained setting where we imagine we are given some length constraint $h \geq 1$ and define a length-constrained flow as a collection of $S$ to $T$ paths each of length at most $h$ where each such path $P$ receives some some value $f_P$. Additionally, these values must respect the capacities of arcs. More precisely, we have the following LP with a variable $f_P$ for each path $P \in \mathcal{P}_h(s,t)$.

$$\max \sum_{P \in \mathcal{P}_h(S,T)} f_P \quad \text{s.t.} \qquad \text{(Length-Constrained Flow LP)}$$

$$\sum_{P: a \in P} f_P \leq U_a \qquad \forall a \in A$$

$$0 \leq f_P \qquad \forall P \in \mathcal{P}_h(s,t)$$

For a length-constrained flow $f$, we will use the shorthand $f(a) := \sum_{P \ni a} f_P$ and $\text{supp}(f) := \{P : f_P > 0\}$ to give the support of $f$. We will let $\text{val}(f) := \sum_{P \in \mathcal{P}_h(s,t)} f_P$ give the value of $f$. An $h$-length flow, then, is simply a feasible solution to this LP.

**Definition 3.1** ($h$-Length Flow)**.** *Given digraph $D = (V, A)$ with lengths $\ell$, capacities $U$ and vertices $S, T \subseteq V$, an $h$-length $S$-$T$ flow is any feasible solution to Length-Constrained Flow LP.*

With the above definition of length-constrained flows we can now define moving cuts as the dual of length-constrained flows. In particular, taking the dual of the above LP we get the moving cut LP with a variable $w_a$ for each $a \in A$.

$$\min \sum_{a \in A} U_a \cdot w_a \quad \text{s.t.} \qquad \text{(Moving Cut LP)}$$

$$\sum_{a \in P} w_a \geq 1 \qquad \forall P \in \mathcal{P}_h(S,T)$$

$$0 \leq w_a \qquad \forall a \in A$$

An $h$-length moving cut is simply a feasible solution to this LP.

**Definition 3.2** ($h$-Length Moving Cut)**.** *Given digraph $D = (V, A)$ with lengths $\ell$, capacities $U$ and vertices $S, T \subseteq V$, an $h$-length moving cut is any feasible solution to Moving Cut LP.*

[2] We only make use of this assumption once and only make use of it in our deterministic algorithms (in Lemma 10.3). Furthermore, we do not require this assumption if the underlying digraph is a DAG.

We will use $f$ and $w$ to stand for solutions to Length-Constrained Flow LP and Moving Cut LP respectively. We say that $(f, w)$ is a feasible pair if both $f$ and $w$ are feasible for their respective LPs and that $(f, w)$ is $(1 \pm \epsilon)$-approximate for $\epsilon \geq 0$ if the moving cut certifies the value of the length-constrained flow up to a $(1 - \epsilon)$; i.e. if $(1 - \epsilon) \sum_a U_a \cdot w_a \leq \sum_P f_P$.

We clarify what it means to compute $(f, w)$ in CONGEST. When we are working in CONGEST we will say that $f$ is computed if each vertex $v$ stores the value $f_a(h') := \sum_{P \in \mathcal{P}_{h,h'}(s,a,t)} f_P$ for every $a \in A$ and $h' \leq h$. Here, we let $\mathcal{P}_{h,h'}(s, a, t)$ be all paths in $\mathcal{P}_h(s, t)$ of the form $P' = (a_1, a_2, \ldots a, b_1, b_2, \ldots)$ where the path $(a, b_1, b_2, \ldots)$ has length exactly $h'$ according to $\ell$. We say moving cut $w$ is computed if each vertex $v$ knows the value of $w_a$ for its incident arcs. Likewise, we imagine that each node initially knows the capacities and lengths of its incident arcs.

With the above notions, we can now state our main results which say that one can efficiently compute a feasible pair $(f, w)$ in parallel and distributedly. In the following we say $f$ is integral if $f_P$ is an integer for every path in $\mathcal{P}_h(S, T)$. The notable aspect of our results is the polynomial dependence on $h$ and $\frac{1}{\epsilon}$; the polynomials could be optimized to be much smaller.

**Theorem 3.1.** *Given a digraph $D = (V, A)$ with capacities $U$, lengths $\ell$, length constraint $h \geq 1$, $\varepsilon > 0$ and source and sink vertices $S, T \subseteq V$, one can compute a feasible $h$-length flow, moving cut pair $(f, w)$ that is $(1 \pm \epsilon)$-approximate in:*

1. *Deterministic parallel time $\tilde{O}(\frac{1}{\varepsilon^9} \cdot h^{17})$ with $m$ processors*
2. *Randomized CONGEST time $\tilde{O}(\frac{1}{\varepsilon^9} \cdot h^{17})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\varepsilon^9} \cdot h^{17} + \frac{1}{\varepsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.*

*Also, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$ and each $f_j$ is an integral $h$-length $S$-$T$ flow.*

All of our algorithms compute and separately store each $f_j$. The above result immediately gives the deterministic parallel and randomized CONGEST algorithms running in time $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}))$ mentioned in Section 1.1. For our deterministic CONGEST algorithms, $\rho_{CC}$ in the above gives the quality of the optimal deterministic CONGEST cycle cover algorithm. We formally define this parameter in Section 5 but for now we simply note that $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$ by known results [24, 34]. Applying this bound on $\rho_{CC}$ gives deterministic CONGEST algorithms running in time $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}) \cdot 2^{O(\sqrt{\log n})})$. If $\rho_{CC}$ is shown to be $\text{poly}(\log n)$, we immediately would get an $\tilde{O}(\text{poly}(h, \frac{1}{\epsilon}))$ time deterministic algorithm for solving $(1 - \epsilon)$-approximate $h$-length flow in CONGEST. Also, as mentioned in Section 1.1, $k$ in the above result is optimal up to $\tilde{O}(1)$ factors by the results of Guruswami et al. [19] and Baier et al. [4].

# 4 Intuition and Overview of Approach

Before moving on to details, we give an overview of our strategy for computing length-constrained flows. For simplicity, we will assume that the capacity $U_a$ of arc $a$ is 1 in this section.

## 4.1 Using Lightest Path Blockers for Multiplicative Weights

Computing a length-constrained flow, moving cut pair is naturally suggestive of the following multiplicative-weights-type approach. We initialize our moving cut value $w_a$ to some very small

value for every $a$. Then, we find a lightest $h$-length path from $S$ to $T$ according to $w$, send some small ($\approx \epsilon$) amount of flow along this path and multiplicatively increase the value of $w$ on all arcs in this path by $\approx (1+\epsilon)$. We repeat this until $S$ and $T$ are at least 1 apart according to $d_w^{(h)}$.

The principle shortcoming of such an algorithm is that it is easy to construct examples where there are polynomially-many arc-disjoint $h$-length paths between $S$ and $T$ and so we would clearly have to repeat the above process at least polynomially-many times until $S$ and $T$ are at least 1 apart according to $d_w^{(h)}$. This is not consistent with our goal of poly($h$) complexities since $h$ may be much smaller than $n$. To solve this issue, we use an algorithm similar to the above but instead of sending flow along a single path at a time, we send it along a large batch of arc-disjoint paths.

What can we hope to say about how long such an algorithm takes to make $S$ and $T$ at least 1 apart according to $d_w^{(h)}$? If it were the case that every lightest (according to $w$) $h$-length path from $S$ to $T$ shared an arc with some path in our batch of paths then after each batch we would know that we increased $d_w^{(h)}(S,T)$ by some non-zero amount. However, there is no way to lower bound this amount; in principle we might only increase $d_w^{(h)}(S,T)$ by some tiny $\epsilon' > 0$. To solve this issue we find a batch of arc-disjoint paths which have weight essentially $d_w^{(h)}(S,T)$ but which share an arc with every $h$-length path with weight at most $(1+\epsilon) \cdot d_w^{(h)}(S,T)$. Thus, when we increment weights in our batch we know that all *near*-lightest $h$-length paths have their weights incremented and this, in turn, allows us to lower bound the rate at which $d_w^{(h)}(S,T)$ increases and therefore to argue that our algorithm completes quickly.

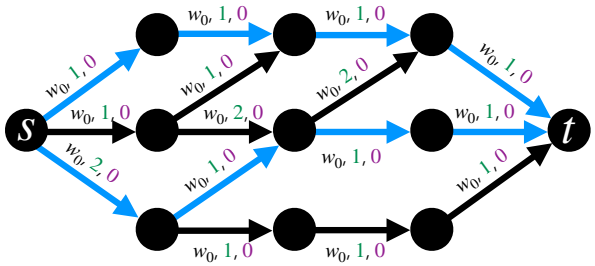
Thus, in summary we repeatedly find a batch of arc-disjoint $h$-length paths between $S$ and $T$ which have weight about $d_w^{(h)}(S,T)$; these paths satisfy the property that every $h$-length path from $S$ to $T$ with weight at most $(1+\epsilon) \cdot d_w^{(h)}(S,T)$ shares an edge with at least one of these paths; we call such a collection an $h$-length $(1+\epsilon)$-lightest path blocker. We then send a small amount of flow along these paths and multiplicatively increase the weight of all incident edges, appreciably increasing $d_w^{(h)}(S,T)$. We repeat this until our weights form a feasible moving cut. See Figure 1.

### 4.2 Length-Weight Expanded DAG to Approximate $h$-Length Lightest Paths

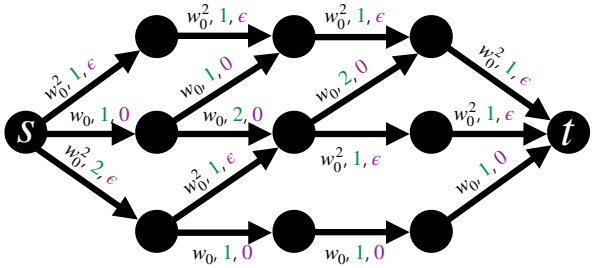
The above strategy relies on the computation of $h$-length lightest path blockers. Without the presence of a weight constraint computing such an object easily reduces to computing an integral blocking $S$-$T$ flow on an $h$-layer $S$-$T$ DAG. Specifically, consider the problem of computing a collection of paths from $S$ to $T$ so that every $h$-length $S$ to $T$ path shares an arc with one path in this collection. It is easy to see that all paths of length at most $h$ between $S$ and $T$ induce an $h$-layer $S$-$T$ DAG. One can then consider this DAG and compute an integral blocking $S$-$T$ flow in it—i.e. a maximal arc-disjoint collection of $h$-length $S$-$T$ paths. By maximality of the flow, the paths corresponding to this flow will guarantee that every $h$-length $S$ to $T$ path shares an arc with one path in this collection.

However, when we are working in the presence of both a length constraint and weight constraint computing such an object becomes significantly more tricky. Indeed, lightest paths subject to length constraints are known to be notoriously poorly behaved; not only do lightest paths subject to a length constraint not induce a metric but they are also arbitrarily far from any metric [1, 21]. As a consequence of this, all lightest paths subject to a length constraint from $S$ to $T$ do not induce a DAG, much less an $h$-layer $S$ to $T$ DAG; see Figure 2 for an example.

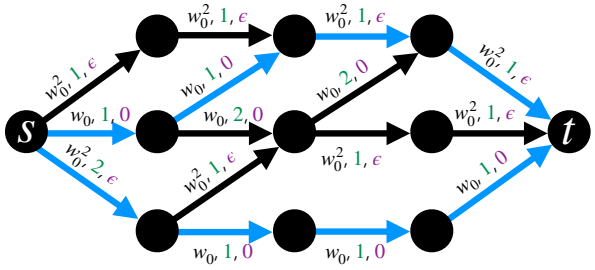
Our solution to this issue is to observe that, if we are allowed to duplicate vertices, then we



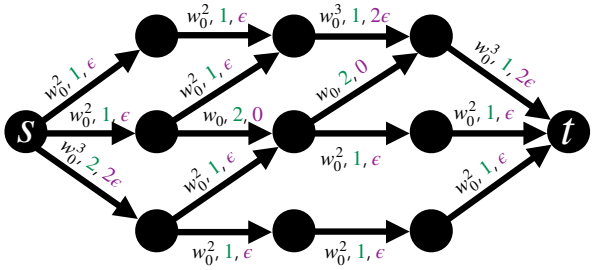

(a) Compute lightest path blocker.




(b) Update flow and weights.




(c) Compute lightest path blocker.




(d) Update flow and weights.

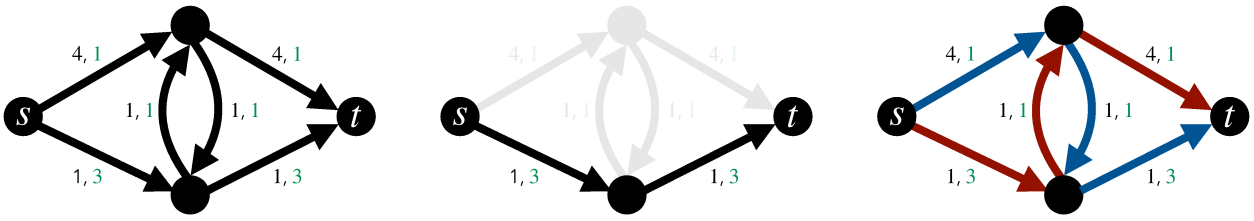
Figure 1: An illustration of the first two iterations of our multiplicative-weights-type algorithm where $h = 5$, $S = \{s\}$ and $T = \{t\}$ and capacities are all 1. Each arc is labelled with its weight (initialized to $w_0 := 1 + \epsilon$) then length then flow. Our $h$-length shortest path blockers are in blue.

can construct an $S$-$T$ DAG with about $h^2$ layers that approximately captures the structure of all $h$-length $(1+\varepsilon)$-lightest paths. Specifically, we discretize weights and then make a small number of copies of each vertex to compute a DAG $D^{(h,\lambda)}$—which we call the length-weight expanded DAG. $D^{(h,\lambda)}$ will satisfy the property that if we compute an integral blocking flow in it and then project this back into $D$ as a set of paths $\mathcal{P}$, then $\mathcal{P}$ is *almost* a $(1+\varepsilon)$-lightest path blocker. In particular, $\mathcal{P}$ will guarantee that some arc of any $h$-length path with weight at most $(1+\varepsilon) \cdot d_w^{(h)}(S,T)$ is used by some path in $\mathcal{P}$; however, the paths of $\mathcal{P}$ may not be arc-disjoint which is required of our lightest path blockers. Nonetheless, by carefully choosing the capacities in $D^{(h,\lambda)}$, we will be able to argue that $\mathcal{P}$ is nearly arc-disjoint and these violations of arc-disjointness can be repaired with bounded loss by a "decongesting" procedure. Summarizing, these ideas reduce computing $h$-length $(1+\varepsilon)$-lightest path blockers to computing integral blocking flows in layered $S$-$T$ DAGs.

### 4.3 Deterministic Integral Blocking Flows Paths via Flow Rounding

Lastly, we describe how we compute integral blocking flows in layered $S$-$T$ DAGs.

A somewhat straightforward adaptation of a *randomized* algorithms of Lotker et al. [30] solves this problem in $\tilde{O}(\text{poly}(h))$ time both in parallel and in CONGEST. This algorithm samples an integral $S$-$T$ flow in $D$ (i.e. a collection of arc-disjoint $S$ to $T$ paths) according to a carefully chosen distribution based on "path counts", deletes these paths and repeats. The returned solution is the




(a) Digraph $D$. (b) Lightest paths of $D$. (c) Lightest 5-length paths of $D$.

Figure 2: A digraph $D$ with $S = \{s\}$ and $T = \{t\}$ where the 5-length lightest $S$-$T$ paths do not induce a DAG. 2a gives $D$ where each arc is labeled with its weight (in black) and length (in green). 2b shows how all lightest $S$-$T$ paths have weight 2 and induce a DAG. 2c shows how the two 5-length lightest $S$-$T$ paths (in blue and red) have weight 6 and induce a digraph with a cycle.

flow induced by all paths that were ever deleted. Unfortunately Lotker et al. [30]'s algorithm seems inherently randomized and our goal is to solve this problem deterministically.

We derandomize the algorithm of Lotker et al. [30] in the following way. Rather than integrally sampling according to Lotker et al. [30]'s distribution and then deleting arcs that appear in sampled paths, we instead calculate the probability that an arc is in a path in this distribution and then "fractionally delete" it to this extent. We repeat this until every path between $S$ and $T$ has some arc which has been fully deleted. In other words, we run a smoothed version of Lotker et al. [30] which behaves (deterministically) like the algorithm of Lotker et al. [30] does in expectation. A simple counting argument shows that we need only iterate this process about $h$ times to separate $S$ and $T$. The fractional deletion values of arcs at the end of this process induce a blocking $S$-$T$ flow but a blocking flow that *may be fractional*. We call this flow the "iterated path count flow."

However, recall that our goal is to compute an *integral* blocking flow in an $S$-$T$ DAG. Thus, we may naturally hope to round the iterated path count flow. Indeed, drawing on some flow rounding techniques of Cohen [15], doing so is not too difficult in parallel. Unfortunately, it is less clear how to do so in CONGEST. Indeed, Chang and Saranurak [12] state:

> *...Cohen's algorithm that rounds a fractional flow into an integral flow does not seem to have an efficient implementation in CONGEST...*

Roughly, Cohen's technique relies on partitioning edges in a graph into cycles and paths and then rounding each cycle and path independently. The reason this seems infeasible in CONGEST is that the cycles and paths that Cohen's algorithm relies on can have unbounded diameter and so communicating within one of these cycles or paths is prohibitively slow. To get around this, we argue that, in fact, one may assume that these cycles and paths have low diameter *if* we allow ourselves to discard some small number of arcs. This, in turn allows us to orient these cycles and paths and use them in rounding flows. We formalize such a decomposition with the idea of a $(1-\varepsilon)$-near Eulerian partition. Arguing that discarding these arcs does bounded damage to our rounding then allows us to make use of Cohen-type rounding to deterministically round the path count flow, ultimately allowing us to compute $h$-length $(1+\epsilon)$-lightest path blockers.

### 4.4 Overview of Paper

In Section 6 we more formally define path counts. In Section 7 we describe how to use these path counts with randomization to compute integral blocking flows in an $h$-layer $S$-$T$ DAG. In Section 9

we do the same but deterministically, employing the above flow rounding strategy and the idea of near Eulerian partitions as introduced and constructed in Section 8. Next, in Section 10 we show how to use these blocking flow primitives along with our length-weight expanded DAG to compute $(1+\epsilon)$-lightest path blockers. In Section 11 we formalize how to use these $(1+\epsilon)$-lightest path blockers to compute length-constrained flows and moving cuts by applying multiplicative-weights-types arguments, thereby showing our main result.

In Section 12 we observe that our main result solves the aforementioned problem of Chang and Saranurak [12] by giving deterministic algorithms for many disjoint paths problems in CONGEST. We also observe that our algorithms give essentially optimal parallel and distributed algorithms for maximum arc-disjoint paths. In Section 13 we give more details of how our results simplify expander decomposition constructions. In Section 14 we give our new algorithms for bipartite $b$-matching based on our flow algorithms and in Section 15 we show how to compute length-constrained cutmatches using our main theorem. Lastly, in Appendix A we observe that our length-constrained flow algorithms generalize to the multi-commodity setting.

# 5 Preliminaries

Before moving on to our own technical content, we briefly review some well-known algorithmic tools and slight variants thereof (mostly for deterministic CONGEST).

## 5.1 Deterministic CONGEST Maximal and Maximum Independent Set

We will rely on deterministic CONGEST primitives for maximal and maximum independent sets. Given graph $G = (V, E)$, a subset of vertices $V' \subseteq V$ is independent if no two vertices in $V'$ are adjacent in $G$. A maximal independent set (MIS) is an independent set $V'$ such that any $w \in V \setminus V'$ is adjacent to at least one node in $V'$. If we are additionally given node weights $\{x_v\}_v$ where $x_v > 0$ for every $v$, then a maximum independent set is an independent set $V'$ maximizing $\sum_{v \in V'} x_v$; we say that an independent set is $\alpha$-approximate if its total weight is within $\alpha$ of that of the maximum independent set.

The following summarizes the deterministic CONGEST algorithm we will use for MIS.

**Theorem 5.1** (Censor-Hillel et al. [10])**.** *There is a deterministic CONGEST algorithm which given a graph $G = (V, E)$ with diameter $D$, outputs a maximal independent set in time $O(D \cdot \log^2 n)$.*

The following gives the deterministic CONGEST algorithm we will use for maximum independent set.

**Theorem 5.2** (Bar-Yehuda et al. [6])**.** *There is a deterministic CONGEST algorithm which given an instance of maximum independent in a graph $G = (V, E)$ with maximum degree $\Delta$ and node weights $\{x_v\}_v$, outputs a solution that is $\frac{1}{\Delta}$-approximate in time $O(\Delta + \log^* n)$.*

## 5.2 Deterministic Low Diameter Decompositions

A well-studied object in metric theory is the low diameter decomposition which is usually defined as a distribution over vertex partitions [29, 32]. For our deterministic algorithms, we will make use of a deterministic version of these objects defined as follows where $G[V_i] := (V_i, \{\{u, v\} \in E : u, v \in V_i\})$ gives the induced graph on $V_i$.

**Definition 5.3** (Deterministic Low Diameter Decomposition)**.** *Given graph $G = (V, E)$, a deterministic low diameter decomposition (DLDD) with diameter $d$ and cut fraction $\epsilon$ is a partition of $V$ into sets $V_1, V_2, \ldots$ where:*

1. ***Low Diameter:*** *$G[V_i]$ has diameter at most $d$ for every $i$;*
2. ***Cut Edges:*** *The number of cut edges is at most $\epsilon|E|$; i.e. $|\{e = (u, v) : u \in V_i \wedge v \in V_j \wedge i \neq j\}| \leq \epsilon|E|$.*

One can efficiently compute DLDDs deterministically in CONGEST as a consequence of many well-known results in distributed computing. We will use a result of Chang and Ghaffari [11] to do so.

**Theorem 5.4.** *Given a graph $G = (V, E)$ and desired diameter $d$, one can compute a DLDD with diameter $d$ and cut fraction $\epsilon = \tilde{O}(\frac{1}{d})$ in deterministic CONGEST time $\tilde{O}(d)$.*

*Proof.* Theorem 1.2 of Chang and Ghaffari [11] states that there is a deterministic CONGEST algorithm which, given a graph $G = (V, E)$ and desired diameter $d'$, computes a set $\bar{V} \subseteq V$ where $|\bar{V}| \leq \frac{1}{d'} \cdot |V|$ and $G[V \setminus \bar{V}]$ has connected components $C_1, C_2, \ldots, C_k$ where each $C_i$ has diameter at most $\tilde{O}(d')$ in $\tilde{O}(d')$ rounds.

Given graph $G = (V, E)$ we can compute a DLDD in $G$ by applying the above result in a new graph $G' = (V', E')$. For each vertex $v \in V$, $G'$ will have a clique of $\Delta(v)$-many vertices where $\Delta(v)$ is the degree of $v$ in $G$. We then connect these cliques in the natural way. More formally, to construct $G'$ we do the following. For each $v$ with edges to vertices $v_1, v_2, \ldots, v_{\Delta(v)}$ we create a clique of vertices $v(v_1), v(v_2), \ldots, v(v_{\Delta(v)})$. Next, for each edge $e = \{u, v\}$ in $E$, we add the edge $\{v(u), u(v)\}$ to $G'$. Observe that each vertex of $G'$ corresponds to exactly one edge in $G$; that is, $v(u)$ in $V'$ corresponds to the edge $\{u, v\} \in E$.

Next, we apply the above theorem of Chang and Ghaffari [11] to $G'$ to get set $\bar{V}$. Let $\bar{E} \subseteq E$ be the set of edges to which these vertices correspond. We return as our solution $\bar{E}$. Observe that the size of $\bar{E}$ is

$$\begin{aligned} |\bar{E}| &\leq |\bar{V}| \\ &\leq \frac{1}{d'} \cdot |V'| \\ &= \frac{2}{d'}|E|. \end{aligned}$$

Letting $d' = \frac{1}{\tilde{\Theta}(1)} \cdot d$ for an appropriately large hidden poly-log in $\tilde{\Theta}(1)$ gives us that each component in $G$ has diameter at most $d$ since otherwise there would be a component in $G'$ after deleting $\bar{v}$ with diameter more than $d'$. Likewise, the above gives us cut fraction at most $\tilde{O}(\frac{1}{d})$.

Simulating a CONGEST algorithm on $G'$ on $G$ is trivial since each vertex can simulate its corresponding clique and so the entire algorithm runs in time $\tilde{O}(d') = \tilde{O}(d)$. □

## 5.3 Sparse Neighborhood Covers

A closely related notion to low diameter decompositions is that of the sparse neighborhood cover [3]. We use the following definition phrased in terms of partitions.

**Definition 5.5** (Sparse Neighborhood Cover)**.** *Given a simple graph $G = (V, E)$, an $s$-sparse $k$-neighborhood cover with weak-diameter $d$ and overlap $o$ is a set of partitions $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_s$ of $V$ where each partition is a collection of disjoint vertex sets $V_i^{(j)} \subset V$ whose union is $V$, i.e., $\mathcal{V}_i = \{V_i^{(1)}, V_i^{(2)}, \ldots\}$ and:*

1. ***Weak-Diameter and Overlap:*** *Each $V_i^{(j)}$ comes with a rooted tree $T_i^{(j)}$ in $G$ of diameter at most $d$ that spans all nodes in $V_i^{(j)}$; Any node in $G$ is contained in at most $o$ trees overall.*
2. ***Neighborhood Covering:*** *For every node $v$ its $k$ neighborhood $B_k(v)$, containing all vertices in $G$ within distance $k$ of $v$, is fully covered by at least one cluster, i.e., $\forall v\ \exists i, j:\ B_k(v) \subseteq V_i^{(j)}$.*

The below summarizes the current state of the art in deterministic sparse neighborhood covers in CONGEST.

**Lemma 5.6** ([11, 18, 37])**.** *There is a deterministic CONGEST algorithm which given any radius $r \geq 1$, computes an $s$-sparse $r$-neighborhood cover with $s, o = \tilde{O}(1)$ and diameter at most $\tilde{O}(r)$ in $\tilde{O}(r)$ time.*

*Furthermore, there is a deterministic CONGEST algorithm which given an $O(k)$-bit value $x_v$ for every $v$ computes $x_{i,v}$ for every $v$ and $i$ in $\tilde{O}(r + k)$ rounds, where $x_{i,v}$ is the maximum $x$-value among nodes in the same cluster as $v$ in the partition $\mathcal{V}_i$. That is, letting $\mathcal{V}_i(v)$ be the one cluster in $\mathcal{V}_i$ containing $v$, we have*

$$x_{i,v} = \max_{u \in \mathcal{V}_i(v)} x_u.$$

## 5.4 Cycle Covers

Our flow rounding algorithm will make use of low diameter cycles. Thus, it will be useful for us to make use of some recent insights into distributely and deterministically decomposing graphs into low diameter cycles. We define the diameter of a cycle $C$ as $|C|$ and the diameter of a collection of cycles $\mathcal{C}$ as the maximum diameter of any cycle in it. Likewise the congestion of $\mathcal{C}$ is $\max_e |\{C : e \in C\}|$.

The idea of covering a graph with low congestion cycles is well-studied [13, 24, 34] and formalized by the idea of a cycle cover.

**Definition 5.7** (Cycle Cover)**.** *Given a simple graph $G = (V, E)$ where $E_0$ are all non-bridge edges[3] of $G$, a $(d, c)$ cycle cover is a collection of (simple) cycles $\mathcal{C}$ in $G$ such that:*

1. ***Covering:*** *Every $e \in E_0$ is contained in some cycle of $\mathcal{C}$;*
2. ***Low Diameter:*** $\max_{C \in \mathcal{C}} |C| \leq d$*;*
3. ***Low Congestion:*** $\max_{e \in E} |\{C : e \in C\}| \leq c$.

We now formally define the parameter $\rho_{CC}$; recall that this parameter appears in the running time of our deterministic CONGEST algorithm in our main theorem (Theorem 3.1).

[3]Recall that a bridge edge of a graph is one whose removal increases the number of connected components in the graph.

**Definition 5.8** ($\rho_{CC}$)**.** *Given a deterministic CONGEST algorithm that constructs a $(d, c)$ cycle cover in worst-case time $T$ in graphs of diameter $D$, we say that the quality of this algorithm is $\max\{\frac{d}{D}, c, \frac{T}{D}\}$. We let $\rho_{CC}$ be the smallest quality of any deterministic CONGEST algorithm for constructing cycle covers.*

The following summarizes the current state-of-the-art in deterministic cycle cover computation in CONGEST.

**Theorem 5.9** ([24, 34])**.** *There is a deterministic CONGEST algorithm that given a graph $G$ with diameter $D$ computes a $(d, c)$ cycle cover with $d = 2^{O(\sqrt{\log n})} \cdot D$ and $c = 2^{O(\sqrt{\log n})}$ in time $2^{O(\sqrt{\log n})} \cdot D$. In other words, $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$*

# 6 Path Counts for $h$-Layer $S$-$T$ DAGs

We begin be recounting the notion idea of path counts which we will use for our randomized algorithm to sample flows and for our deterministic algorithms to compute the iterated path count flow. This idea has been used in several prior works [12, 15, 30].

Suppose we are given an $h$-layer $S$-$T$ DAG $D$ with capacities $U$. We define these path counts as follows. We define the capacity of a path as the product of its edge capacities, namely given a path $P$ we let $U(P) := \prod_{a \in P} U_a$. Recall that we use $\mathcal{P}(S, T)$ to stand for all paths between $S$ and $T$. We will slightly abuse notation and let $\mathcal{P}(v, T) = \mathcal{P}(\{v\}, T)$ and $\mathcal{P}(S, v) = \mathcal{P}(S, \{v\})$. For vertex $v$ we let $n_v^+$ be the number of paths from $v$ to $T$, weighted by $U$, namely $n_v^+ := \sum_{P \in \mathcal{P}(v,T)} U(P)$. Symmetrically, we let $n_v^- := \sum_{P \in \mathcal{P}(S,v)} U(P)$. For any arc $a = (u, v)$, we define $n_a$ as

$$n_a := n_u^- \cdot U_a \cdot n_v^+.$$

Equivalently, we have that $n_a$ is the number of paths in $\mathcal{P}(S, T)$ that use $a$ weighted by capacities:

$$n_a = \sum_{P \in \mathcal{P}(S,T): a \in P} U(P).$$

It may be useful to notice that if we replace each arc $a$ with $U_a$-many parallel arcs then $n_a$ exactly counts the number of unique paths from $S$ to $T$ that use $a$ in the resulting (multi) digraph. A simple dynamic-programming type algorithms that does a "sweep" from $S$ to $T$ and $T$ to $S$ shows that one can efficiently compute the path counts.

**Lemma 6.1.** *Let $D$ be a capacitated $h$-layer $S$-$T$ DAG. Then one can compute $n_v^+$ and $n_v^-$ for every vertex $v$ and $n_a$ for every arc $a$ in:*

1. *Parallel time $O(h)$ with $m$ processors;*
2. *CONGEST time $\tilde{O}(h^2)$.*

*Proof.* To compute $n_a$ it suffices to compute $n_v^+$ and $n_v^-$. We proceed to describe how to compute $n_v^-$; computing $n_v^+$ is symmetric.

First, notice that $n_v^-$ can be described by the recurrence

$$n_v^- := \begin{cases} 1 & \text{if } v \in S \\ \sum_{(u,v) \in \delta^-(v)} U_{uv} \cdot n_u^- & \text{otherwise} \end{cases}$$

We repeat the following for iteration $i = 2, \ldots, h+1$. Let $V_i$ be all vertices in the $i$th layer of our graph. In iteration $i$ we will compute $n_v^-$ for every $v \in V_i$ by applying the above recurrence.

Running one of the above iterations in parallel is trivial to do in $O(1)$ parallel time with $m$ processors, leading to the above parallel runtime. Running one iteration of this algorithm in CONGEST requires that every vertex in $v \in V_j$ for $j < i$ broadcast its $n_v^-$. Since $n_v^- \leq (n \cdot U_{\max})^h$ this can be done in $h\left(1 + \frac{\log U_{\max}}{\log n}\right)$ rounds of CONGEST, leading to the stated CONGEST runtime. □

# 7 Randomized Blocking Integral Flows in $h$-Layer DAGs

We now describe how to compute blocking integral flows in $h$-layer $S$-$T$ DAGs with high probability by using the path counts of the previous section. This is the general capacities version of the problem described in Section 4.3. More or less, the algorithm we use is one of Chang and Saranurak [12] adapted to the general capacities case; the algorithm of Chang and Saranurak [12] is itself an adaptation of an algorithm of Lotker et al. [30]. We mostly include these results for the sake of completeness.

Our randomized algorithm will repeatedly sample an integral flow proportional to the path counts of Section 6, add this to our existing flow, reduce capacities and then repeat. We will argue that we need only iterate this process a small number of times until we get a blocking integral flow by appealing to the fact that "high degree" paths have their capacities reduced with decent probability.

One can see this as essentially running the randomized MIS algorithm of Luby [31] but with two caveats: (1) the underlying graph in which we compute an MIS has a node for every path between $S$ and $T$ and so has up to $O(n^h)$-many nodes; as such we cannot explicitly construct this graph but rather can only implicitly run Luby's algorithm on it; (2) Luby's analysis assumes nodes attempt to enter the MIS independently but our sampling will have some dependencies between nodes (i.e. paths) entering the MIS which must be addressed in our analysis.

More formally, suppose we are given a capacitated $S$-$T$ DAG $D$. For a given path $P \in \mathcal{P}(S,T)$ we let $\Delta_P$ be $\sum_{P'} \prod_{a \in P' \setminus P} U_a$ be the "degree" of path $P$ where the sum over $P'$ ranges over all $P'$ that share at least one arc with $P$ and are in $\mathcal{P}(S,T)$. We let $\Delta = \max_{P \in \mathcal{P}(S,T)} \Delta_P$ be the maximum degree. Similarly, we let $\mathcal{P}_{\approx \max} := \{P : \Delta_P \geq \frac{\Delta}{2}\}$ be all paths with near-maximum degree. The following summarizes the flow we repeatedly compute; in this lemma the constant $\frac{2046}{2047}$ is arbitrary and could be optimized to be much smaller.

**Lemma 7.1.** *Given a $h$-layer $S$-$T$ DAG $D$ with capacities $U$ and $\tilde{\Delta}$ satisfying $\frac{\Delta}{2} \leq \tilde{\Delta} \leq \Delta$, one can sample an integral $S$-$T$ flow $f$ where for each $P \in \mathcal{P}_{\approx \max}$ we have $\prod_{a \in P}(U_a - f_a) \leq \frac{2047}{2048} \cdot U(P)$ with probability at least $\Omega(1)$. This can be done in:*

1. *Parallel time $O(h)$ with $m$;*
2. *CONGEST time $\tilde{O}\left(h^2\right)$ with high probability.*

*Proof.* The basic idea is to have each path $P$ sample about $U(P)/\tilde{\Delta}$ copies of itself.

More formally, we do the following. Consider the (multi) digraph $D'$ that is created by starting with $D$ and replacing each arc $a$ with $U_a$ copies. For a given path $P$ in $D'$ from $S$ to $T$, we let $\Delta'_P$ be the number of distinct $S$ to $T$ paths in $D'$ which share an arc with $P$. Likewise, we let

$\Delta' = \max_P \Delta'_P$ where this max is taken over all $S$ to $T$ paths in $D'$. We let $\mathcal{P}'_{\approx\max}$ be all paths $P$ for which $\Delta'_P \geq \Delta'/2$. By how we defined the degree of paths in $D$, if a given path $P$ is in $\mathcal{P}'_{\approx\max}$ then so too is its corresponding path in $D$ in $\mathcal{P}_{\approx\max}$. Lastly, we let $N(P)$ be all paths from $S$ to $T$ in $D'$ which share an arc with $P$ other than $P$ itself and let $N^+(P) := N(P) \cup \{P\}$.

In what follows we show how to sample a collection of arc-disjoint paths $\mathcal{P}_2$ in $D'$ where each $P \in \mathcal{P}'_{\approx\max}$ is such that with probability at least $\frac{1}{1024}$ the set $\mathcal{P}_2 \cap N^+(P)$ is non-empty. Before doing so, we observe that this suffices to show our claim. In particular, we can construct a flow $f$ by setting its value on arc $a$ to be $|\{P \in \mathcal{P}_2 : a \in P\}|$. Observe that by the arc-disjointness of $\mathcal{P}_2$ and how we constructed $D'$, $f$ is indeed a feasible $S$-$T$ flow. Moreover, we claim that for a given $\tilde{P} \in \mathcal{P}_{\approx\max}$ in $D$ we have $\prod_{a\in\tilde{P}}(U_a - f_a) \leq \frac{1}{2}U(\tilde{P})$ with probability $\Omega(1)$. In particular, let $X_P$ be the indicator of whether a given path $P$ in $D'$ from $S$ to $T$ is such that $N^+(P) \cap \mathcal{P}_2 = \emptyset$ so that $\mathbb{E}[X_P] \leq \frac{1023}{1024}$. Also, let $\tilde{\mathcal{P}}$ be all the paths in $D'$ that visit the same vertices as $\tilde{P}$ in $D$. Then we have

$$\prod_{a\in\tilde{P}}(U_a - f_a) = \sum_{P\in\tilde{\mathcal{P}}} X_P.$$

But, looking at the expectation of this, we have

$$\begin{aligned}\mathbb{E}\left[\sum_{P\in\tilde{\mathcal{P}}} X_P\right] &\leq \sum_{P\in\tilde{\mathcal{P}}} \frac{1023}{1024} \\ &= \frac{1023}{1024}\cdot U(\tilde{P})\end{aligned}$$

Thus, by Markov's inequality we have that $\sum_{P\in\tilde{\mathcal{P}}} X_P \geq \frac{2047}{2046}\cdot\mathbb{E}\left[\sum_{P\in\tilde{\mathcal{P}}} X_P\right]$ with probability at most $\frac{2046}{2047}$ and so with probability $\Omega(1)$ we get that $\sum_{P\in\tilde{\mathcal{P}}} X_P \leq \frac{2047}{2046}\cdot\mathbb{E}\left[\sum_{P\in\tilde{\mathcal{P}}} X_P\right] \leq \frac{2047}{2048}\cdot U(\tilde{P})$.

Thus, it remains to show how to sample our collection of arc-disjoint paths $\mathcal{P}_2$ in $D'$ where each $P \in \mathcal{P}'_{\approx\max}$ is such that with probability at least $\frac{1}{1024}$ the set $\mathcal{P}_2 \cap N^+(P)$ is non-empty. We will sample $\mathcal{P}_2$ as follows. Imagine that $s$ initially receives $B\left(n_s^+, \frac{1}{64\tilde{\Delta}}\right)$-many balls where $B(n,p)$ is a binomial with $n$ trials each with probability of success $p$. We let $n_a$ and $n_v^+$ be as defined in Section 6 for $D'$ where $U_{a'} = 1$ for every arc $a'$ in $D'$.

When a vertex $v$ receives a ball, it tosses it to vertex $u \in N^+(v)$ with probability $n_u^+/n_v^+$. As $n_v^+ = \sum_{w\in N^+(v)} n_w^+$ this induces a valid probability distribution. Let $\mathcal{P}_1$ be the (multi) set of all paths traced out by balls. We will let $\mathcal{P}_2$ be all paths in $\mathcal{P}_1$ which are arc-disjoint (in $D'$) from all other paths in $\mathcal{P}_1$.

We first consider this process from the perspective of a single path $P$ from $S$ to $T$ in $D'$. Specifically, notice that the probability that a ball traces out a path $P = (s = v_1, v_2, \ldots, v_{h+1} = t)$ where $s \in S$ and $t \in T$ is uniform over paths. In particular, the probability that a given ball traces out path $P$ in $D'$ from $s$ to $t$ nicely telescopes as

$$\begin{aligned}\frac{n_{v_2}^+}{n_{v_1}^+}\cdot\frac{n_{v_3}^+}{n_{v_2}^+}\cdot\ldots\cdot\frac{n_{v_{h+1}}^+}{n_{v_h}^+} &= \frac{n_{v_{h+1}}^+}{n_{v_1}^+} \\ &= \frac{1}{n_s^+}.\end{aligned}$$

Thus, each ball that starts at $s$ traces out a uniformly random path incident to $s$ in $\mathcal{P}(S,T)$. Applying the parameters of our binomial distribution, it follows that the expected number of times

a given path $P$ is included in $\mathcal{P}_1$ is $\frac{1}{64\cdot\tilde{\Delta}}$. Markov's inequality then shows that a given path has some copy in $\mathcal{P}_1$ with probability at most $\frac{1}{64\cdot\tilde{\Delta}} \leq \frac{1}{32\cdot\Delta}$. On the other hand, $P$ has exactly one copy included in $\mathcal{P}_1$ with probability $\frac{1}{64\tilde{\Delta}} n_s^+ \cdot \frac{1}{n_s^+}\left(1-\frac{1}{n_s^+}\right)^{n^+-1} \geq \frac{1}{128\tilde{\Delta}}$. Thus, $P$ has at least one copy in $\mathcal{P}_1$ with probability at least $\frac{1}{128\tilde{\Delta}} \geq \frac{1}{128\Delta}$.

We proceed to bound two simple probabilities regarding how paths are sampled. In particular, fix a path $P \in \mathcal{P}'_{\approx\max}$ in $D'$ from $S$ to $T$. Next, fix a $P' \in N^+(P)$. Then, let $\mathcal{E}_1(P')$ be the event that some copy of $P'$ is in $\mathcal{P}_1$ and no other path in $N^+(P)$ has a copy in $\mathcal{P}_1$. Likewise, let $\mathcal{E}_2(P')$ be the event that no path in $N(P')$ is in $\mathcal{P}_1$. Notice that if $\mathcal{E}_1(P')$ and $\mathcal{E}_2(P')$ hold then we have $P' \in \mathcal{P}_2$.

- **Bounding** $\Pr(\mathcal{E}_1(P'))$. We will argue that $\Pr(\mathcal{E}_1(P')) \geq \frac{1}{256\Delta}$.

  Notice that since $N^+(P) \setminus \{P'\}$ consists of at most $\Delta$-many paths, the expected number of copies of paths in $N^+(P) \setminus \{P'\}$ in $\mathcal{P}_1$ is at most $\frac{1}{32}$. It follows by a Markov bound that with probability at least $\frac{1}{2}$ we have $N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset$.

  Next, imagine that we condition on the event $N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset$. Conditioning on this event can only increase the probability that a ball traces out $P'$. Since some copy of $P'$ is included in $\mathcal{P}_1$ with probability at least $\frac{1}{128\Delta}$ when we don't condition on this event, we conclude that

$$\begin{aligned}\Pr(\mathcal{E}_1(P')) &= \Pr(N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset) \cdot \Pr(P' \in \mathcal{P}_1 \mid N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset) \\ &\geq \Pr(N^+(P) \setminus \{P'\} \cap \mathcal{P}_1 = \emptyset) \cdot \Pr(P' \in \mathcal{P}_1) \\ &\geq \frac{1}{256\Delta}.\end{aligned}$$

- **Bounding** $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P'))$. We argue that $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P')) \geq \frac{1}{2}$.

  Notice that $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P'))$ is minimized when $N^+(P)$ is of size exactly $\Delta + 1$. However, in this case we have $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P')) \geq \Pr(\mathcal{E}_2(P'))$. Thus, we conclude by a union bound that in general $\Pr(\mathcal{E}_2(P') \mid \mathcal{E}_1(P')) \geq \Pr(\mathcal{E}_2(P')) \geq 1 - \Delta \cdot \frac{1}{32\Delta} \geq \frac{1}{2}$.

Putting these facts together and applying the fact that $P \in \mathcal{P}'_{\approx\max}$, we have that there is path in $N^+(P)$ included in $\mathcal{P}_2$ with probability at least

$$\begin{aligned}\sum_{P' \in N^+(P)} \Pr(\mathcal{E}_1(P')) \cdot \Pr(\mathcal{E}_2(P') \mid \mathcal{E}_2(P')) &\geq \sum_{P' \in N^+(P)} \frac{1}{512\Delta} \\ &\geq \frac{1}{1024}.\end{aligned}$$

as required.

It remains to argue that we can accomplish the above sampling of $\mathcal{P}_1$ and the construction of our flow $f$ in the stated times. Constructing $f$ from $\mathcal{P}_1$ is trivial to do in parallel and CONGEST so we focus on sampling $\mathcal{P}_1$. By Lemma 6.1 we can compute $n_v^+$ in the stated times. Passing balls to construct $\mathcal{P}_1$ and then $\mathcal{P}_2$ and constructing the above flow is trivial to do in the stated parallel time. For the CONGEST algorithm, we note that expected number of balls to cross any one arc in $D'$ when constructing $\mathcal{P}_1$ is at most 1 and so a Chernoff and union bound shows that

with high probability we never need to transmit more than $O(\log n)$ balls across an arc in $D'$ when constructing $\mathcal{P}_1$, with high probability. It follows that we never need to transmit more than $\tilde{O}(U_{\max})$ balls across any one arc in $D$. Since it suffices to just transmit the number of balls, this can be done in $\tilde{O}(\log U_{\max}) = \tilde{O}(1)$ rounds with high probability. Thus we can pass all balls from one layer to the next in $\tilde{O}(1)$ rounds of CONGEST with high probability. Lastly, constructing $\mathcal{P}_2$ from $\mathcal{P}_1$ is trivial to do in $O(h)$ rounds of CONGEST. □

**Lemma 7.2.** *There is an algorithm which, given an $h$-layer $S$-$T$ DAG $D$ with capacities $U$, computes an integral $S$-$T$ flow that is blocking in:*

1. *Parallel time $\tilde{O}(h^3)$ with $m$ processors with high probability;*
2. *CONGEST time $\tilde{O}(h^4)$ with high probability.*

*Proof.* Our algorithm simply repeatedly calls Lemma 7.1. In particular we initialize our output flow $\hat{f}$ to be 0 on all arcs and our working capacities on $D$ to be $\hat{U} = U$. Then for each $\tilde{\Delta} = (n \cdot U_{\max})^h, (n \cdot U_{\max})^h/2, (n \cdot U_{\max})^h/4, \ldots$ we repeat the following $\Theta(h \cdot \log n \cdot \log U_{\max})$ times. Let $f$ be the flow computed according to Lemma 7.1. Update $\hat{U}_a = \hat{U}_a - f_a$ for every $a$ and update $\hat{f} = \hat{f} + f$. Clearly $\hat{f}$ is an integral $S$-$T$ flow.

We need only verify that $\hat{f}$ is blocking. Since initially $\Delta \leq (n \cdot U_{\max})^h$, to do so it suffices to argue that when we fix a value of $\tilde{\Delta}$ for which $\frac{\Delta}{2} \leq \tilde{\Delta} \leq \Delta$, then over the course of the $\Theta(h \cdot \log n \cdot \log U_{\max})$ iterations where we use this value of $\tilde{\Delta}$ we have that $\Delta$ decreases by at least a factor of 2 with high probability.

Consider $\Theta(h \cdot \log n \cdot \log U_{\max})$ contiguous iterations of the above with a $\tilde{\Delta}$ that satisfies $\frac{\Delta}{2} \leq \tilde{\Delta} \leq \Delta$ at the beginning of these iterations. Let $\mathcal{P}_0$ be $\mathcal{P}_{\approx\max}$ at the beginning of these iterations. To show that $\Delta$ decreases by at least a factor of 2 over the course of these $\Theta(h \cdot \log n \cdot \log U_{\max})$ iterations it suffices to show that no path in $\mathcal{P}_0$ is in $\mathcal{P}_{\approx\max}$ for all of these iterations. Suppose for the sake of contradiction that some path $P \in \mathcal{P}_0$ is in $\mathcal{P}_{\approx\max}$ for all of these iterations. Then, applying the guarantees of Lemma 7.1, we get that with high probability $U(P)$ decreases by a $\frac{2047}{2048}$ factor at least $\Theta(h \cdot \log U_{\max}))$ times. However, since $U(P) \leq O((U_{\max})^h)$, we get that after these iterations we would have reduced $U(P)$ to 0 with high probability by a union bound, i.e. $\Delta$ must have reduced by at least a factor of 2.

The running time of our algorithm is immediate from the fact that we simply invoke Lemma 7.1 $\tilde{O}(h^2)$ times. □

# 8 Deterministic and Distributed Near Eulerian Partitions

In the previous section we showed how to efficiently compute blocking integral flows in $h$-layer DAGs with high probability. In this section, we introduce the key idea we make use of in doing so deterministically, a near Eulerian partition.

Informally, a near Eulerian partition will discard a small number of edges and then partition the remaining edges into cycles and paths. Because these cycles and paths will have small diameter in our construction, we will be able to efficiently orient them in CONGEST. In Section 9 we will see how to use these oriented cycles and paths to efficiently round flows in a distributed fashion in order to computer a blocking integral flow in $h$-layer DAGs.

We now formalize the idea of a $(1 - \varepsilon)$-near Eulerian partition.

**Definition 8.1** ($(1-\varepsilon)$-Near Eulerian Partition)**.** *Let $G = (V, E)$ be an undirected graph and $\epsilon \geq 0$. A $(1-\varepsilon)$-near Eulerian partition $\mathcal{H}$ is a collection of edge-disjoint cycles and paths in $G$, where*

1. *$(1-\varepsilon)$-**Near Covering:** The number of edges in $E[\mathcal{H}]$ is at least $(1-\varepsilon) \cdot |E|$;*
2. ***Eulerian Partition:** Each vertex is the endpoint of at most one path in $\mathcal{H}$.*

The following is the main result of this section and summarizes our algorithms for construction $(1-\varepsilon)$-near Eulerian partitions. In what follows we say that a cycle is oriented if every edge is directed so that every vertex in the cycle has in and out degree 1; a path $P$ is oriented if it has some designated source and sink $s_P$ and $t_P$. We say that a collection of paths and cycles $\mathcal{H}$ is oriented if each element of $\mathcal{H}$ is oriented. In CONGEST we will imagine that a cycle is oriented if each vertex knows the orientation of its incident arcs and a path is oriented if every vertex knows which of its neighbors are closer to $s_P$.

**Lemma 8.2.** *One can deterministically compute an oriented $(1-\varepsilon)$-near Eulerian partitions in:*

1. *Parallel time $\tilde{O}(1)$ with $m$ processors and $\epsilon = 0$;*
2. *CONGEST time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot (\rho_{CC})^{10})$ for any $\varepsilon > 0$.*

Again, see Section 5.4 for a definition of $\rho_{CC}$.

## 8.1 High-Girth Cycle Decompositions

In order to compute our near Eulerian partitions we will make use of a slight variant of cycle covers which we call high-girth cycle decompositions (as introduced in Section 5.4). The ideas underpinning these decompositions seem to be known in the literature but there does not seem to be a readily citable version of quite what we need; hence we give details below.

To begin, in our near-Eulerian partitions we would like for our cycles to be edge-disjoint so that each cycle can be rounded independently. Thus, we give a subroutine for taking a collection of cycles and computing a large edge-disjoint subset of this collection. This result comes easily from applying a deterministic approximation algorithm for maximal independent set (MIS). Congestion and dilation in what follows are defined in Section 5.4.

**Lemma 8.3.** *There is a deterministic CONGEST algorithm that, given a graph $G = (V, E)$ and a collection of (not necessarily edge-disjoint) cycles $\mathcal{C}$ with congestion $c$ and diameter $d$, outputs a set of edge disjoint cycles $\mathcal{C}' \subseteq \mathcal{C}$ which satisfies $|E[\mathcal{C}']| \geq \frac{1}{d^2c^2} \cdot |E[\mathcal{C}]|$ in time $\tilde{O}(c^3d^3)$.*

*Proof.* Our algorithm simply computes an approximately-maximum independent set in the conflict graph which has a node for each cycle. In particular, we construct conflict graph $G' = (\mathcal{C}, E')$ as follows. Our vertex set is $\mathcal{C}$. We include edge $\{C, C'\}$ in $E'$ if $C \in \mathcal{C}$ and $C' \in \mathcal{C}$ overlap on an edge; that is, if $E[C] \cap E[C'] \neq \emptyset$.

Observe that since each cycle in $\mathcal{C}$ has at most $d$-many edges and since each edge is in at most $c$-many cycles, we have that the maximum degree of $G'$ is $cd$. Next, we let the "node-weight" of cycle $C \in \mathcal{C}$ be $|C|$. We apply Theorem 5.2 with these node-weights to compute a $\frac{1}{cd}$-approximate maximum independent set $\mathcal{C}'$. We return $\mathcal{C}'$ as our solution.

First, observe that since $\mathcal{C}'$ is an independent set in $G'$, we have that the cycles of $\mathcal{C}'$ are indeed edge-disjoint.

Next, we claim that $|E[\mathcal{C}']| \geq \frac{1}{d^2c^2} \cdot |E[\mathcal{C}]|$. Since Theorem 5.2 guarantees that $\mathcal{C}'$ is a $\frac{1}{dc}$-approximate solution, to show this, it suffices to argue that $|E[\mathcal{C}^*]| \geq \frac{1}{dc} \cdot |E[\mathcal{C}]|$ where $\mathcal{C}^* \subseteq \mathcal{C}$ is the set of edge-disjoint cycles of maximum edge cardinality, i.e. the maximum node-weight independent set in $G'$. However, notice that since the total node weight in $G'$ is $\sum_{C \in \mathcal{C}} |E[C]|$ and the max degree in $G'$ is at most $cd$, we have that the maximum node-weight independent set in $G'$ must have node-weight at least $\frac{1}{cd} \sum_{C \in \mathcal{C}} |E[C]| \geq \frac{1}{cd} |E[\mathcal{C}]|$. Thus, we conclude that $|E[\mathcal{C}']| \geq \frac{1}{d^2c^2} \cdot |E[\mathcal{C}]|$.

Next, we argue that we can implement the above in the stated running times. Computing our $\frac{1}{cd}$-approximate maximum independent set on $G'$ takes deterministic CONGEST time $\tilde{O}(cd)$ on $G'$ by Theorem 5.2. Furthermore, we claim that we can simulate a CONGEST algorithm on $G'$ in $G$ with only an overhead of $O(c^2d^2)$. In particular, since the maximum degree on $G'$ is $cd$, in each CONGEST round on $G'$ each node (i.e. cycle in $G$) receives at most $cd$-many messages. Fix a single round of CONGEST on $G'$. We will maintain the invariant that if $v \in V$ is a node in a cycle $C \in \mathcal{C}$, then in our simulation $v$ receives all the same messages as $C$ in our CONGEST algorithm on $G'$. We do so by broadcasting all messages that $C$ receives in this one round on $G'$ to all nodes in $C$. As a cycle in $G'$ receives at most $cd$ messages in one round of CONGEST on $G'$ and each edge is in at most $c$-many cycles, it follows that in such a broadcast the number of messages that need to cross any one edge is at most $c^2d$. Since the diameter of each cycle is at most $d$, we conclude that this entire broadcast can be done deterministically in time $O(c^2d^2)$, giving us our simulation.

Combining this $O(c^2d^2)$-overhead simulation with the $\tilde{O}(cd)$ running time of our approximate maximum independent set algorithm on $G'$ gives an overall running time of $O(c^3d^3)$. □

Recall that the girth of a graph is the minimum length of a cycle in it. The following formalizes the notion of high-girth cycle decompositions that we will need.

**Definition 8.4** (High-Girth Cycle Decomposition)**.** *Given a graph $G = (V, E)$ and $\varepsilon > 0$ where $E_0$ are all non-bridge edges of $G$, a high-girth cycle decomposition with diameter $d$ and deletion girth $k$ is a collection of edge-disjoint (simple) cycles $\mathcal{C}$ such that:*

1. ***High Deletion Girth:*** *The graph $(V, E \setminus E[\mathcal{C}])$ has girth at least $k$.*
2. ***Low Diameter:*** $\max_{C \in \mathcal{C}} |C| \leq d$*;*

The following theorem gives the construction of high-girth cycle decompositions that we will use.

**Theorem 8.5.** *There is a deterministic CONGEST algorithm that, given a graph $G = (V, E)$ and desired girth $k \geq 0$, computes a high-girth cycle decomposition with diameter $\tilde{O}(k \cdot \rho_{CC})$ and girth $k$ in time $\tilde{O}(k^5 \cdot (\rho_{CC})^{10})$.*

*Proof.* The basic idea is: take a sparse neighborhood cover; compute cycle covers on each part of our neighborhood cover; combine all of these into a single cycle cover; decongest this cycle cover into a collection of edge-disjoint cycles; delete these cycles; repeat.

More formally, our algorithm is as follows, We initialize our collection of cycles $\mathcal{C}$ to $\emptyset$.

Next, we repeat the following $\tilde{\Theta}\left(k^2 \cdot (\rho_{CC})^4\right)$ times. Apply Lemma 5.6 to compute an $\tilde{O}(1)$-sparse $k$-neighborhood cover of $G$ with diameter $\tilde{O}(k)$ and overlap $\tilde{O}(1)$. Let $\mathcal{V}_1, \mathcal{V}_2, \ldots$ be the partitions of this neighborhood cover. By definition of a neighborhood cover, for each $\mathcal{V}_i$ and each $V_i^{(j)} \in \mathcal{V}_i$, we have that $V_i^{(j)}$ comes with a tree $T_i^{(j)}$ where each node in the tree is in $\tilde{O}(1)$ other $V_i^{(j)}$. We let $H_i^{(j)} := G[V_i^{(j)}] \cup T_i^{(j)}$ be the union of this tree and the graph induced on $V_i^{(j)}$. By the

guarantees of our neighborhood cover we have that the diameter of $H_i^{(j)}$ is at most $\tilde{O}(k)$. We then compute a cycle cover $\mathcal{C}_i^{(j)}$ of each $H_i^{(j)}$ with diameter $\tilde{O}(k \cdot \rho_{CC})$ and congestion $\rho_{CC}$ (we may do so by definition of $\rho_{CC}$). We let $\mathcal{C}_0 = \bigcup_{i,j} \mathcal{C}_i^{(j)}$ be the union of all of these cycle covers. Next, we apply Lemma 8.3 to compute a large edge-disjoint subset $\mathcal{C}_0' \subseteq \mathcal{C}_0$ of $\mathcal{C}_0$. We add $\mathcal{C}_0'$ to $\mathcal{C}$ and delete from $G$ any edge that occurs in a cycle in $\mathcal{C}_0'$.

We first argue that the solution we return is indeed a high-girth cycle decomposition. Our solution consists of edge-disjoint cycles by construction. Next, consider one iteration of our algorithm. Observe that since each $\mathcal{C}_i^{(j)}$ has congestion at most $\rho_{CC}$, it follows by the $\tilde{O}(1)$ overlap and $\tilde{O}(1)$ sparsity of our neighborhood cover that $\mathcal{C}_0$ has congestion $\tilde{O}(\rho_{CC})$. Likewise, since each $H_i^{(j)}$ has diameter $\tilde{O}(k)$, it follows that each $\mathcal{C}_i^{(j)}$ has diameter at most $\tilde{O}(k \cdot \rho_{CC})$ and so $\mathcal{C}_0$ has diameter at most $\tilde{O}(k \cdot \rho_{CC})$. Thus, $\mathcal{C}_0$ has congestion at most $\tilde{O}(\rho_{CC})$ and diameter at most $\tilde{O}(k \cdot \rho_{CC})$. Since $\mathcal{C}_0' \subseteq \mathcal{C}_0$, it immediately follows that the solution we return has diameter at most $\tilde{O}(k \cdot \rho_{CC})$.

It remains to show that the deletion of our solution induces a graph with high girth. Towards this, observe that applying the congestion and diameter of $\mathcal{C}_0$ and the guarantees of Lemma 8.3, it follows that

$$|E[\mathcal{C}_0']| \geq \tilde{\Omega}\left(\frac{1}{k^2(\rho_{CC})^4}\right) \cdot |E[\mathcal{C}_0]|. \tag{1}$$

On the other hand, let $E_0$ be all edges in cycle of diameter at most $k$ at the beginning of this iteration. Consider an $e \in E_0$. Since $\mathcal{V}_1, \mathcal{V}_2, \ldots$ is a $k$-neighborhood cover we know that there is some $\mathcal{C}_i^{(j)}$ which contains a cycle which contains $e$. Thus, we have

$$|E[\mathcal{C}_0]| \geq |E_0|. \tag{2}$$

Combining Equation (1) and Equation (2), we conclude that

$$|E[\mathcal{C}_0']| \geq \tilde{\Omega}\left(\frac{1}{k^2(\rho_{CC})^4}\right) \cdot |E_0|.$$

However, since in this iteration we delete every edge in $E[\mathcal{C}_0']$, it follows that we reduce the number of edges that are in a cycle of diameter at most $k$ by at least a $1 - \tilde{\Omega}\left(\frac{1}{k^2(\rho_{CC})^4}\right)$ multiplicative factor. Since initially the number of such edges is at most $|E|$, it follows that after $\tilde{O}(k^2 \cdot (\rho_{CC})^4)$-many iterations we have reduced the number of edges in a cycle of diameter at most $k$ to 0; in other words, our graph has girth at most $k$. This shows the high girth of our solution, namely that $(V, E \setminus E[\mathcal{C}])$ has girth at least $k$ after the last iteration of our algorithm.

Next, we argue that we achieve the stated running times. Fix an iteration.

- By the guarantees of Lemma 5.6, the sparse neighborhood cover that we compute takes time $\tilde{O}(k)$.
- We claim that by definition of $\rho_{CC}$, the $\tilde{O}(k)$ diameter of each part in our sparse neighborhood cover and the $\tilde{O}(1)$ overlap of our sparse neighborhood cover, we can compute every $\mathcal{C}_i^{(j)}$ in time $\tilde{O}(k \cdot \rho_{CC})$. Specifically, for a fixed $i$ we run the cycle cover algorithm simultaneously in meta-rounds, each consisting of $\tilde{\Theta}(1)$ rounds. In each meta-round a node can send the messages that it must send for the cycle cover algorithm of each of the $H_i^{(j)}$ to which it

is incident by our overlap guarantees. Since the total number of $i$ is $\tilde{O}(1)$ by our sparsity guarantee, we conclude that we can compute all $\mathcal{C}_i^{(j)}$ in a single iteration in at most $\tilde{O}(k \cdot \rho_{CC})$ time.

- Lastly, by the guarantees of Lemma 8.3 and the fact that $\mathcal{C}_0$ has congestion at most $\tilde{O}(\rho_{CC})$ and diameter at most $\tilde{O}(k \cdot \rho_{CC})$, we can compute $\mathcal{C}_0'$ in time $\tilde{O}(k^3 \cdot (\rho_{CC})^6)$.

Combining the above running times with the fact that we have $\tilde{\Theta}\left(k^2 \cdot (\rho_{CC})^4\right)$-many iterations gives us a running time of $\tilde{O}(k^5 \cdot (\rho_{CC})^{10})$. □

## 8.2 Efficient Algorithms for Computing Near Eulerian Partitions

We conclude by proving the main section of this theorem, namely the following which shows how to efficiently compute near Eulerian partitions in deterministic CONGEST by making use of our high-girth cycle decomposition construction and DLDDs.

**Lemma 8.2.** *One can deterministically compute an oriented $(1-\varepsilon)$-near Eulerian partitions in:*

1. *Parallel time $\tilde{O}(1)$ with $m$ processors and $\epsilon = 0$;*
2. *CONGEST time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot (\rho_{CC})^{10})$ for any $\varepsilon > 0$.*

*Proof.* The parallel result is well-known since a 1-near Eulerian partition is just a so-called Eulerian partition; see e.g. Karp and Ramachandran [25].

The rough idea of our CONGEST algorithm is as follows. First we compute a high-girth cycle decomposition (Definition 8.4), orient these cycles and remove all edges covered by this decomposition. The remaining graph has high girth by assumption. Next we compute a DLDD (Definition 5.3) on the remaining graph; by the high girth of our graph each part of our DLDD is a low diameter tree. Lastly, we decompose each such tree into a collection of paths.

More formally, our CONGEST algorithm to return cycles $\mathcal{C}$ and paths $\mathcal{P}$ is as follows. Apply Theorem 8.5 to compute a high-girth cycle decomposition $\mathcal{C}$ with deletion girth $\tilde{\Theta}(\frac{1}{\varepsilon})$ and diameter $\tilde{O}(\frac{1}{\varepsilon} \cdot \rho_{CC})$. Orient each cycle in $\mathcal{C}$ and delete from $G$ any edge in a cycle in $\mathcal{C}$. Next, apply Theorem 5.4 to compute a DLDD with diameter $\tilde{\Theta}(\frac{1}{\varepsilon})$ and cut fraction $\varepsilon$. Delete all edges cut by this DLDD. Since $\mathcal{C}$ has deletion girth $\tilde{\Theta}(\frac{1}{\varepsilon})$, by appropriately setting our hidden constant and poly-logs, it follows that no connected component in the remaining graph contains a cycle; in other words, each connected component is a tree with diameter $\tilde{\Theta}(\frac{1}{\varepsilon})$.

We decompose each tree $T$ in the remaining forest as follows. Fix an arbitrary root $r$ of $T$. We imagine that each vertex of odd degree in $T$ starts with a ball. Each vertex waits until it has received a ball from each of its children. Once a vertex has received all such balls, it pairs off the balls of its children arbitrarily, deletes these balls and adds to $\mathcal{P}$ the concatenation of the two paths traced by these balls in the tree. It then passes its up to one remaining ball to its parent. Lastly, we orient each path in $\mathcal{P}$ arbitrarily.

We begin by arguing that the above results in a $(1-\varepsilon)$-near Eulerian partition. Our paths and cycles are edge-disjoint by construction. The only edges that are not included in some element of $\mathcal{C} \sqcup \mathcal{P}$ are those that are cut by our DLDD; by our choice of parameters this is at most an $\varepsilon$ fraction of all edges in $E$. To see the Eulerian partition property, observe that every vertex of odd degree in $G[\mathcal{C} \sqcup \mathcal{P}]$ is an endpoint of exactly one path in $\mathcal{P}$ since each odd degree vertex starts with exactly

one ball. Likewise, a vertex of even degree will never be the endpoint of a path since no such vertex starts with a ball.

It remains to argue that the above algorithm achieves the stated CONGEST running time.

- Computing $\mathcal{C}$ takes time at most $\tilde{O}(\frac{1}{\varepsilon^5} \cdot (\rho_{CC})^{10})$ by Theorem 8.5. Furthermore, by Theorem 8.5, each cycle in $\mathcal{C}$ has diameter $\tilde{O}(\frac{1}{\varepsilon} \cdot \rho_{CC})$ and so can be oriented in time $\tilde{O}(\frac{1}{\varepsilon} \cdot \rho_{CC})$.
- Computing our DLDD takes time $\tilde{O}(\frac{1}{\varepsilon})$ by Theorem 5.4.
- Since our DLDD has diameter $\tilde{O}(\frac{1}{\varepsilon})$, we have that the above ball-passing to comptue $\mathcal{P}$ can be implemented in time at most $\tilde{O}(\frac{1}{\varepsilon})$.

Thus, overall our CONGEST algorithm takes time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot (\rho_{CC})^{10})$. □

# 9 Deterministic Blocking Integral Flows in $h$-Layer DAGs

In Section 7 we showed how to efficiently compute blocking integral flows in $h$-layer DAGs with high probability. In this section, we show how to do so deterministically by making use of the near Eulerian partitions of Section 8. Specifically, we show the following.

**Lemma 9.1.** *There is a deterministic algorithm which, given a capacitated $h$-layer $S$-$T$ DAG $D$, computes an integral $S$-$T$ flow that is blocking in:*

1. *Parallel time $\tilde{O}(h^3)$ with $m$ processors;*
2. *CONGEST time $\tilde{O}(h^6 \cdot (\rho_{CC})^{10})$.*

The above parallel algorithm is more or less implied by the work of Cohen [15]. However, the key technical challenge we solve in this section is a distributed implementation of the above. Nonetheless, for the sake of completeness we will include the parallel result as well alongside our distributed implementation.

Our strategy for showing the above lemma has two key ingredients.

**Iterated Path Count Flow.** First, we construct the iterated path count flow. This corresponds to repeatedly taking the expected flow induced by the sampling of our randomized algorithm (as given by Lemma 7.1). As the flow we compute is the expected flow of the aforementioned sampling, this process is deterministic. The result of this is a $\tilde{\Omega}(\frac{1}{h})$-blocking but not necessarily integral flow. We argue that any such flow is also $\tilde{\Omega}(\frac{1}{h^2})$-approximate and so the iterated path count flow is nearly optimal but fractional.

**Flow Rounding.** Next, we provide a generic way of rounding a fractional flow to be in integral in an $h$-layer DAG while approximately preserving its value. Here, the main challenge is implementing such a rounding in CONGEST; the key idea we use is that of a $(1-\varepsilon)$-near Eulerian partition from Section 8 which discards a small number of edges and then partitions the remaining graph into cycles and paths.

These partitions enables us to implement a rounding in the style of Cohen [15]. In particular, we start with the least significant bit of our flow, compute a $(1-\varepsilon)$-near Eulerian partition of the

graph induced by all arcs which set this bit to 1 and then use this partition to round all these bits to 0. Working our way from least to most significant bit results in an integral flow. The last major hurdle to this strategy is showing that discarding a small number of edges does not damage our resulting integral flow too much; in particular discarding edges in the above way can increase the deficit of our flow. However, by always discarding an appropriately small number of edges we show that this deficit is small and so after deleting all flow that originates or ends at vertices not in $S$ or $T$, we are left with a flow of essentially the same value of the input fraction flow. The end result of this is a rounding procedure which rounds the input fractional flow to an integral flow while preserving the value of the flow up to a constant.

Our algorithm to compute blocking integral flows in $h$-layer DAGs deterministically combines the above two tools. Specifically, we repeatedly compute the iterated path count flow, round it to be integral and add the resulting flow to our output. As the iterated path count flow is $\tilde{\Omega}(\frac{1}{h^2})$-approximate, we can only repeat this about $h^2$ time (otherwise we would end up with a flow of value greater than that of the optimal flow).

### 9.1 Iterated Path Count Flows

In this section we define our iterated path count flows and prove that they are $\tilde{\Omega}(\frac{1}{h})$-approximate.

Specifically, the path counts of Section 6 naturally induce a flow. In particular, they induce what we will call the path count flow where the flow on arc $(u, v)$ is defined as:

$$f_a = U_a \cdot \frac{n_a}{\max_{a \in A} n_a}.$$

It is easy to see these path counts induce an $S$-$T$ flow.

**Lemma 9.2.** *For a given capacitated $S$-$T$ DAG the path count flow is an $S$-$T$ flow.*

*Proof.* The above flow does not violate capacities by construction. Moreover, it obeys flow conservation for all vertices other than those in $S$ and $T$ since it is a convex combination of paths between $S$ and $T$. More formally, for any vertex $v \notin S \cup T$ we have flow conservation by the calculation:

$$\begin{aligned}
\sum_{a=(u,v)\in\delta^-(v)} f_a &= \frac{U_a}{\max_{a\in A} n_a} \sum_{a=(u,v)\in\delta^-(v)} \sum_{P\in\mathcal{P}(S,T):a\in P} U(P) \\
&= \frac{U_a}{\max_{a\in A} n_a} \sum_{a=(u,v)\in\delta^+(v)} \sum_{P\in\mathcal{P}(S,T):a\in P} U(P) \\
&= \sum_{a=(u,v)\in\delta^+(v)} f_a
\end{aligned}$$

where the second line follows from the fact that every path from $S$ to $T$ which enters $v$ must also exit $v$. □

Path count flows were first introduced by Cohen [15]. Our notion of an iterated path count flow is closely related to Cohen [15]'s algorithm for computing blocking flows in parallel. In particular, in order to compute an integral blocking flow, Cohen [15] iteratively computes a path count flow, rounds it, decrements capacities and then iterates. For us it will be more convenient to do something

slightly different; namely, we will compute a path count flow, decrement capacities and iterate; once we have *a single* blocking fractional flow we will apply our rounding *once*. Nonetheless, we note that many of the ideas of this section appear implicitly in Cohen [15].

We proceed to define the iterated path count flow which is always guaranteed to be near-optimal. The iterated path count flow will be a sum of several path count flows. More formally, suppose we are given an $h$-layer capacitated $S$-$T$ DAG $D = (V, A)$ with capacities $U$. In such a DAG we have $n_a \leq (n \cdot U_{\max})^h$. We initialize $f_0$ to be the flow that assigns 0 to every arc and $U_0 = U$. We then let $D_i = (V, A)$ with capacities $U_i$ where $U_i = U_{i-1} - f_{i-1}$ and $f_{i-1}$ is the path count flow of $D_{i-1}$. Lastly, we define the iterated path count flow as a convex combination of these path count flows iterated $k = \Theta(h \cdot (\log n) \cdot \log(n \cdot U_{\max}))$ times. That is, the iterated path count flow is

$$\tilde{f} := \sum_{i=0}^{k} f_i.$$

We begin by observing that the iterated path count flow is reasonably blocking.

**Lemma 9.3.** *The iterated path count flow $\tilde{f}$ is a (not necessarily integral) blocking $S$-$T$ flow.*

*Proof.* Since each path count flow is an $S$-$T$ flow by Lemma 9.2, by how we reduce capacities it immediately follows that $\tilde{f}$ is an $S$-$T$ flow.

Thus, it remains to argue that $\tilde{f}$ is blocking. Towards this, consider computing the $i$th path count flow when the current path counts are $\{n_a\}_a$ and the flow over arc $a$ is $(f_i)_a = (U_i)_a \cdot \frac{n_a}{\max_a n_a}$. Letting $A_{\approx\max}$ be all arcs for which $n_a \geq \frac{1}{2} \max_a n_a$, we get that $(U_{i+1})_a \leq \frac{1}{2} \cdot (U_i)_a$ for all $a \in A_{\approx\max}$. It follows that after $\Theta(\log n)$ iterations we will reduce $\max_a n_a$ by at least a multiplicative factor of 2. Since initially $n_a \leq (n \cdot U_{\max})^h$, it follows that after $k = \Theta(h \cdot (\log n) \cdot \log(n \cdot U_{\max}))$ iterations we have reduced $n_a$ to 0 for every arc which is to say that for any path $P$ between $S$ and $T$ we have that there is some arc $a \in P$ it holds that $\sum_i (f_i)_a = U_a$. Since $\tilde{f}_a = \sum_i (f_i)_a$, we conclude that $\tilde{f}$ is blocking. □

Next, we observe that any blocking flow is near-optimal.

**Lemma 9.4.** *Any $\alpha$-blocking $S$-$T$ flow in an $h$-layer $S$-$T$ DAG is $\left(\frac{\alpha}{h}\right)$-approximate.*

*Proof.* Let $f$ be our $\alpha$-blocking flow and let $D$ be the input graph. Let $f^*$ be the optimal $S$-$T$ flow in the input DAG and let $\sum_P f_P$ be it's flow decomposition into path flows where each $P$ is a directed path from $S$ to $T$ and $(f_P)_a$ is 1 if $a \in P$ and 0 otherwise.

Since $f$ is blocking, for each such path there is some arc, $a_P$ where $f_{a_P} \geq \alpha \cdot U_{a_P} \geq \alpha \cdot f^*_{a_P}$. Let $A' = \{a_P : P \text{ in flow decomposition of } f^*\}$ be the union of all such blocked arcs. Thus, $\text{val}(f^*) \leq \sum_{a \in A'} f^*_a \leq \sum_{a \in A'} \frac{f_a}{\alpha}$. However, since $D$ is $h$-layered, by an averaging argument we have that there must be some $j$ such that $f(\delta^+(V_j) \cap A') \geq \frac{1}{h} \sum_{a \in A'} f_a$ where $V_j$ is the $j$th layer of our digraph. On the other hand, $\text{val}(f) \geq f(\delta^+(V_j)) \geq f(\delta^+(V_j) \cap A')$ and so we conclude that

$$\begin{aligned} \text{val}(f) &\geq f(\delta^+(V_j) \cap A') \\ &\geq \frac{1}{h} \cdot \sum_{a \in A'} f_a \\ &\geq \frac{\alpha}{h} \cdot \text{val}(f^*), \end{aligned}$$

showing that $f$ is $\left(\frac{\alpha}{h}\right)$-approximate as desired. □

We conclude that the iterated path count flow is near-optimal and efficiently computable; our CONGEST algorithm will make use of sparse neighborhood covers to deal with potentially large diameter graphs.

**Lemma 9.5.** *Let $D$ be a capacitated $h$-layer $S$-$T$ DAG with diameter at most $\tilde{O}(h)$. Then one can deterministically compute a (possibly non-integral) flow $\tilde{f}$:*

1. *In parallel that is $\Omega\left(\frac{1}{h}\right)$-approximate in time $\tilde{O}(h^2)$ with $m$ processors;*
2. *In CONGEST that is $\tilde{\Omega}\left(\frac{1}{h}\right)$-approximate in time $\tilde{O}\left(h^4\right)$.*

*Proof.* Combining Lemma 9.3 and Lemma 9.4 shows that the iterated path count flow is an $S$-$T$ flow that is $\Omega(\frac{1}{h})$-approximate.

For our parallel algorithm, we simply return the iterated path count flow. The iterated path count flow is simply a sum of $k = \Theta(h \cdot (\log n) \cdot \log(n \cdot U_{\max})$-many path count flows. Thus, it suffices to argue that we can compute path count flows in $O(h)$ parallel time with $m$ processors. By Lemma 6.1 we can compute $n_a$ for every $a$ in these times and so to then compute the corresponding path count flows we need only compute $\max_a n_a$ which is trivial to do in parallel in the stated time.

For our CONGEST algorithm we do something similar but must make use of sparse neighborhood covers, because we cannot outright compute $\max_a n_a$ as the diameter of $D$ might be very large. Specifically, we do the following. Apply Lemma 5.6 to compute an $s$-sparse $h$-neighborhood cover with diameter $\tilde{O}(h)$ and partition $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_s$ for $s = \tilde{O}(1)$. Then we iterate through each of these partitions for $i = 1, 2, \ldots, s$. For each part $V_i^{(j)} \in \mathcal{V}_i$, we let $\tilde{f}_i^{(j)}$ be the iterated path count flow of $D[V_i^{(j)}]$ with source set $S \cap V_i^{(j)}$ and sink set $T \cap V_i^{(j)}$. We let $\tilde{f}_i := \sum_j f_i^{[j]}$ be the path count flows associated with the $i$th partition and return as our solution the average path count flow across partitions; namely we return

$$\tilde{f} = \frac{1}{s} \cdot \sum_i \tilde{f}_i.$$

This flow is an $S$-$T$ flow since it is a convex combination of $S$-$T$ flows. We now argue that this flow is $\tilde{\Omega}(\frac{1}{h})$-optimal. Let $\hat{f}_i^{[j]}$ be the optimal flow on $D[V_i^{(j)}]$ with source set $S \cap V_i^{(j)}$ and sink set $T \cap V_i^{(j)}$. As our path count flows are $\Omega(\frac{1}{h})$-approximate, we know that

$$\mathrm{val}(\tilde{f}_i^{[j]}) \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot \mathrm{val}(\hat{f}_i^{[j]}).$$

Moreover, since every $h$-neighborhood is contained in one of the $V_i^{[j]}$, it follows that $\sum_{i,j} \mathrm{val}(\hat{f}_i^{[j]}) \geq \mathrm{val}(f^*)$ where $f^*$ is the optimal $S$-$T$ flow on $D$ with source set $S$ and sink set $T$. Thus, we conclude that

$$\begin{aligned}
\mathrm{val}(\tilde{f}) &= \frac{1}{s} \cdot \sum_{i,j} \tilde{f}_i^{[j]} \\
&\geq \Omega\left(\frac{1}{h}\right) \cdot \frac{1}{s} \cdot \sum_{i,j} \hat{f}_i^{[j]} \\
&\geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot \mathrm{val}(f^*).
\end{aligned}$$

Lastly, we argue the running time of our CONGEST algorithm. We describe how to compute $\tilde{f}_i$ for a fixed $i$. Again, $\tilde{f}_i$ on each part is simply a sum of $k = \tilde{\Theta}(h)$-many path count flows. To compute one of these path count flows we first compute the path counts $\{n_a\}_a$ on each part by applying Lemma 6.1 which takes $\tilde{O}(h^2)$ time. Next, we compute $\max_a n_a$ in $\tilde{O}(h)$ time by appealing to Lemma 5.6 and the fact that $\max_a n_a \leq O(n^h)$. Thus, computing each $\tilde{f}_i$ takes time $\tilde{O}(h^3)$ and since there are $\tilde{O}(h)$ of these, overall this takes $\tilde{O}(h^4)$ time. □

## 9.2 Deterministic Rounding of Flows in $h$-Layer DAGs

In the previous section we showed how to construct our iterated path count flows and that they were near-optimal but possibly fractional. In this section, we give the flow rounding algorithm that we will use to round our iterated path count flows to be integral. Specifically, in this section we show the following flow rounding algorithm.

**Lemma 9.6.** *There is a deterministic algorithm which, given a capacitated $h$-layer $S$-$T$ DAG $D$, $\varepsilon = \Omega(\frac{1}{poly(n)})$ and (possibly fractional) flow $f$, computes an integral $S$-$T$ flow $\hat{f}$ in:*

1. *Parallel time $\tilde{O}(h)$ with $m$ processors;*
2. *CONGEST time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^5 \cdot (\rho_{CC})^{10})$.*

*Furthermore, $val(\hat{f}) \geq (1-\varepsilon) \cdot val(f)$.*

Parts of the above parallel result are implied by the work of Cohen [15] while the CONGEST result is entirely new.

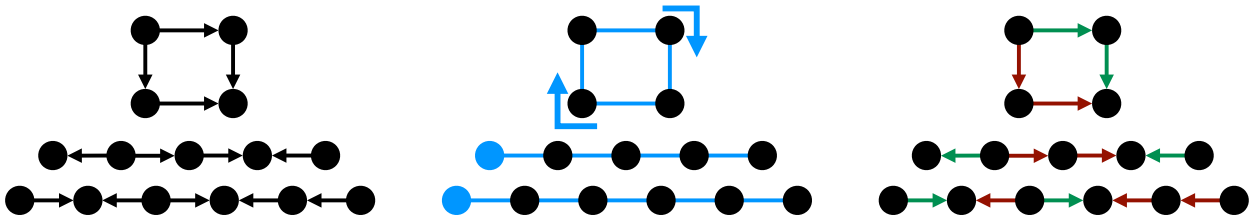
### 9.2.1 Turning Flows on $(1-\varepsilon)$-Near Eulerian Partitions

As discussed earlier, our rounding will round our flow from the least to most significant bit. To round the input flow on a particular bit we will consider the graph induced by the arcs which set this bit to 1. We then compute an oriented near-Eulerian partition of these edges and "turn" flow along each cycle and path consistently with its orientation. We will always turn flow so as to not increase the deficit of our flow.

We now formalize how we use our $(1-\varepsilon)$-near Eulerian partitions to update our flow. Given a path or cycle $H$, our flow update will carefully choose a subset of arcs of $H$ along which to increase flow (denoted $H^+$) and decrease flow along all other arcs of $H$. Specifically, let $H$ be an oriented cycle or path of a graph produced by forgetting about the directions in a digraph $D = (V, A)$. Then $H^+$ is illustrated in Figure 3 and defined as follows:

- Suppose $H$ is an oriented cycle. Then, we let $H^+$ be all arcs of $D$ in this cycle that point in the same direction as their orientation.
- Suppose $H = (s_H = v_0, v_1, v_2, \ldots)$ is an oriented path. We let $H^+$ be all arcs of $D$ in this path that point in the same direction as the one arc in $D$ incident to $s_H$ (i.e. the designated source of the path). That is either $(v_0, v_1)$ or $(v_1, v_0)$ are in $D$. In the former case we let $H^+$ be all arcs in $D$ of the form $(v_i, v_{i+1})$ for some $i$. In the latter case we let $H^+$ be all arcs in $D$ of the form $(v_{i+1}, v_i)$ for some $i$.

With our definition of $H^+$ in hand, we now define our flow updates as follows.



(a) Near Eulerian Partition $\mathcal{H}$. (b) Orientation of $\mathcal{H}$. (c) $H^+$ for each $H \in \mathcal{H}$.

Figure 3: An illustration of a near Eulerian partition $\mathcal{H}$ and $H^+$ for each $H \in \mathcal{H}$. 3a gives $\mathcal{H}$ which consists of one cycle and two paths. 3b gives the orientation of $\mathcal{H}$ where the source of each path is in blue. 3c gives $H^+$ (in green) and $H \setminus H^+$ (in red) for each $H \in \mathcal{H}$.

**Definition 9.7** ($(1-\varepsilon)$-Near Eulerian Partition Flow Update)**.** *Let $f$ be a flow in a capacitated DAG for which $f_a \in \{0, c\}$ for every $a \in A$ for some $c$ and let $\mathcal{H}$ be an oriented $(1-\varepsilon)$-near Eulerian partition of $supp(f)$ after forgetting about edge directions. Then if $H \in \mathcal{H}$, we define the flow $f_H$ on arc $a$:*

$$(f_H)_a := \begin{cases} 2c & \text{if } a \in H^+ \\ 0 & \text{otherwise} \end{cases}$$

*Likewise, we define the flow corresponding to $(f, \mathcal{H})$ as*

$$f_{\mathcal{H}} := \sum_{H \in \mathcal{H}} f_H.$$

The following shows that our flow update will indeed zero out the value of each bit on each edge while incurring a negligible deficit.

**Lemma 9.8.** *Let $f$ be a flow in a capacitated DAG $D$ with specified source and sink vertices $S$ and $T$ where $f_a \in \{0, c\}$ for every $a \in A$ for some $c$. Let $\mathcal{H}$ be an oriented $(1-\varepsilon)$-near Eulerian partition of $supp(f)$ after forgetting about edge directions. Then $f_{\mathcal{H}}$ (as defined in Definition 9.7) satisfies:*

1. *$(f_{\mathcal{H}})_a \in \{0, 2c\}$ for every $a \in A$;*
2. *$deficit(f_{\mathcal{H}}) \leq deficit(f) + 2\epsilon \cdot \sum_a f_a$.*

*Proof.* $(f_{\mathcal{H}})_a \in \{0, 2c\}$ holds by the definition of $f_{\mathcal{H}}$ and the fact that the elements of $\mathcal{H}$ are edge-disjoint.

We next argue that $\text{deficit}(f') \leq \text{deficit}(f) + 2\epsilon \cdot \sum_a f_a$. The basic idea is that each edge in the support of $f$ which does not appear in $A[\mathcal{H}]$ contributes its value to the deficit but any way of turning a cycle in $\mathcal{H}$ leaves the deficit unchanged and the way we chose to turn paths also leaves the deficit unchanged.

We let $f'$ be $f$ projected onto the arcs in $A[\mathcal{H}]$. That is, on arc $a$ the flow $f'$ takes value

$$f'_a =: \begin{cases} f_a & \text{if } a \in A[\mathcal{H}] \\ 0 & \text{otherwise} \end{cases}$$

We have that $\text{deficit}(f') \leq \text{deficit}(f) + 2\epsilon \cdot \sum_a f_a$ since each arc $a \notin A[\mathcal{H}]$ increases the deficit of $f'$ by at most $2f_a$ and, from Definition 8.1, there are at most $\varepsilon$-fraction of arcs not in $A[\mathcal{H}]$. Thus, to show our claim it suffices to argue that $\text{deficit}(f_{\mathcal{H}}) \leq \text{deficit}(f')$. For a given vertex $v$, we let $n_i(v)$ be the number of elements of $\mathcal{H}$ in which $v$ has in-degree 2. Similarly, we let $n_o(v)$ be the number of elements of $\mathcal{H}$ for which $v$ has out-degree 2. Lastly, we let $s(v)$ be the indicator of whether $v$ is the source of some path in $\mathcal{H}$ and $t(v)$ be the indicator of whether $v$ is the sink of a path in $\mathcal{H}$. Thus, we have

$$\text{deficit}(f', v) = 2c \cdot |n_i(v) - n_o(v)| + c \cdot (s(v) + t(v))$$

and so

$$\begin{aligned}\text{deficit}(f') &= \sum_v 2c \cdot |n_i(v) - n_o(v)| + c \cdot (s(v) + t(v)) \\ &= 2c|\mathcal{P}| + \sum_v 2c \cdot |n_i(v) - n_o(v)|\end{aligned}$$

On the other hand, we have

$$\text{deficit}(f_{\mathcal{H}}, v) \leq 2c \cdot |n_i - n_o| + 2c \cdot t(v)$$

and so

$$\begin{aligned}\text{deficit}(f_{\mathcal{H}}) &\leq \sum_v 2c \cdot |n_i(v) - n_o(v)| + 2c \cdot t(v) \\ &= 2c|\mathcal{P}| + \sum_v 2c \cdot |n_i(v) - n_o(v)|\end{aligned}$$

showing $\text{deficit}(f_{\mathcal{H}}) \leq \text{deficit}(f')$ as required. □

### 9.2.2 Extracting Integral $S$-$T$ Subflows

The last piece of our rounding deals with how to fix the damage that the accumulating deficit incurs. Specifically, as we round each bit we discard some edges, increasing our deficit. This means that after rounding all bits we are left with some (small) deficit. In this section we show how to delete flows that originate or end at vertices not in $S$ or $T$, thereby reducing the value of our flow by the deficit but guaranteeing that we are left with a legitimate $S$-$T$ flow.

**Lemma 9.9.** *Let $\hat{f}$ be an integral (not necessarily $S$-$T$) flow on an $h$-layer $S$-$T$ DAG. Then one can compute an $S$-$T$ integral flow $f'$ which is a subflow of $\hat{f}$ and satisfies $val(f') \geq val(\hat{f}) - deficit(\hat{f})$ in:*

1. *Parallel time $O(h)$ with $m$ processors;*
2. *CONGEST time $\tilde{O}(h)$.*

*Proof.* Our algorithm will simply delete out flow that originates not in $S$ or ends at vertices not in $T$. More formally, we do the following. We initialize our flow $f'$ to $\hat{f}$. Let $S = V_1, V_2, \ldots, V_{h+1} = T$ be the vertices in each layer of our input $S$-$T$ DAG $D = (V, A)$. Recall that we defined a flow $\hat{f}$ as an arbitrary function on the arcs so that $\hat{f}_a \leq U_a$ for every $a$. The basic idea of our algorithm is to

first push all "positive" deficit from left to right and then to push all "negative" deficit from right to left. The deficit will be non-increasing under both of these processes.

More formally, we push positive deficit as follows. For $i = 2, 3, \ldots h$ we do the following. For each $v \in V_i$, let

$$\text{deficit}^+(v) := \max\left(0, \sum_{a \in \delta^+(v)} f'_a - \sum_{a \in \delta^-(v)} f'_a\right)$$

be the positive deficit of $v$. Then, we reduce $\sum_{a \in \delta^+(v)} f'_a$ to be equal to $\sum_{a \in \delta^-(v)} f'_a$ by arbitrarily (integrally) reducing $f'_a$ for some subset of $a \in \delta^+(v)$.

It is easy to see by induction that at this point we have $\text{deficit}^+(v) = 0$ for all $v \notin S \cup T$. Likewise, we have that $\sum_{v \notin S \cup T} \text{deficit}^+(v)$ is non-increasing each time we iterate the above. Thus, if $\text{deficit}^+$ is the initial value of $\sum_{v \notin S \cup T} \text{deficit}^+(v)$ then in the last iteration of the above we may decrease the flow into $T$ by at most $\text{deficit}(\hat{f})$.

Next, we do the same thing symmetrically to reduce the negative deficits. For $i = h, h-1, \ldots, 2$ we do the following for each $v \in V_i$. Let

$$\text{deficit}^-(v) := \max\left(0, \sum_{a \in \delta^-(v)} f'_a - \sum_{a \in \delta^+(v)} f'_a\right)$$

be the negative deficit of $v$. Then, we reduce $\sum_{a \in \delta^-(v)} f'_a$ to be equal to $\sum_{a \in \delta^+(v)} f'_a$ by arbitrarily (integrally) reducing $f'_a$ for some subset of $a \in \delta^-(v)$. Notice that this does not increase $\text{deficit}^+(v)$ for any $v \notin S \cup T$.

Symmetrically to the positive deficit case, it is easy to see that at the end of this process we have reduced $\text{deficit}^-(v)$ to 0 for every $v \notin S \cup T$ while reducing the flow out of $S$ by at most $\text{deficit}(\hat{f})$.

Thus, at the end of this process we have an $S$-$T$ integral flow $f'$ whose value is at least $\text{val}(\hat{f}) - \text{deficit}(\hat{f})$. Implementing the above in the stated running times is trivial; the only caveat is that updating a flow in CONGEST requires updating it for both endpoints but since the flow is integral and we reduce it integrally, this can be done along a single arc in time $O(\log U_{\max}) = \tilde{O}(1)$ by assumption. □

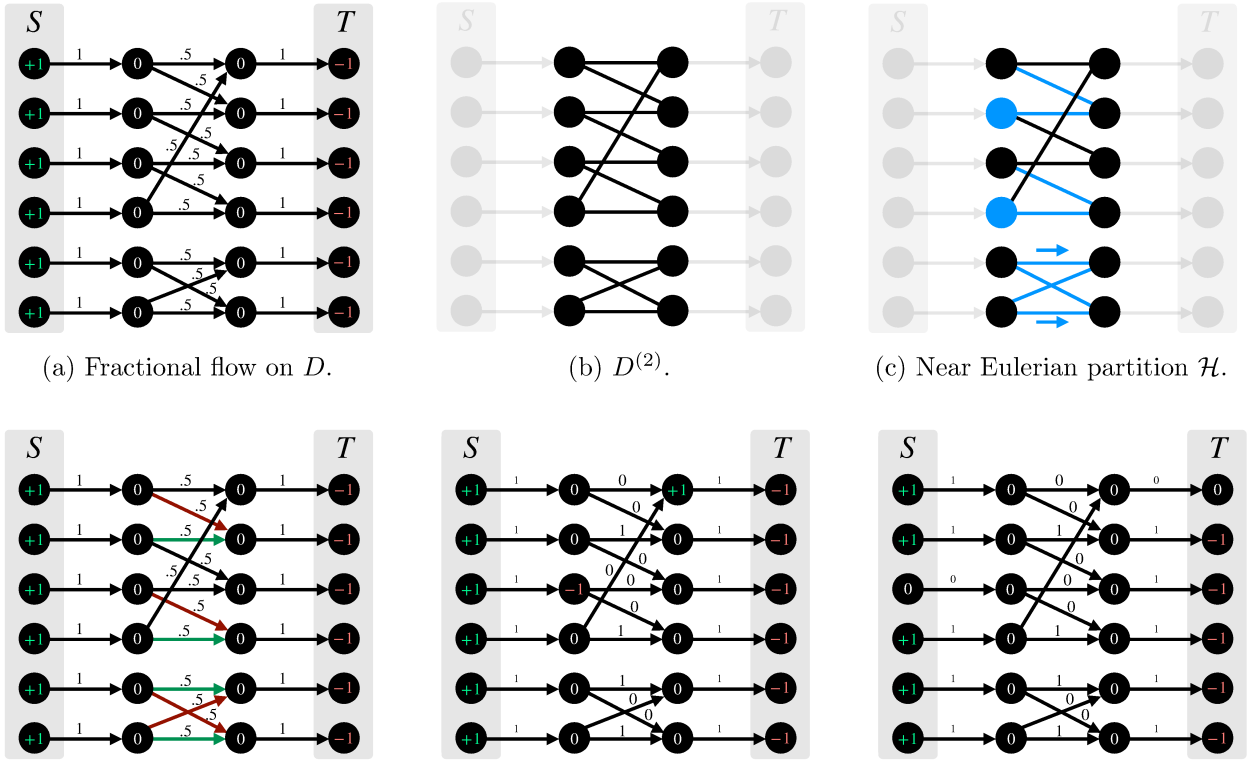
### 9.2.3 Flow Rounding Algorithm

Having defined the flow update we use for each $(1-\varepsilon)$-near Eulerian partition and how to extract a legitimate $S$-$T$ flow from the resulting rounding, we conclude with our algorithm for rounding flows from least to most significant bit. Our algorithm is given in Algorithm 1 and illustrated in Figure 4.

We conclude that the above rounding algorithm rounds with negligible loss in the value.

**Lemma 9.6.** *There is a deterministic algorithm which, given a capacitated $h$-layer $S$-$T$ DAG $D$, $\varepsilon = \Omega(\frac{1}{poly(n)})$ and (possibly fractional) flow $f$, computes an integral $S$-$T$ flow $\hat{f}$ in:*

1. *Parallel time $\tilde{O}(h)$ with $m$ processors;*
2. *CONGEST time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^5 \cdot (\rho_{CC})^{10})$.*




(a) Fractional flow on $D$. (b) $D^{(2)}$. (c) Near Eulerian partition $\mathcal{H}$.

(d) Flow update using $\mathcal{H}$. (e) Flow after flow update. (f) Returned integral $S$-$T$ flow.

Figure 4: An example of our flow rounding algorithm on digraph $D$ with unit capacities. 4a gives the input flow where arcs are labelled with their flow and vertices are labelled with their deficit. 4b gives $D^{(2)}$, the graph induced by all arcs with flow value .5. 4c gives our oriented near Eulerian partition of $D^{(2)}$ (in blue). 4d shows how we update our flow based on the near Eulerian partition. 4e gives the result of this flow update; notice that some vertices not in $S$ and $T$ have non-zero deficit. 4f gives the $S$-$T$ subflow we return where only vertices in $S$ and $T$ have non-zero deficit.

**Algorithm 1** Deterministic Flow Rounding

**Input:** $h$-layer DAG $D$, $S$-$T$ flow $f = \sum_{i=0} f^{(i)}$ where $(f^{(i)})_a \in \{0, 2^{\log(U_{\max})-i}\}$ for every $a$, $i$.
**Output:** integral $S$-$T$ flow $\hat{f}$.
$\hat{f} \leftarrow \sum_{i=0}^{k} f^{(i)}$ for $k = \Theta(\log n + \log(U_{\max}))$. ▷ Truncate lower order bits of input flow
**for** $i = k, \ldots, \log(U_{\max})$ **do**
  Let $\hat{f} = \sum_j \hat{f}^{(j)}$ be the bitwise flow decomposition of $\hat{f}$ (defined in Section 2) and let $D^{(i)}$ be the undirected graph induced by the support of $f^{(i)}$.
  Compute an oriented $(1-\varepsilon')$-near Eulerian partition $\mathcal{H}$ of $D^{(i)}$ (using Lemma 8.2 with $\epsilon' = 0$ for the parallel algorithm and $\epsilon' = \Theta\left(\frac{\varepsilon}{h \cdot \log n}\right)$ for the CONGEST algorithm).
  $\hat{f} \leftarrow \hat{f}^{(i)}_{\mathcal{H}} + \sum_{j<i} \hat{f}^{(j)}$ (as defined in Definition 9.7). ▷ Turn flow along $\mathcal{H}$
Let $\hat{f}$ be an $S$-$T$ subflow of $\hat{f}$ (compute using Lemma 9.9).
**return** $\hat{f}$.

*Furthermore,* $val(\hat{f}) \geq (1-\varepsilon) \cdot val(f)$.

*Proof.* We use Algorithm 1.

We first argue that the above algorithm returns an integral flow. Notice that by the fact that we initialize $\hat{f}$ to $\sum_{i=0}^{k}$ it follows that for $j > k$ on every $a$ we have $\hat{f}_a^{(j)} = 0$ just before the first iteration of our algorithm. Thus, to argue that the returned flow is integral it suffices to argue that if $\hat{f}^{(j)}$ is the $j$th bit flow of $\hat{f}$ just after the $i$th iteration then for $j \leq i$ we have $f_a^{(j)} = 0$ for every $a$. However, notice that, by Lemma 9.8, after we update $\hat{f}$ each $\hat{f}_a^{(i)}$ value is either doubled or set to 0, meaning that $\hat{f}_a^{(i)} = 0$ after this update.

Next, we argue that $\text{val}(\hat{f}) \geq (1-\varepsilon) \cdot \text{val}(f)$. By Lemma 9.9 it suffices to argue that just before we compute our $S$-$T$ subflow of $\hat{f}$ we have $\text{deficit}(\hat{f}) \leq \varepsilon \cdot \text{val}(f)$. We may set the constant in $k = \Theta(\log n + \log(U_{\max}))$ to be appropriately large so that when we initialize $\hat{f}$ we reduce the flow value on each arc by at most $\frac{1}{\text{poly}(n)}$. It follows that at this point $\text{deficit}(\hat{f}) \leq \frac{2}{\text{poly}(n)} \sum_a f_a$. Similarly, by Lemma 9.8 in the $i$th iteration of our algorithm we increase the deficit of $\hat{f}$ by at most $2\epsilon' \sum_a \hat{f}_a^{(i)} \leq 2\epsilon' \sum_a f_a$.

For our parallel algorithm, since we have $\epsilon' = 0$, it immediately then follows that $\text{deficit}(\hat{f}) \leq \frac{2}{\text{poly}(n)} \sum_a f_a \leq \varepsilon \cdot \text{val}(f)$ by our assumption that $\varepsilon = \Omega(\frac{1}{\text{poly}(n)})$. For our CONGEST algorithm we choose $\varepsilon' = \Theta(\frac{\varepsilon}{h \log n})$ for some appropriately small constant. Since we have $\Theta(\log n)$ iterations it follows that after all of our iterations (but before we compute an $S$-$T$ subflow) it holds that $\text{deficit}(\hat{f}) \leq \frac{\varepsilon}{h} \cdot \sum_a f_a \leq \varepsilon \cdot \text{val}(f)$ where the last inequality follows from the fact that our flow is $h$-length.

Lastly, we argue that the algorithm achieves the stated running times. The above algorithm runs for $k = \Theta(\log n)$ iterations. The computation in each iteration is dominated by computing a $(1-\varepsilon')$-near Eulerian partition. For our parallel algorithm, computing each $(1-\varepsilon')$-near Eulerian partition takes time at most $\tilde{O}(1)$ with $m$ processors by Lemma 8.2. For our CONGEST algorithm computing each $(1-\varepsilon')$-near Eulerian partition takes time at most $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^5 \cdot (\rho_{CC})^{10})$ by Lemma 8.2. Lastly, we must compute an $S$-$T$ subflow of $\hat{f}$ which by Lemma 9.9 takes $O(h)$ parallel time with $m$ processors or $\tilde{O}(h)$ CONGEST time. □

### 9.3 Deterministic Blocking Integral Flows

Having shown that the iterated path count flow is near-optimal and fractional but that we can efficiently round fractional flows to be integral, we conclude with our algorithm to compute a blocking integral flow by repeatedly rounding iterated path count flows.

**Lemma 9.1.** *There is a deterministic algorithm which, given a capacitated $h$-layer $S$-$T$ DAG $D$, computes an integral $S$-$T$ flow that is blocking in:*

1. *Parallel time $\tilde{O}(h^3)$ with $m$ processors;*
2. *CONGEST time $\tilde{O}(h^6 \cdot (\rho_{CC})^{10})$.*

*Proof.* We repeatedly compute the iterated path count flow, round it to be integral, reduce capacities appropriately and repeat. We will return flow $f$ initialized to 0 on all arcs.

Specifically, we repeat the following $\tilde{\Theta}(h)$ times. Apply Lemma 9.5 to compute a $\tilde{\Omega}(1/h)$-approximate (possibly fractional) flow $\tilde{f}$. Next, apply Lemma 9.6 with $\varepsilon = .5$ to round this to an

integral flow $\hat{f}$ where $\text{val}(\hat{f}) \geq \frac{1}{2}\text{val}(\tilde{f})$. Next, we update $f$ to $f + \hat{f}$ and for each arc $a$ we reduce $U_a$ by $\hat{f}_a$.

After each time we iterate the above $\tilde{\Theta}(h)$ times we must reduce the value of the optimal solution by at least a multiplicative $\frac{1}{2}$ since otherwise $f$ would be a flow with value greater than the max $S$-$T$ flow in the graph at the beginning of these iterations. Since the optimal solution is at most $m \cdot U_{\max}$, it follows that we need only iterate the above $\tilde{\Theta}(h)$ times until the value of the optimal $S$-$T$ flow is 0 which is to say that $f$ is a blocking flow.

By Lemma 9.5 and Lemma 9.6 each of the above iterations takes parallel time $\tilde{O}(h^2)$ with $m$ processors and CONGEST time $\tilde{O}(h^5 \cdot (\rho_{CC})^{10})$, giving the stated running times. □

# 10 $h$-Length $(1+\epsilon)$-Lightest Path Blockers

In this section we show how to efficiently compute our main subroutine for our multiplicative-weights-type algorithm; what we call $h$-length $(1+\epsilon)$-lightest path blockers. We will use the blocking integral flow primitives of Section 7 for our randomized algorithm and that of Section 9 for our deterministic algorithm.

Our $(1+\epsilon)$-lightest path blockers are defined below. In what follows, $\lambda$ is intuitively a guess of $d_w^{(h)}(S,T)$. Also, in the following recall that if $f$ is an $h$-length flow then $f$ assigns flow values to entire paths (rather than just arcs as a non-length-constrained flow does). As such the support of $f$, $\text{supp}(f)$, is a collection of paths. However, as mentioned earlier, for an $h$-length flow $f$, we will use $f(a)$ as shorthand for $\sum_{P \ni a} f_P$.

**Definition 10.1** ($h$-length $(1+\epsilon)$-Lightest Path Blockers)**.** *Let $G = (V, E)$ be a graph with lengths $\ell$, weights $w$ and capacities $U$. Fix $\epsilon > 0$, $h \geq 1$, $\lambda \leq d_w^{(h)}(S,T)$ and $S, T \subseteq V$. Let $f$ be an $h$-length integral $S$-$T$ flow. $f$ is an $h$-length $(1+\epsilon)$-lightest path blocker if:*

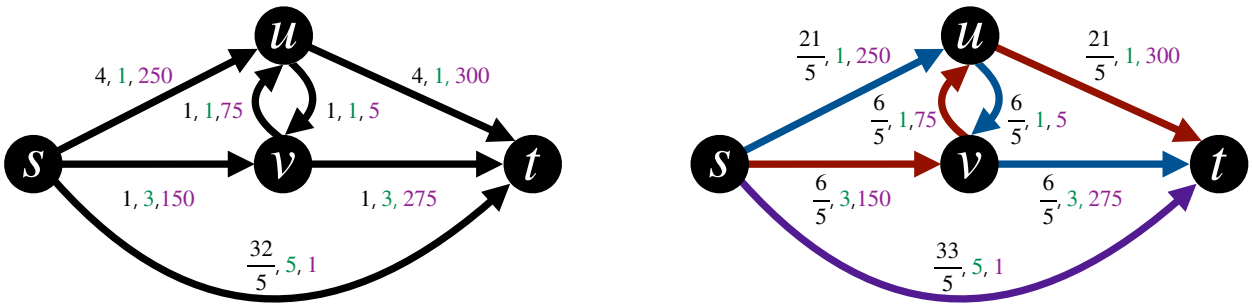
1. ***Near-Lightest:*** *$P \in \text{supp}(f)$ has weight at most $(1+2\epsilon) \cdot \lambda$;*
2. ***Near-Lightest Path Blocking:*** *If $P' \in \mathcal{P}_h(S,T)$ has weight at most $(1+\epsilon) \cdot \lambda$ then there is some $a \in P'$ where $f(a) = U_a$.*

The main theorem of this section we show is how to compute our $(1+\epsilon)$-lightest path blockers efficiently.

**Theorem 10.1.** *Given digraph $D = (V, A)$ with lengths $\ell$, weights $w$, capacities $U$, length constraint $h \geq 1$, $\varepsilon > 0$, $S, T \subseteq V$ and $\lambda \leq d_w^{(h)}(S,T)$, one can compute an $h$-length $(1+\epsilon)$-lightest path blocker in:*

1. *Deterministic parallel time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^{16})$ with $m$ processors*
2. *Randomized CONGEST time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^{16})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\varepsilon^5} \cdot h^{16} + \frac{1}{\varepsilon^3} \cdot h^{15} \cdot (\rho_{CC})^{10}\right)$.*

The main idea for computing these objects is to reduce finding them to computing a series of blocking flows in a carefully constructed "length-weight expanded DAG." In particular, by rounding arc weights up to multiples of $\frac{\varepsilon}{h}\lambda$ we can essentially discretize the space of weights. Since each path has at most $h$ arcs, it follows that this increases the length of a path by at most only $\lambda\varepsilon$. This discretization allows us to construct DAGs from which we may extract blocking flows which we then project back into $D$ and then "decongest" so as to ensure they are feasible flows.




(a) Digraph $D$ with $w$.

(b) Digraph $D$ with $\tilde{w}$.

Figure 5: An illustration of how we round weights according to $\varepsilon$, $\lambda$ and $h$. Here $h = 5$, $\lambda = 6$ and $\varepsilon = .5$ and so we round to multiples of $\frac{\varepsilon}{h}\lambda = \frac{3}{5}$. 5a gives our input DAG where each arc is labeled with its weight, then length then capacity and 5b gives the weights after we round them where we color each lightest 5-length path from $s$ to $t$.

### 10.1 Length-Weight Expanded DAG

We now formally define the length-weight-expanded DAGs on which we compute blocking integral flows. Roughly, the length-weight expanded graph will create many copies of vertices and organize them into a grid where moving further down in rows corresponds to increases in length and moving further along in columns corresponds to increases in weight.

Let $D = (V, A)$ be a digraph with specified source and sink vertices $S$ and $T$, lengths $\ell$, weights $w$, capacities $U$ and a parameter $\lambda \leq d_w^{(h)}(S, T)$. We let $\tilde{w}$ be $w$ but rounded up to the nearest multiple of $\frac{\varepsilon}{h} \cdot \lambda$. That is, for each $a \in A$ we have

$$\tilde{w}_a = \frac{\varepsilon \cdot \lambda}{h} \cdot \left\lceil w_a \cdot \frac{h}{\varepsilon \cdot \lambda} \right\rceil$$

See Figure 5 for an illustration of $\tilde{w}$.

Next, we define the length-weight expanded DAG $D^{(h,\lambda)} = (V', A')$ with capacities $U'$. See Figure 6 for an illustration of $D^{(h,\lambda)}$.

- **Vertices:** We construct the vertices $V'$ as follows. For each each vertex $v \in V$ we make $\kappa = h \cdot (\frac{h}{\epsilon} + 2h)$ copies of $v$, where we let $v(x, h')$ be one of these vertices; here $x$ ranges over all multiples of $\frac{\varepsilon}{h} \cdot \lambda$ up to $(1 + 2\varepsilon) \cdot \lambda$ (of which there are $\frac{h}{\epsilon} + 2h$) and $h' \leq h$. Intuitively, there will be a path from a copy of a vertex $s \in S$ to a vertex $v(x, h')$ iff there is a path with exactly $x$ weight (according to $\tilde{w}$) and $h'$-length from $s$ to $v$ in $D$.

- **Arcs:** We construct the arcs $A'$ as follows. For each each vertex $v \notin T$ and each $a = (v, u) \in \delta^+(v)$ we do the following. For each copy $v(x, h')$ of $v$ we add an arc to $A'$ from $v(x, h')$ to $u(x + \tilde{w}_a, h' + \ell_a)$ provided $u(x + \tilde{w}_a, h' + \ell_a)$ is actually a vertex in $V'$. That is, provided $x + \tilde{w}_a \leq (1 + 2\varepsilon) \cdot \lambda$ and $h' + \ell_a \leq h$. We say that the arc $v(x, h')$ to $u(x + \tilde{w}_a, h' + \ell_a)$ in $A'$ is a copy of arc $a$. For a given $a \in A$, we let $A'(a)$ give all copies of arc $a$ that are in $A'$.

- **Capacities:** We construct the capacities $U'$ as follows. For low capacity arcs we set the capacity of all copies to 1; for high capacity arcs we evenly distribute the capacity across all

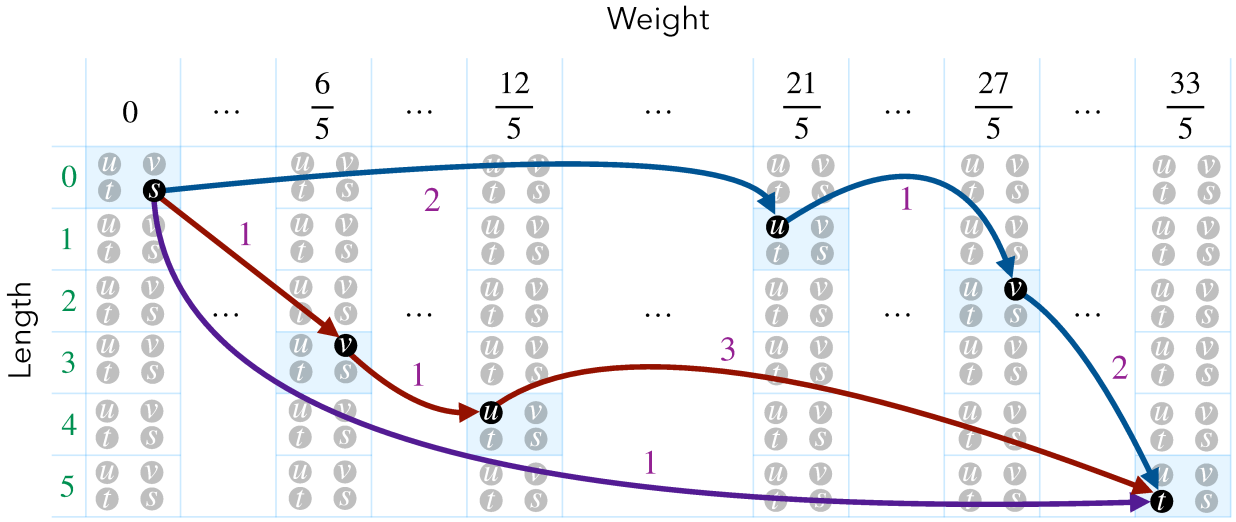



Figure 6: An illustration of $D^{(h,\lambda)}$ where $D$ and the parameters we use are given by Figure 5, $\kappa = 100$, $S = \{s\}$ and $T = \{t\}$. Copy $v(x, h')$ of vertex $v$ is in the $(x, h')$th grid cell and each arc is labelled with its capacity. We only illustrate the subgraph between $s(0, 0)$ and $t(\frac{33}{5}, 5)$. Each path is colored according to the path in Figure 5b of which it is a copy. Notice that the graph induced by all 5-length lightest paths in Figure 5b is not a DAG but $D^{(h,\lambda)}$ is.

copies. Specifically, suppose arc $a' \in A'$ is a copy of arc $a \in A$. Then if $0 < U_{uv} \leq \kappa$ we let $U'_{a'} = 1$. Otherwise, we let $U'_{a'}$ have capacity $\lfloor U_a/\kappa \rfloor$. As we will see later in our proofs, this rebalancing of flows will guarantee that when we "project" a flow from $D^{(h,\lambda)}$ to $D$, the only arcs that end up overcapacitated in $D$ are arcs with capacity at most $\kappa$. This, in turn, will allow us to argue that the conflict graph on which we compute an MIS is small.

We let $V'(S)$ and $V'(T)$ be all copies of $S$ and $T$ in $D^{(h,\lambda)}$ and we delete any vertex from $D^{(h,\lambda)}$ that does not lie on a $V'(S)$ to $V'(T)$ path. This will guarantee that the resulting digraph is indeed an $V'(S)$-$V'(T)$ DAG.

Lastly, we clarify what it means for a path to have its copy in $D^{(h,\lambda)}$. Suppose $P = (a_1, a_2, \ldots)$ is a path in $D$ that visits vertices $s = v_1, v_2, \ldots, v_k = t$ in $D$ and let $\tilde{w}_i$ and $\ell_i$ be the weight (according to $\tilde{w}$) and length of $P$ summed up to the $i$th vertex it visits. Then we let $a'_i$ be the arc from $v_i(\tilde{w}_i, \ell_i)$ to $v_{i+1}(\tilde{w}_i + \tilde{w}_{a_i}, \ell_i + \ell_{a_i})$. If $a'_i$ is in $D^{(h,\lambda)}$ for every $i$ then we call $P' = (a'_1, a'_2, \ldots)$ the copy of $P$ in $D^{(h,\lambda)}$. Observe that a path in $D$ has at most one copy in $D^{(h,\lambda)}$ but every path in $D^{(h,\lambda)}$ is the copy of some path in $D$.

The following summarizes the key properties of our length-weight expanded digraphs.

**Lemma 10.2.** *Let $D = (V, A)$ be a digraph with weights $w$, $S, T \subseteq V$ and some $\lambda \leq d^{(h)}_w(S, T)$. Let $D^{(h,\lambda)} = (V', A)$ be the length-weight expanded digraph of $D$. Then $D^{(h,\lambda)}$ is an $h$-layer $V'(S)$-$V'(T)$ DAG which satisfies*

1. ***Few Arc Copies:*** *$|A'(a)| \leq O(\frac{h^2}{\varepsilon})$.*
2. ***Forward Path Projection:*** *For each path $P$ in $D$ from $S$ to $T$ of weight at most $\lambda \cdot (1 + \epsilon)$ according to $w$, there is a copy of $P$ in $D^{(h,\lambda)}$ from $V'(S)$ to $V'(T)$.*

3. ***Backward Path Projection****: If $P'$ is a $V'(S)$ to $V'(T)$ path in $D^{(h,\lambda)}$ then it is a copy of a path with weight at most $(1+2\epsilon)\cdot\lambda$ according to $w$.*

4. ***Optimal Flow Preserving****: the maximum S-T flow on $D^{(h,\lambda)}$ has value at least $\Omega(\frac{\varepsilon}{h^2})$ times that of the maximum flow on $D$.*

*Proof.* First, we argue that $D^{(h,\lambda)}$ is indeed a DAG. To see this, observe that if $a'$ is an arc in $A'$ from $v(x_1,h_1)$ to $v(x_2,h_2)$ then by construction it must be the case that $h_1 < h_2$. It follows that $D^{(h,\lambda)}$ has no cycles and has at most $h$ layers. Next, observe that $D^{(h,\lambda)}$ is a $V'(S)$-$V'(T)$ DAG by construction since we deleted any any vertices that do not lie on a path between $V'(S)$ and $V'(T)$. Additionally, we have $|A'(a)| \leq O(\frac{h^2}{\varepsilon})$ for every $a$ since each vertex has at most $O(\frac{h^2}{\varepsilon})$-many copies.

Next, consider an arc $a$ with weight $w_a$ according to $w$ and weight $\tilde{w}_a$ according to $\tilde{w}$. Observe that since we are rounding arc weights up we have $w_a \leq \tilde{w}_a$. Combining this with the fact that we are rounding to multiples of $\frac{\epsilon}{h}\cdot\lambda$ we have that

$$w_a \leq \tilde{w}_a \leq w_a + \frac{\epsilon}{h}\cdot\lambda \tag{3}$$

We next argue our forward path projection property. That is, for each $h$-length path $P$ in $D$ from $S$ to $T$ of weight at most $\lambda\cdot(1+\epsilon)$ according to $w$, there is a copy of $P$ in $D^{(h,\lambda)}$ from $V'(S)$ to $V'(T)$. First, observe that $P$ consists of at most $h$-many edges and so applying Equation (3), its weight according to $\tilde{w}$ is at most $\lambda\cdot(1+\epsilon)+h\cdot\frac{\epsilon}{h}\cdot\lambda = \lambda\cdot(1+2\epsilon)$. Next, observe that since $P$ has weight at most $\lambda\cdot(1+2\epsilon)$ according to $\tilde{w}$, it must have a copy in $D^{(h,\lambda)}$. In particular, suppose $P=(a_1,a_2,\ldots)$ visits vertices $s=v_1,v_2,\ldots,v_k=t$ in $D$ and let $\tilde{w}_i$ and $\ell_i$ be the weight (according to $\tilde{w}$) and length of $P$ up to the $i$th vertex it visits. Then $D^{(h,\lambda)}$ always includes the arc from $v_i(\tilde{w}_i,\ell_i)$ to $v_{i+1}(\tilde{w}_i+\tilde{w}_{a_i},\ell_i+\ell_{a_i})$ since $\tilde{w}_i \leq (1+2\epsilon)\lambda$ and $\ell_i \leq h$ for every $i$.

We argue our backward path projection property. That is, if $P'$ is a $V'(S)$ to $V'(T)$ path in $D^{(h,\lambda)}$ then it is a copy of a path with weight at most $(1+2\epsilon)\cdot\lambda$ in $D$ according to $w$. Since each arc in $D^{(h,\lambda)}$ is a copy of some arc in $D$, we know that $P'$ is a copy of *some* path in $D$. Moreover, since we let $v(x,h')$ only range over $x \in \frac{h}{\epsilon}+2h$, it follows that the weight of this path according to $\tilde{w}$ is at most $(1+2\varepsilon)\cdot\lambda$. However, since weights according to $\tilde{w}$ are only larger than those according to $w$ by Equation (3), it follows that $P'$ is a copy of a path with weight at most $(1+2\varepsilon)\cdot\lambda$ according to $w$.

Lastly, to see the optimal flow preserving property notice that if $f^*$ is the optimal flow on $D$ then by how chose the capacities of $D^{(h,\lambda)}$ we have that the flow that gives path $P'$ in $D^{(h,\lambda)}$ value $\Theta(\frac{\varepsilon}{h^2})\cdot f^*_P$ where $P'$ is the copy of $P$ is indeed a feasible flow in $D^{(h,\lambda)}$. □

## 10.2 Decongesting Flows

Part of what makes using our length-weight expanded digraph non-trivial is that when we compute a flow in it and then project this flow back into $D$, the projected flow might not respect capacities. However, this flow will only violate capacities to a bounded extent and so in this section we show how to resolve such flows at a bounded loss in the value of the flow. In the below we say that an $h$-length flow $\hat{f}$ is $\alpha$-congested if any arc $a$ where $\hat{f}(a) > U_a$ satisfies $\hat{f}(a) \leq \alpha$.

**Lemma 10.3.** *There is a deterministic algorithm that, given a digraph $D=(V,A)$ with capacities $U$, a length constraint $h \geq 1$, $S,T \subseteq V$ and an $h$-length $\alpha$-congested S-T integral flow $\hat{f}$, computes an S-T $h$-length integral flow $f$ where $val(f) \geq \frac{1}{\alpha^2h^2}\cdot val(\hat{f})$ in:*

1. *Parallel time* $\tilde{O}(\alpha^2 \cdot h)$ *with* $m$ *processors;*

2. *CONGEST time* $\tilde{O}(\alpha^3 \cdot h^3)$.

*Proof.* The basic idea is to consider the conflict graph induced by our flow paths and then to compute a approximate maximum-weighted independent set among these flow paths where flow paths are weighted according to their flow value.

Specifically, construct our conflict graph $G' = (V', E')$ of $\text{supp}(\hat{f})$ as follows. $V' = \text{supp}(\hat{f})$ has a vertex for each path in the support of $\hat{f}$. We say that $P_1$ and $P_2$ in $\text{supp}(\hat{f})$ *conflict* if there is some arc $a$ in both $P_1$ and $P_2$ such that $\hat{f}(a) > U_a$. Then we add edge $\{P_1, P_2\}$ to $E'$ iff $P_1$ and $P_2$ conflict.

Observe that since each path in $\text{supp}(\hat{f})$ consists of at most $h$ arcs and since $\hat{f}$ is $\alpha$-congested, we know that the maximum degree in $G'$ is at most $h \cdot \alpha$.

We then apply Theorem 5.2 to $G'$ to compute a $\frac{1}{h\alpha}$-approximate maximum independent set in $G'$ in deterministic CONGEST time $\tilde{O}(h\alpha)$ with the node weight of $P \in \text{supp}(\hat{f})$ as $\hat{f}_P$. Let $\mathcal{I}$ be this independent set and let $f = \sum_{P \in \mathcal{I}} \hat{f}_P$ be the flow corresponding to this set. We return $f$.

We first argue that $\text{val}(f) \geq \frac{\text{val}(\hat{f})}{\alpha^2 h^2}$ Since the total node weight in $G'$ is $\text{val}(\hat{f})$ and the maximum degree in $G'$ is $\alpha \cdot h$, it follows that the maximum independent set in $G'$ has node weight at least $\frac{\text{val}(\hat{f})}{\alpha h}$. Since $\mathcal{I}$ is $\frac{1}{\alpha h}$-approximate, we conclude that $f$ has $\text{val}(f) \geq \frac{\text{val}(\hat{f})}{\alpha^2 h^2}$.

Lastly, we argue that we achieve the claimed running times. Notice that the total number of vertices in $G'$ is at most $m \cdot \alpha$ because each congested arc $a$ where $U_a < \hat{f}(a) \leq \alpha$ is contained in at most $\alpha$ integral flow paths. Hence, we can simulate any CONGEST algorithm in $G'$ with at most $\alpha$ overhead. Theorem 5.2 tells us that we can compute $\mathcal{I}$ in time at most $\tilde{O}(\alpha \cdot h)$ in $G'$, giving our parallel running time.

It remains to describe how to simulate $G'$ in $D$ in CONGEST. We keep the following invariant: if a node $P_1$ in $G'$ receives a message, we make sure that all vertices $v \in P_1$ in $G$ receive the same message too. Because of this, any vertex $v \in P_1$ in $G$ can determine what $P_1$ as a node in $G'$ will do next. Let us assume that each message in $G'$ from $P_1$ to $P_2$ is of the form $(\texttt{msg}, P_1, P_2)$. To simulate sending $(\texttt{msg}, P_1, P_2)$ in $G$, a vertex $v_1 \in P_1$ first forwards $(\texttt{msg}, P_1, P_2)$ through $P_1$ to make sure that every node in $P_1$ gets this message. Let $v_2 \in P_1 \cap P_2$ be a common vertex in both $P_1$ and $P_2$. Then, $v_2$ forwards $(\texttt{msg}, P_1, P_2)$ through $P_2$. After we are done simulating all messages sent in $G'$, our invariant is maintained.

Now, we analyze the overhead of simulating one round of $G'$ in $G$. The dilation for simulating sending each message in $G'$ is clearly $O(h)$. Next, we analyze the congestion. Each arc $a$ is contained in at most $\max\{U_a, \alpha\} \leq \alpha U_a$ paths. For each such path $P$, there are at most $\alpha h$ messages needed to sent through $a$ because the maximum degree in $G'$ at most $\alpha h$. Therefore, the congestion is at most $\frac{\alpha U_a \cdot \alpha h}{U_a} = \alpha^2 h$. Note that, here (and nowhere else in this work) we rely on the fact that we may send $O(U_a)$ messages over an arc $a$ with capacity $U_a$ in one round of CONGEST.

To conclude, the deterministic simulation overhead is at most dilation times congestion which is at most $O(h) \cdot \alpha^2 h = O(\alpha^2 h^2)$. Combining this simulation with the $\tilde{O}(\alpha \cdot h)$ running time of our approximate maximal independent set algorithm gives our CONGEST running time. □

### 10.3 Computing $h$-Length $(1+\varepsilon)$-Lightest Path Blockers

Having described our length-weight expanded DAGs, their properties and how to decongest flows that we compute using them, we now use these primitives to build our $h$-length $(1+\epsilon)$-lightest path

blockers. Again, the basic idea is to compute the length-weight expanded DAG $D^{(h,\lambda)}$, compute blocking flows in $D^{(h,\lambda)}$, project these back into $D$, decongest the resulting flows and then repeat. Algorithm 2 gives our algorithm. We prove its properties below.

**Algorithm 2** $(1+\epsilon)$-Lightest Path Blocker

**Input:** $D = (V, A)$ with weights $w$, lengths $\ell$, capacities $U$, $h \geq 1$, $S, T \subseteq V$, $\lambda > 0$ and $\varepsilon > 0$.
**Output:** $h$-length $(1+\varepsilon)$-lightest path blocker $f$.
Initialize solution $f$ to be 0 on all arcs.
Let $D^{(h,\lambda)} = (V', A')$ be the length-weight expanded digraph of $D$ with capacities $\hat{U} = U$
**for** $\tilde{\Theta}(\frac{h^7}{\varepsilon^2})$ repetitions **do**
  **Blocking Flows**: Let $f'$ be a blocking integral flow in $D^{(h,\lambda)}$ with capacities $\hat{U}$ (compute using Lemma 7.2 with randomness or Lemma 9.1 deterministically).
  **Project Into** $D$: Let $\tilde{f}$ be the $h$-length flow that gives path $P$ value $f'_{P'}$ where $P'$ is the copy of $P$ in $D^{(h,\lambda)}$.
  **Decongest Flow**: Let $\hat{f}$ be the result of decongesting $\tilde{f}$ with Lemma 10.3.
  For each copy $a' \in A'$ of $a \in A$ update capacities as $\hat{U}_{a'} = \hat{U}_{a'} - \hat{f}(a)$.
  Update $f = f + \hat{f}$.
**return** $f$.

**Theorem 10.1.** *Given digraph $D = (V, A)$ with lengths $\ell$, weights $w$, capacities $U$, length constraint $h \geq 1$, $\varepsilon > 0$, $S, T \subseteq V$ and $\lambda \leq d^{(h)}_w(S, T)$, one can compute an $h$-length $(1+\epsilon)$-lightest path blocker in:*

1. *Deterministic parallel time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^{16})$ with $m$ processors*
2. *Randomized CONGEST time $\tilde{O}(\frac{1}{\varepsilon^5} \cdot h^{16})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\varepsilon^5} \cdot h^{16} + \frac{1}{\varepsilon^3} \cdot h^{15} \cdot (\rho_{CC})^{10}\right)$.*

*Proof.* We first argue that $f$ is a $h$-length $(1+\varepsilon)$-lightest path blocker (Definition 10.1). $f$ is an integral $h$-length $S$-$T$ flow by construction. Moreover, the support of $f$ is near-lightest by the backward path projection property of $D^{(h,\lambda)}$, as stated in Lemma 10.2.

Thus, it remains to argue the near-lightest path blocking property of $f$ and, in particular that if $P \in \mathcal{P}_h(S, T)$ is a path in $D$ and $P$ has weight at most $(1+\varepsilon) \cdot \lambda$ according to $w$ then there is some $a \in P$ where $f(a) = U_a$. Towards this, observe that by the forward path projection property as stated in Lemma 10.2, such a path $P$ has copy in $D^{(h,\lambda)}$. By how we construct $f$, it follows that to show $f(a) = U_a$ for some $a$, it suffices to show that $\hat{U}_a = 0$ by the end of our algorithm. To show that such an $a$ exists, it suffices to show that the maximum flow in $D^{(h,\lambda)}$ under the capacities $\hat{U}$ is 0 by the end of our algorithm.

We do so now. Our strategy will be to show that we have implicitly computed a flow on $D^{(h,\lambda)}$ of near-optimal value and so after just a few iterations it must be the case that the optimal flow on $D^{(h,\lambda)}$ is reduced to 0.

Consider a fixed iteration of our algorithm and let $\mathrm{OPT}^{(h,\lambda)}$ be the value of the maximum $V'(S)$-$V'(T)$ flow on $D^{(h,\lambda)}$. Since $f'$ is a blocking flow in $D^{(h,\lambda)}$ and $D^{(h,\lambda)}$ is an $h$-layer DAG by Lemma 10.2, it follows from Lemma 9.4 that

$$\mathrm{val}(f') \geq \frac{1}{h} \cdot \mathrm{OPT}^{(h,\lambda)}. \tag{4}$$

Continuing, we claim that $\tilde{f}$ is an $O(\frac{h^2}{\varepsilon})$-congested flow. In particular, any arc $a$ with capacity in $D$ greater than $O(\frac{h^2}{\varepsilon})$ is such that the sum of its capacities across copies in $D^{(h,\lambda)}$ is at most $\hat{U}_a$. Thus, such an arc is never overcongested by $\tilde{f}$. Any arc with capacity less than $O(\frac{h^2}{\varepsilon})$ in $D'$ has up to $O(\frac{h^2}{\varepsilon})$ copies in $D^{(h,\lambda)}$ each of which has capacity 1; thus, such an arc may have flow value up to $O(\frac{h^2}{\varepsilon})$ in $\tilde{f}$. Thus, by $\text{val}(\tilde{f}) = \text{val}(f')$ and this bound on the congestedness of $\tilde{f}$, we have from Lemma 10.3 that

$$\begin{aligned}\text{val}(\hat{f}) &\geq \frac{\varepsilon^2}{h^6} \cdot \text{val}(\tilde{f}) \\ &= \frac{\varepsilon^2}{h^6} \cdot \text{val}(f'). \end{aligned} \tag{5}$$

Combining Equation (4) and Equation (5), we get

$$\text{val}(\hat{f}) \geq \frac{\varepsilon^2}{h^7} \cdot \text{OPT}^{(h,\lambda)}. \tag{6}$$

Lastly, let $f''$ be $\hat{f}$ projected back into $D^{(h,\lambda)}$. That is, if arc $a'$ is a copy of arc $a$ then $f''$ assigns to $a'$ the flow value $\sum_{P \ni a} \hat{f}_P$. Observe that by construction of $\hat{f}$, we know that $f''$ is a $V'(S)$-$V'(T)$ flow in $D^{(h,\lambda)}$ of value $\text{val}(f'') = \text{val}(\hat{f})$. Thus, applying this and Equation (6) we get

$$\text{val}(f'') \geq \frac{\varepsilon^2}{h^7} \cdot \text{OPT}^{(h,\lambda)}.$$

Since we decrement the value of $\hat{U}_a$ by $f''_a$ in each iteration, it follows that after $\tilde{O}(\frac{h^7}{\varepsilon^2})$ many repetitions of Algorithm 2, we must decrease the value of the optimal flow in $D^{(h,\lambda)}$ by at least a constant fraction since otherwise we would have computed a flow with value greater than that of the optimal flow. Since initially $\text{OPT}^{(h,\lambda)} \leq \text{poly}(n)$, we get that after $\tilde{O}(\frac{h^7}{\varepsilon^2})$-many repetitions we have reduced the value of the optimal flow to 0 on $D^{(h,\lambda)}$, therefore showing that $f$ satisfies the near-lightest path blocking property.

It remains to show our running times. The computation in each of our iterations is dominated by constructing the length-expanded digraph $D^{(h,\lambda)}$, computing our maximal integral flow $f^{(h)}$ in $D^{(h,\lambda)}$ and decongesting our flow.

- We can construct $D^{(h,\lambda)}$ by e.g. Bellman-Ford for $\tilde{O}(h)$ rounds for a total running time of $\tilde{O}(h)$ in either CONGEST or parallel. Likewise projecting flows back from $D^{(h,\lambda)}$ is trivial.

- It is is easy to simulate $D^{(h,\lambda)}$ in either CONGEST or in parallel with an overhead of $O(\frac{h^2}{\varepsilon})$ since this is a bound on the number copies of each vertex.

  With randomization, by Lemma 7.2 computing $f'$ takes time $\tilde{O}(h^3)$ in parallel with $m$ processors or $\tilde{O}(h^4)$ in CONGEST on $D^{(h,\lambda)}$ and so $\tilde{O}(\frac{h^5}{\varepsilon})$ in parallel or $\tilde{O}(\frac{h^6}{\varepsilon})$ in CONGEST on $D$.

  For our deterministic algorithm, by Lemma 9.1 doing so takes $\tilde{O}(h^3)$ in parallel with $m$ processors and CONGEST time $\tilde{O}(h^6 \cdot (\rho_{CC})^{10})$ on $D^{(h,\lambda)}$ and so $\tilde{O}(\frac{1}{\varepsilon} \cdot h^5)$ parallel time on $D$ or $\tilde{O}(\frac{1}{\varepsilon} \cdot h^8 \cdot (\rho_{CC})^{10})$ CONGEST time on $D$.

- Lastly, decongesting our flow by Lemma 10.3 and the fact that $\tilde{f}$ is $O(\frac{h^2}{\varepsilon})$-congested takes deterministic parallel time $\tilde{O}(\frac{h^5}{\varepsilon^2})$ and deterministic CONGEST time $\tilde{O}(\frac{h^9}{\varepsilon^3})$.

Combining these running times with our $\tilde{O}(\frac{h^7}{\varepsilon^2})$-many repetitions gives the stated running times. □

# 11 Computing Length-Constrained Flows and Moving Cuts

Having shown how to compute an $h$-length $(1+\epsilon)$-lightest path blocker, we now use a series of these as batches to which we apply multiplicative-weights-type updates. The result is our algorithm which returns both a length-constrained flow and a (nearly) certifying moving cut.

**Algorithm 3** Length-Constrained Flows and Moving Cuts

**Input:** digraph $D=(V,A)$ with lengths $\ell$, capacities $U$, $h \geq 1$, $S,T \subseteq V$ and $\varepsilon \in (0,1)$.
**Output:** $(1\pm\varepsilon)$-approximate $h$-length flow $f$ and moving cut $w$.
Let $\epsilon_0 = \frac{\epsilon}{6}$, let $\zeta = \frac{1+2\varepsilon_0}{\varepsilon_0} + 1$ and let $\eta = \frac{\varepsilon_0}{(1+\varepsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$.
Initialize $w_a \leftarrow \left(\frac{1}{m}\right)^{\zeta}$ for all $a \in A$.
Initialize $\lambda \leftarrow \left(\frac{1}{m}\right)^{\zeta}$.
Initialize $f_P \leftarrow 0$ for all $P \in \mathcal{P}_h(S,T)$.
**while** $\lambda < 1$ **do**:
  **for** $\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations: **do**
    Compute $h$-length $(1+\epsilon_0)$-lightest path blocker $\hat{f}$ (using Theorem 10.1 with current $\lambda$).
    **Length-Constrained Flow (Primal) Update:** $f \leftarrow f + \eta \cdot \hat{f}$.
    **Moving Cut (Dual) Update:** $w_a \leftarrow (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a$ for every $a \in A$.
  $\lambda \leftarrow (1+\varepsilon_0)\cdot\lambda$
**return** $(f,w)$.

As a reminder for an $h$-length flow $f$, we let $f(a) := \sum_{P \ni a} f_P$. Throughout our analysis we will refer to the innermost loop of Algorithm 3 as one "iteration." We begin by observing that $\lambda$ always lower bounds $d_w^{(h)}(S,T)$ in our algorithm.

**Lemma 11.1.** *At the beginning of each iteration of Algorithm 3 we have* $\lambda \leq d_w^{(h)}(S,T)$

*Proof.* Our proof is by induction. The statement trivially holds at the beginning of our algorithm.

Let $\lambda_i$ be the value of $\lambda$ at the beginning of the $i$th iteration. We argue that if $d_w^{(h)}(S,T) = \lambda_i$ then after $\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ additional iterations we must have $d_w^{(h)}(S,T) \geq (1+\varepsilon_0)\cdot\lambda_i$. Let $\lambda_i' = (1+\epsilon_0)\cdot\lambda$ be $\lambda$ after these iterations. Let $\hat{f}_j$ be our lightest path blocker in the $j$th iteration.

Assume for the sake of contradiction that $d_w^{(h)}(S,T) < \lambda_i'$ after $i + \Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations. It follows that there is some path $P \in \mathcal{P}_h(S,T)$ with weight at most $\lambda_i'$ after $i + \Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations. However, notice that by definition of an $h$-length $(1+\epsilon_0)$-lightest path blocker (Definition 10.1), we know that for every $j \in \left[i, i+\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)\right]$ there is some $a \in P$ for which $\hat{f}_j(a) = U_a$. By averaging, it follows that there is some single arc $a \in P$ for which $\hat{f}_j(a) = U_a$ for at least $\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ of these $j \in [i, i+\Theta\left(\frac{h\log_{1+\epsilon_0} n}{\epsilon_0}\right)]$. Since every such arc starts with dual value

$(\frac{1}{m})^\zeta$ and multiplicatively increases by a $(1+\epsilon_0)$ factor in each of these updates, such an arc after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations must have $w_a$ value at least $(\frac{1}{m})^\zeta \cdot (1+\epsilon_0)^{\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)} \geq n^2$ for an appropriately large hidden constant in our $\Theta$. However, by assumption, the weight of $P$ is at most $\lambda_i'$ after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations and this is at most 2 since $\lambda_i < 1$ since otherwise our algorithm would have halted. But $2 < n^2$ and so we have arrived at a contradiction.

Repeatedly applying the fact that $\lambda_i' = (1+\epsilon_0)\lambda_i$ gives that $\lambda$ is always a lower bound on $d_w^{(h)}(S,T)$. □

We next prove the feasibility of our solution.

**Lemma 11.2.** *The pair $(f, w)$ returned by Algorithm 3 are feasible for Length-Constrained Flow LP and Moving Cut LP respectively.*

*Proof.* First, observe that by Lemma 11.1 we know that $\lambda$ is always a lower bound on $d_w^{(h)}(S,T)$ and so since we only return once $\lambda > 1$, the $w$ we return is always feasible.

To see that $f$ is feasible it suffices to argue that for each arc $a$, the number of times a path containing $a$ has its primal value increased is at most $\frac{U_a}{\eta}$. Notice that each time we increase the primal value on a path containing arc $a$ by $\eta$ we increase the dual value of this edge by a multiplicative $(1+\epsilon_0)^{1/U_a}$. Since the weight of our arcs according to $w$ start at $(\frac{1}{m})^\zeta$, it follows that if we increase the primal value of $k$ paths incident to arc $a$ then $w_a = (1+\epsilon_0)^{k/U_a} \cdot (\frac{1}{m})^\zeta$. On the other hand, by assumption when we increase the dual value of an arc $a$ it must be the case that $w_a < 1$ since otherwise $d_w^{(h)}(S,T) \geq 1$, contradicting the fact that $\lambda$ always lower bounds $d_w^{(h)}(S,T)$. It follows that $(1+\epsilon_0)^{k/U_a} \cdot (\frac{1}{m})^\zeta \leq 1$ and so applying the fact that $\ln(1+\epsilon_0) \geq \frac{\epsilon_0}{1+\epsilon_0}$ for $\epsilon_0 > -1$ and our definition of $\zeta$ and $\eta$ we get

$$\begin{aligned} k &\leq \frac{\zeta \cdot (1+\varepsilon_0)}{\varepsilon_0} \cdot U_a \log m \\ &= \frac{U_a}{\eta} \end{aligned}$$

as desired. □

We next prove the near-optimality of our solution.

**Lemma 11.3.** *The pair $(f, w)$ returned by Algorithm 3 satisfies $(1-\epsilon)\sum_a w_a \leq \sum_P f_P$.*

*Proof.* Fix an iteration $i$ of the above while loop and let $\hat{f}$ be our lightest path blocker in this iteration. Let $k_i$ be $\text{val}(\hat{f})$, let $\lambda_i$ be $\lambda$ at the start of this iteration and let $D_i := \sum_a w_a$ be our total dual value at the start of this iteration. Notice that $\frac{1}{\lambda_i} \cdot w$ is dual feasible and has cost $\frac{D_i}{\lambda_i}$ by Lemma 11.1. If $\beta$ is the optimal dual value then by optimality it follows that $\beta \leq \frac{D_i}{\lambda_i}$, giving us the upper bound on $\lambda_i$ of $\frac{D_i}{\beta}$. By how we update our dual, our bound on $\lambda_i$ and $(1+x)^r \leq 1 + xr$ for any $x \geq 0$ and $r \in (0,1)$ we have that

$$\begin{aligned} D_{i+1} &= \sum_a (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a \cdot U_a \\ &\leq \sum_a \left(1 + \frac{\epsilon_0 \hat{f}(a)}{U_a}\right) \cdot w_a \cdot U_a \end{aligned}$$

$$
\begin{aligned}
&= D_i + \epsilon_0 \sum_a \hat{f}(a) w_a \\
&\leq D_i + \epsilon_0 (1 + 2\varepsilon_0) \cdot k_i \lambda_i \\
&\leq D_i \left(1 + \frac{(1+2\varepsilon_0)\varepsilon_0 \cdot k_i}{\beta}\right) \\
&\leq D_i \cdot \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot k_i}{\beta}\right).
\end{aligned}
$$

Let $T-1$ be the index of the last iteration of our algorithm; notice that $D_T$ is the value of $w$ in our returned solution. Let $K := \sum_i k_i$. Then, repeatedly applying this recurrence gives us

$$
\begin{aligned}
D_T &\leq D_0 \cdot \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot K}{\beta}\right) \\
&= \left(\frac{1}{m}\right)^{\zeta - 1} \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot K}{\beta}\right)
\end{aligned}
$$

On the other hand, we know that $w$ is dual feasible when we return it, so it must be the case that $D_T \geq 1$. Combining this with the above upper bound on $D_T$ gives us $1 \leq \left(\frac{1}{m}\right)^{\zeta} \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot K}{\beta}\right)$. Solving for $K$ and using our definition of $\zeta$ gives us

$$
\begin{aligned}
\beta \log m \cdot \frac{\zeta - 1}{(1+2\varepsilon_0)\cdot \varepsilon_0} &\leq K \\
\beta \log m \cdot \frac{1}{\varepsilon_0^2} &\leq K.
\end{aligned}
$$

However, notice that $K\eta$ is the primal value of our solution so using our choice of $\eta$ and rewriting this inequality in terms of $K\eta$ by multiplying by $\eta = \frac{\varepsilon_0}{(1+\varepsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$ and applying our definition of $\zeta = \frac{1+2\varepsilon_0}{\varepsilon_0} + 1$ gives us

$$
\begin{aligned}
\frac{\beta}{\varepsilon_0 \cdot (1+\varepsilon_0) \cdot \zeta} &\leq K\eta \\
\frac{\beta}{(1+\varepsilon_0)(1+3\varepsilon_0)} &\leq K\eta. \qquad (7)
\end{aligned}
$$

Moreover, by our choice of $\varepsilon_0 = \frac{\varepsilon}{6}$ and the fact that $\frac{1}{1+x+x^2} \geq 1 - x$ for $x \in (0,1)$ we get

$$
\begin{aligned}
1 - \varepsilon &\leq \frac{1}{1+\varepsilon+\varepsilon^2} \\
&\leq \frac{1}{(1+\frac{1}{2}\varepsilon)^2} \\
&\leq \frac{1}{(1+3\varepsilon_0)^2} \\
&\leq \frac{1}{(1+\varepsilon_0)(1+3\varepsilon_0)}. \qquad (8)
\end{aligned}
$$

Combining Equation (7) and Equation (8) we conclude that

$$
(1-\varepsilon)\cdot\beta \leq K\eta.
$$

□

We conclude with our main theorem by proving that we need only iterate our algorithm $\tilde{O}\left(\frac{h}{\epsilon^4}\right)$ times.

**Theorem 3.1.** *Given a digraph $D = (V, A)$ with capacities $U$, lengths $\ell$, length constraint $h \geq 1$, $\varepsilon > 0$ and source and sink vertices $S, T \subseteq V$, one can compute a feasible $h$-length flow, moving cut pair $(f, w)$ that is $(1 \pm \epsilon)$-approximate in:*

1. *Deterministic parallel time $\tilde{O}(\frac{1}{\varepsilon^9} \cdot h^{17})$ with $m$ processors*
2. *Randomized CONGEST time $\tilde{O}(\frac{1}{\varepsilon^9} \cdot h^{17})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\frac{1}{\varepsilon^9} \cdot h^{17} + \frac{1}{\varepsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.*

*Also, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$ and each $f_j$ is an integral $h$-length $S$-$T$ flow.*

*Proof.* We use Algorithm 3. By Lemma 11.2 and Lemma 11.3 we know that our solution is feasible and $(1 \pm \epsilon)$-optimal so it only remains to argue the runtime of our algorithm and that the returned flow decomposes in the stated way.

We argue that we must only run for $O\left(\frac{h \log^2 n}{\epsilon^4}\right)$ total iterations. Since $\lambda$ increases by a multiplicative $(1 + \epsilon_0)$ after every $\Theta\left(\frac{h \log n}{\epsilon_0^2}\right)$ iterations and starts at at least $\left(\frac{1}{m}\right)^{\Theta(1/\varepsilon_0)}$, it follows by Lemma 11.1 that after $y \cdot \Theta\left(\frac{h \log n}{\epsilon_0^2}\right)$ total iterations the $h$-length distance between $S$ and $T$ is at least $(1 + \epsilon_0)^y \cdot \left(\frac{1}{m}\right)^{\Theta(1/\varepsilon_0)}$. Thus, for $y \geq \Omega\left(\frac{\log_{1+\epsilon_0} m}{\epsilon_0}\right) = \Omega\left(\frac{\log n}{\epsilon_0^2}\right)$ we have that $S$ and $T$ are at least 1 apart in $h$-length distance. Consequently, our algorithm must run for at most $O\left(\frac{h \log^2 n}{\epsilon_0^4}\right) = O\left(\frac{h \log^2 n}{\epsilon^4}\right)$ many iterations.

Our running time is immediate from the the bound of $O\left(\frac{h \log^2 n}{\epsilon^4}\right)$ on the number of iterations of the while loop and the running times given in Theorem 10.1 for computing our $h$-length $(1+\epsilon_0)$-lightest path blocker.

Lastly, the flow decomposes in the stated way because we have at most $O\left(\frac{h \log^2 n}{\epsilon^4}\right)$ iterations and each $f_j$ is an integral $S$-$T$ flow by Theorem 10.1. Thus, our final solution is $\eta \cdot \sum_{j=1}^{k} f_j$ and $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$. □

# 12 Application: Maximal and Maximum Disjoint Paths

In this section we show that our main theorem (Theorem 3.1) almost immediately gives deterministic CONGEST algorithms for many varieties of maximal disjoint path problems as well as essentially-optimal algorithms for many maximum disjoint path problems. In Section 12.1 we give the variants we study. In Section 12.2 we observe that it suffices to solve the arc-disjoint directed variants of these problems. Lastly, we give our results for maximal and maximum disjoint path problems in Section 12.3 and Section 12.4 respectively where we observe in Section 12.5 that our algorithms for the latter are essentially optimal.

## 12.1 Maximal and Maximum Disjoint Path Variants

We consider the following maximal disjoint path variants.

**Maximal Vertex-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of $h$-length vertex-disjoint $S$ to $T$ paths $\mathcal{P}$ such that any $h$-length $S$ to $T$ path shares a vertex with at least one path in $\mathcal{P}$.

**Maximal Edge-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of $h$-length edge-disjoint $S$ to $T$ paths $\mathcal{P}$ such that any $h$-length $S$ to $T$ path shares an edge with at least one path in $\mathcal{P}$.

**Maximal Vertex-Disjoint Directed Paths**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of $h$-length vertex-disjoint $S$ to $T$ paths $\mathcal{P}$ such that any $h$-length $S$ to $T$ path shares a vertex with at least one path in $\mathcal{P}$.

**Maximal Arc-Disjoint Directed Paths**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and two disjoint sets $S, T \subseteq V$, find a collection of $h$-length arc-disjoint $S$ to $T$ paths $\mathcal{P}$ such that any $h$-length $S$ to $T$ path shares an arc with at least one path in $\mathcal{P}$.

As discussed in Section 1.1, the existence of efficient deterministic algorithms for the above problems (specifically the maximal vertex-disjoint paths problem) in CONGEST was stated as an open question by Chang and Saranurak [12] and the lack of these algorithms is a major barrier to simple deterministic constructions of expander decompositions.

We consider the following maximum disjoint path variants.

**Maximum Vertex-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of $h$-length vertex-disjoint $S$ to $T$ paths.

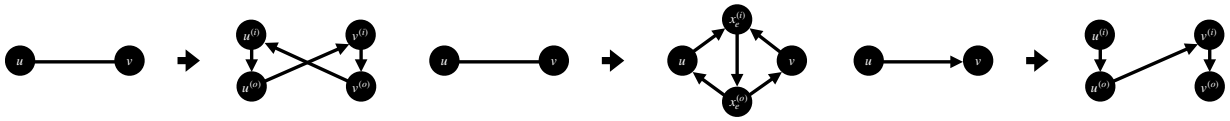
**Maximum Edge-Disjoint Paths**: Given graph $G = (V, E)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of $h$-length edge-disjoint $S$ to $T$ paths.

**Maximum Vertex-Disjoint Directed Paths**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of $h$-length vertex-disjoint $S$ to $T$ paths.

**Maximum Arc-Disjoint Directed Paths**: Given digraph $D = (V, A)$, length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$, find a max cardinality collection of $h$-length arc-disjoint $S$ to $T$ paths.

### 12.2 Reducing Among Variants

We begin by observing that the arc-disjoint directed paths problem is the hardest of the above variants and so it will suffice to solve this problem. The reductions we use are illustrated in Figure 7.




(a) Vertex-disjoint paths. (b) Edge-disjoint paths. (c) Vertex-disjoint directed paths.

Figure 7: Illustration of our reduction on a single edge or arc between $u$ and $v$ for reducing maximal or maximum vertex-disjoint paths, edge-disjoint paths or vertex-disjoint directed paths to arc-disjoint directed paths.

**Lemma 12.1.** *If there is a deterministic algorithm for maximal arc-disjoint directed paths in CONGEST running in time $T$ then there are deterministic CONGEST algorithms for maximal vertex-disjoint paths, edge-disjoint paths and vertex-disjoint directed paths all running in time $O(T)$.*

*Likewise, if there is a deterministic (resp. randomized) parallel with $m$ processors or CONGEST algorithm for maximum arc-disjoint directed paths in CONGEST running in time $T$ with approximation ratio $\tilde{O}(h)$ then there are deterministic (resp. randomized) parallel with $m$ processors and CONGEST algorithms for maximum vertex-disjoint paths, edge-disjoint paths and vertex-disjoint directed paths all running in time $O(T)$ with approximation ratio $\tilde{O}(h)$.*

*Proof.* We reduce each of maximal vertex-disjoint paths, maximal edge-disjoint paths and maximal vertex-disjoint directed paths to maximal arc-disjoint directed paths and do the same for the maximum variants of these problems.

**Reducing from maximal/maximum vertex-disjoint paths.** Consider an instance of maximal or maximum vertex-disjoint paths on graph $G = (V, E)$ with length constraint $h$ and vertex sets $S$ and $T$. We create a digraph $D = (V', A)$ as follows:

- **Vertices:** $V'$ is constructed as follows: for each $v \in V$ we add to $V'$ vertex $v^{(i)}$ and $v^{(o)}$.
- **Arcs:** For each $v \in V$ we add an arc from $v^{(i)}$ to $v^{(o)}$. Furthermore, for each $e = \{u, v\} \in E$ we add to $A$ the arcs $(u^{(o)}, v^{(i)})$ and $(v^{(o)}, u^{(i)})$.

A collection of arc-disjoint paths in $D$ from $S' = \{s^{(i)} : s \in S\}$ to $T' = \{t^{(o)} : t \in T\}$ with length constraint $2h - 1$ uniquely corresponds to an equal cardinality collection of $S$-$T$ vertex-disjoint paths in $G$ with length constraint $h$. Thus, an $\tilde{O}(h)$ approximation on $D$ for the maximum $S'$-$T'$ arc-disjoint directed paths problem gives an $\tilde{O}(h)$ approximation for the maximum vertex-disjoint paths problem on $G$. Likewise, a maximal collection of arc-disjoint $S'$-$T'$ paths on $D$ with length constraint $2h - 1$ corresponds to a maximal collection of vertex-disjoint $S$-$T$ paths with length constraint $h$. Lastly, a $T$-time CONGEST algorithm on $D$ can be simulated on $G$ in time $O(T)$ since each $v \in V$ can simulate $v^{(o)}$ and $v^{(i)}$.

**Reducing from maximal/maximum edge-disjoint paths.** Consider an instance of maximal or maximum edge-disjoint paths on graph $G = (V, E)$ with length constraint $h$ and vertex sets $S$ and $T$. We create a digraph $D = (V', A)$ as follows:

- **Vertices:** $V'$ consists of $V$ along with two vertices for each edge $e$, namely $x_e^{(i)}$ and $v_e^{(o)}$ for each $e \in E$.

- **Arcs:** For each $e \in \{u, v\} \in E$ we add to $A$ an arc from $x_e^{(i)}$ to $x_e^{(o)}$ as well as an arc from $u$ and $v$ to $x_e^{(i)}$ and an arc from $x_e^{(o)}$ to $u$ and $v$.

A collection of arc-disjoint $S$-$T$ paths in $D$ with length constraint $3h$ uniquely corresponds to an equal cardinality collection of $S$-$T$ edge-disjoint paths in $G$ with length constraint $h$. Thus, an $\tilde{O}(h)$ approximation on $D$ for the maximum $S$-$T$ arc-disjoint directed paths problem gives an $\tilde{O}(h)$ approximation for the maximum edge-disjoint paths problem on $G$. Likewise, a maximal collection of arc-disjoint $S$-$T$ paths on $D$ with length constraint $3h$ corresponds to a maximal collection of edge-disjoint $S$-$T$ paths with length constraint $h$ on $G$. Lastly, a $T$-time CONGEST algorithm on $D$ can be simulated on $G$ in time $O(T)$ since the endpoints of $e \in E$ can simulate $x_e^{(i)}$ and $x_e^{(o)}$ with constant overhead.

**Reducing from maximal/maximum vertex-disjoint directed paths.** Consider an instance of maximal or maximum vertex-disjoint directed paths on graph $D = (V, A)$ with length constraint $h$ and vertex sets $S$ and $T$. We create a digraph $D' = (V', A')$ as follows:

- **Vertices:** $V'$ consists of vertices $v^{(o)}$ and $v^{(i)}$ for each $v \in V$.
- **Arcs:** For each $v \in V$ we add to $A'$ the arc $(v^{(i)}, v^{(o)})$. For each arc $a = (u, v) \in A$ we add to $A'$ the arc $(u^{(o)}, v^{(i)})$.

A collection of arc-disjoint paths in $D'$ from $S' = \{s^{(i)} : s \in S\}$ to $T' = \{t^{(o)} : t \in T\}$ with length constraint $2h - 1$ uniquely corresponds to an equal cardinality collection of $S$-$T$ vertex-disjoint paths in $D$ with length constraint $h$. Thus, an $\tilde{O}(h)$ approximation on $D'$ for the maximum $S'$-$T'$ arc-disjoint directed paths problem gives an $\tilde{O}(h)$ approximation for the maximum $S$-$T$ vertex-disjoint directed paths problem on $D$. Likewise, a maximal collection of arc-disjoint $S'$-$T'$ paths on $D'$ with length constraint $2h - 1$ corresponds to a maximal collection of vertex-disjoint $S$-$T$ paths with length constraint $h$ on $D$. Lastly, a $T$-time CONGEST algorithm on $D'$ can be simulated on $D$ in time $T$ each $v \in V$ can simulate $v^{(i)}$ and $v^{(o)}$. □

## 12.3 Maximal Disjoint Path Algorithms

We now observe that our length-constrained flow algorithms allow us to solve maximal arc-disjoint directed paths and therefore all of the above variants efficiently.

**Theorem 12.2.** *There are deterministic CONGEST algorithms for maximal vertex-disjoint paths, edge-disjoint paths, vertex-disjoint directed paths and arc-disjoint directed paths running in time* $\tilde{O}\left(h^{18} + h^{17} \cdot (\rho_{CC})^{10}\right)$.

*Proof.* By Lemma 12.1, it suffices to show that maximal arc-disjoint directed paths can be solved in time $\tilde{O}\left(h^{18} + h^{17} \cdot (\rho_{CC})^{10}\right)$. We proceed to do so on digraph $D$ with length constraint $h$ and vertex sets $S$ and $T$ for the rest of this proof.

Specifically, we repeat the following until no path between $S$ and $T$ consists of $h$ or fewer edges. Apply Theorem 3.1 to compute a $(1 - \epsilon)$-approximate $h$-length $S$-$T$ flow $f$ in $D$ for $\epsilon = .5$ (any constant would suffice) with unit capacities. By the properties of $f$ as guaranteed by Theorem 3.1, we have that $f = \eta \cdot \sum_{j=1}^{k} f_j$ for $\eta = \tilde{\Theta}(1)$ and $k = \tilde{O}(h)$ where each $f_j$ is an integral flow. For each vertex $v$ we let $f_j^{(v)}$ be $f_j$ restricted to its flow paths out of $v$ and let $f_{j^*}^{(v)} := \arg\max_{f_j^{(v)}} \text{val}(f_j^{(v)})$.

Then, we let $f_{j^*} := \sum_v f_{j^*}^{(v)}$ (notice that we cannot simply define $f_{j^*}$ as $\arg\max_{f_j} \text{val}(f_j)$ since we cannot compute $\text{val}(f_j)$ effiicently in CONGEST because $D$ may have diameter much larger than $h$). Observe that since $f_{j^*}$ is integral and $h$-length, it exactly corresponds to an arc-disjoint collection of $S$-$T$ paths $\mathcal{P}'$ in $D$ each of which consists of at most $h$ edges. We add $\mathcal{P}'$ to $\mathcal{P}$, delete from $D$ any arc incident to a path of $\mathcal{P}'$ and continue to the next iteration.

As the above algorithm removes at least one path from $S$ to $T$ each time, it clearly terminates with a feasible solution for the maximal arc-disjoint directed paths problem.

Stronger, though, we claim that we need only iterate the above $\tilde{O}(h)$-many times until $S$ and $T$ are disconnected. Specifically, fix one iteration and let $\mathcal{P}^*$ be the collection of vertex-disjoint paths from $S$ to $T$ of maximum cardinality at the beginning of this iteration. By the $(1-\epsilon)$-optimality of our flow and an averaging argument we have that $\text{val}(f_{j^*}) \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot |\mathcal{P}^*|$ which is to say that $|\mathcal{P}'| \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot |\mathcal{P}^*|$. However, it follows that after $\tilde{\Theta}(h)$-many iterations for a large hidden constant we must at least halve $|P^*|$ since otherwise we would have computed a collection of vertex-disjoint $S$-$T$ paths whose cardinality is larger than the largest cardinality of any set of vertex-disjoint $S$-$T$ paths. Since initially $|P^*| \leq n$, it follows that after iterating the above $\tilde{O}(h)$-many times we have reduced $|P^*|$ to 0 which is to say we have solved the maximal arc-disjoint directed paths problem.

Our running time is immediate from Theorem 3.1 and the above bound we provide on the number of required iterations of $\tilde{O}(h)$ as well as the fact that each vertex can easily compute $f_{j^*}^{(v)}$ and $\mathcal{P}$ deterministically in parallel or CONGEST time $\tilde{O}(h)$ since our flows are $h$-length. □

Applying the fact that it is known that $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$ (see Section 5.4), the above gives deterministic CONGEST algorithms running in time $\tilde{O}(\text{poly}(h) \cdot 2^{O(\sqrt{\log n})})$. If $\rho_{CC}$ where improved to be poly-log in $n$ then we would get a $\tilde{O}(\text{poly}(h))$ running time.

### 12.4 Maximum Disjoint Path Algorithms

Lastly, we observe that our length-constrained flow algorithms allow us to $\tilde{O}(h)$-approximate maximum arc-disjoint directed paths and therefore all of the above variants efficiently.

**Theorem 12.3.** *There are $\tilde{O}(h)$-approximation algorithms for maximum vertex-disjoint paths, edge-disjoint paths, vertex-disjoint directed paths and arc-disjoint directed paths running in:*

- *Deterministic parallel time $\tilde{O}(h^{17})$ with $m$ processors;*
- *Randomized CONGEST time $\tilde{O}(h^{17})$ with high probability;*
- *Deterministic CONGEST time $\tilde{O}\left(h^{17} + h^{16} \cdot (\rho_{CC})^{10}\right)$.*

*Proof.* By Lemma 12.1, it suffices to provide a $\tilde{O}(h)$-approximate algorithm for maximum arc-disjoint directed paths with the stated running times. We do so for the rest of this proof. Let the input be digraph $D = (V, A)$ with length constraint $h \geq 1$ and disjoint sets $S, T \subseteq V$.

We apply Theorem 3.1 to compute an $\epsilon$-approximate $h$-length constrained flow $f$ in $D$ for $\epsilon = .5$ (any constant would suffice) and capacities $U_a = 1$ for every $a$. By the properties of $f$ as guaranteed by Theorem 3.1, we have that $f = \eta \cdot \sum_{j=1}^{k} f_j$ for $\eta = \Theta(1)$ and $k = \tilde{O}(h)$ where each $f_j$ is an integral flow. For each vertex $v$ we let $f_j^{(v)}$ be $f_j$ restricted to its flow paths out of $v$ and let $f_{j^*}^{(v)} := \arg\max_{f_j^{(v)}} \text{val}(f_j^{(v)})$. Then, we let $f_{j^*} := \sum_v f_{j^*}^{(v)}$. Observe that since $f_{j^*}$ is integral and

$h$-length, it exactly corresponds to an arc-disjoint collection of paths $\mathcal{P}$ in $D$ each of which consists of at most $h$ edges. We return $\mathcal{P}$ as our solution.

Letting $\mathcal{P}^*$ be the optimal solution to the input problem we have by $k = \tilde{O}(h)$ and an averaging argument that

$$|\mathcal{P}| = \text{val}(f_{j^*}) \geq \tilde{\Omega}\left(\frac{1}{h}\right) \cdot |\mathcal{P}^*|$$

and so our solution is $\tilde{\Omega}(\frac{1}{h})$-approximate.

For our running time, observe that each vertex can easily compute $f_{j^*}^{(v)}$ and $\mathcal{P}$ deterministically in parallel or CONGEST time $\tilde{O}(h)$ since our flows are $h$-length. Thus, our running time is dominated by Theorem 3.1. □

### 12.5 On the Hardness of Maximum Disjoint Paths

Guruswami et al. [19] give hardness results for a variety of length-constrained maximum disjoint path problems. In their work they state hardness of approximation result in terms of $m$, the number of edges in the graph. In the following we restate these results but in terms of $h$, the length-constraint.

**Theorem 12.4** (Adaptation of Theorem 1 of Guruswami et al. [19])**.** *Assume the strong exponential time hypothesis (SETH). Then there does not exist a polynomial-time $O(h)$-approximation algorithm solving the maximum arc-disjoint directed paths problem for instances where $h = \Omega(\log n)$.*

Observe that it follows that assuming SETH, the parallel algorithm in Theorem 12.3 is optimal up to poly-logs.

## 13 Application: Simple Distributed Expander Decompositions

In this section, we explain how our maximal disjoint path algorithm can significantly simplify the distributed deterministic expander decomposition of Chang and Saranurak [12].

The key algorithmic primitive of [12] in their distributed deterministic expander decomposition is their Lemma D.8. Instead of computing maximal bounded-hop disjoint paths, they were only be able to compute a set of paths that are "nearly maximal". The formal statement is as follows:

**Lemma 13.1** (Nearly maximal disjoint paths (Lemma D.8 of [12])**.** *Consider a graph $G = (V, E)$ of maximum degree $\Delta$. Let $S \subseteq V$ and $T \subseteq V$ be two subsets. There is an $O(d^3\beta^{-1}\log^2\Delta\log n)$-round deterministic algorithm that finds a set $P$ of $S-T$ vertex-disjoint paths of length at most $d$, together with a vertex set $B$ of size at most $\beta|V \setminus T| < \beta|V|$, such that any $S-T$ path of length at most $d$ that is vertex-disjoint to all paths in $P$ must contain a vertex in $B$.*

The set $P$ from the lemma is nearly maximal in the sense that if $B$ is deleted from $G$, then $P$ would be maximal. However, we can see that there might possibly be many additional disjoint paths that go through $B$. This set $B$ complicates all of their later algorithmic steps.

The high-level summary of the issue is that all their flow primitives that are based on Lemma D.8 must work with source/sink sets that are very big only. Otherwise, the guarantee becomes meaningless or the running time becomes very slow.

Now, we explain in more details. Given two sets $S$ and $T$ where $|S| \leq |T|$, normally if the matching player from the cut-matching game does not return a sparse cut, then it returns an embedding of a matching where every vertex in $S$ is matched to some vertex in $T$. However, in Lemma D.9 of [12], the matching player based on Lemma D.8 may return an embedding that leaves as many as $\approx \beta|V \setminus T|$ vertices in $S$ unmatched. This is called the "left-over" set. We think of $\beta \geq 1/n^{o(1)}$ as the round complexity of Lemma D.8 is proportional to $\beta^{-1}$. Therefore, it is only when $|S|, |T| \geq 2\beta|V| \geq |V|/n^{o(1)}$ that Lemma D.9 in [12] may give some meaningful guarantee, yet this is still weaker than normal.

The same issue holds for their multi-commodity version of the matching player (i.e. Lemma D.11 of [12]). For the same reasoning, the lemma is meaningful only when the total number of source and sink is at least $\Omega(\beta|V|)$. The issue propagates to their important subroutine (Theorem 4.1 of [12]) for computing most balanced sparse cut. The guarantee holds when only the returned cut $C$ is such that $|C| \geq \Omega(\beta|V|)$. At the end, they managed to obtain an deterministic expander decomposition (just treat the edges incident to the left-over part as inter-cluster edges at the end). However, they need to keep track of this left-over parameter from the first basic primitive until the end result.

In contrast, in their randomized algorithm for computing expander decomposition, this issues does not appear anyway because of the randomized maximal disjoint path algorithm. Therefore, by plugging in our deterministic maximal disjoint path algorithm into the expander decomposition of [12], all these issue will be resolved immediately.

# 14 Application: $(1-\epsilon)$-Approximate Distributed Bipartite $b$-Matching

In this section we give the first efficient $(1-\epsilon)$-approximate CONGEST algorithms for maximum cardinality bipartite $b$-matching. In fact, our results are for the slightly more general edge-capacitated maximum bipartite $b$-matching problem, defined as follow.

> **Edge-Capacitated Maximum Bipartite $b$-Matching**: Given bipartite graph $G = (V, E)$, edge capacities $U$ and function $b : V \to \mathbb{Z}_{>0}$ compute an integer $x_e \in [0, U_e]$ for each $e \in E$ maximizing $\sum_e x_e$ so that for each $v \in V$ we have $\sum_{e \in \delta(v)} x_e \leq b(v)$.

Notice that the case where $b(v) = 1$ for every $v$ is just the classic maximum cardinality matching problem. "$b$-matching" seems to refer to two different problems in the literature depending on whether edges can be chosen with multiplicity: either it is the above problem where $U_e = 1$ for every $e \in E$ or it is the above problem where $U_e = \max_v b_v$ for each $e \in E$. Our algorithms will work for both of these variants since they solve the above problem which generalizes both of these problems.

The following theorem summarizes our main result for bipartite $b$-matching in CONGEST. Again, recall that $\rho_{CC}$ is defined in Definition 5.8 and is known to be at most $2^{O(\sqrt{\log n})}$.

**Theorem 14.1.** *There is a deterministic $(1-\epsilon)$-approximation for edge-capacitated maximum bipartite $b$-matching running in CONGEST time $\tilde{O}\left(\frac{1}{\varepsilon^9} + \frac{1}{\varepsilon^7} \cdot (\rho_{CC})^{10}\right)$.*

*Proof.* Our algorithm works in two steps. First, we reduce edge-capacitated $b$-matching to length-constrained flow and use our length constrained flow algorithm to efficiently compute a fractional flow. Then, we apply the flow rounding technology we developed in Section 9.2 to round this flow to an integral flow which, in turn, corresponds to an integral $b$-matching.

More formally our algorithm is as follows. Suppose we are given an instance of edge-capacitated $b$-matching on bipartite graph $G = (V, E)$. Let $L$ and $R$ be the corresponding bipartition of vertices of $G$. We construct the following instance of length-constrained flow on digraph $D = (V', A)$ with $h = 3$ as follows. Each $v \in V$ has two copies $v^{(i)}$ and $v^{(o)}$ in $V'$. We add arc $(v^{(i)}, v^{(o)})$ to $A$ with capacity $b(v)$. If $\{u, v\} \in E$ where $u \in L$ and $v \in R$ then we add arc $(u^{(o)}, v^{(i)})$ with capacity $U_e$ to $A$. Lastly, we let $S = \{u^{(i)} : u \in L\}$, $T = \{v^{(o)} : v \in R\}$ and the length of each arc in $D$ be 1. Next, we apply Theorem 3.1 to compute a $(1 - \varepsilon_1)$-approximate maximum 3-length $S$-$T$ flow $f$ on $D$ for some small $\varepsilon_1$ to be chosen later. Since $D$ is a 3-layer $S$-$T$ DAG we may interpret this as a (non-length-constrained) flow where the flow value on arc $a$ is $f(a)$.

We then apply Lemma 9.6 to this non-length-constrained flow to get integral $S$-$T$ flow $f'$ satisfying $\text{val}(f') \geq (1 - \varepsilon_2) \cdot \text{val}(f)$ for some small $\varepsilon_2$ to be chosen later. We return as our solution the $b$-matching which naturally corresponds to $f'$. Namely, if $e = \{u, v\}$ then since $f'$ is integral it assigns arc $(u^{(o)}, v^{(i)})$ a value in $\{0, 1, \ldots, U_e\}$. We let $x_e$ be this value for $e = \{u, v\}$ and we return as our $b$-matching solution $\{x_e\}_e$.

$f'$ is a $(1 - \varepsilon_1)(1 - \varepsilon_2)$-approximate maximum $S$-$T$ flow. Letting OPT be the value of the optimal $b$-matching solution, it is easy to see that the maximum $S$-$T$ flow has value OPT and so the solution we return has value at least $(1 - \varepsilon_1)(1 - \varepsilon_2) \cdot \text{OPT}$. Letting $\varepsilon_1 = \varepsilon_2 = \Theta(\varepsilon)$ for an appropriately small hidden constant we get that $(1 - \varepsilon_1)(1 - \varepsilon_2) \cdot \text{OPT} \geq (1 - \varepsilon) \cdot \text{OPT}$.

Lastly, we argue our running time. Our running time is dominated by one call to Theorem 3.1 with $\varepsilon_1 = \Theta(\varepsilon)$ which takes $\tilde{O}\left(\frac{1}{\varepsilon^9} + \frac{1}{\varepsilon^7} \cdot (\rho_{CC})^{10}\right)$ and one call to Lemma 9.6 with $\varepsilon_2 = \Theta(\varepsilon)$ which takes $\tilde{O}(\frac{1}{\varepsilon^5} \cdot (\rho_{CC})^{10})$. Combining these running times gives the overall running time of our algorithm. □

# 15 Application: Length-Constrained Cutmatches

As it captures low-latency communication subject to bandwidth constraints, the problem of computing low-congestion $h$-length paths between two set of nodes $S$ and $T$ occurs often in network optimization.

In this section we give algorithms that either finds a low-congestion $h$-length collection of paths between two sets of nodes or, if this is not possible, finds as large of such a collection of paths as possible together with a moving cut that (approximately) certifies that there is no low-congestion way of extending the current collection of paths. Such a construction is called a length-constrained cutmatch.

A recent work [23] uses the algorithms we give for cutmatches to give the first efficient constructions of a length-constrained version of expander decompositions. These constructions were then used to give the first distributed CONGEST algorithms for many problems including MST, $(1 + \epsilon)$-min-cut and $(1 + \epsilon)$-lightest paths that are guaranteed to run in sub-linear rounds as long as such algorithms exist on the input network.

In what follows, for a vertex subset $W \subseteq V$ we let $U^+(W) = \sum_{v \in W} \sum_{a \in \delta^+(v)} U_a$ and $U^-(W) = \sum_{v \in W} \sum_{a \in \delta^-(v)} U_a$. We also let $\delta^{\pm}(S, T) := \bigcup_{v \in S} \delta^+(v) \cup \bigcup_{v \in T} \delta^-(T)$

**Definition 15.1** ($h$-Length Cutmatch)**.** *Given digraph $D = (V, A)$ with capacities $U$ and lengths $\ell$, an $h$-length $\phi$-sparse cutmatch of congestion $\gamma$ between two node sets $S, T \subseteq V$ with $U^+(S) \leq U^-(T)$ consists of:*

- *An integral $h$-length $S$-$T$ flow $f$ in $D$ with capacities $\{U_a\}_{a \in \delta^{\pm}(S,T)} \cup \{\gamma \cdot U_a\}_{a \notin \delta^{\pm}(S,T)}$;*

- *A moving cut $w$ of $S$ and $T$ in $D$ with capacities $\{U_a - f_a\}_{a\in\delta^{\pm}(S,T)} \cup \{U_a\}_{a\notin\delta^{\pm}(S,T)}$ of value $\sum_a w_a \leq \phi\left(U^+(S) - val(f)\right)$.*

We proceed to show how to efficiently compute a cutmatch using our previous algorithm. As a reminder $\rho_{CC}$ is defined in Section 5.4 and is known to be at most $2^{O(\sqrt{\log n})}$.

**Theorem 15.2.** *Suppose we are given a digraph $D = (V, A)$ with capacities $U$ and lengths $\ell$. There is an algorithm that, given two node sets $S, T \subseteq V$ with $U^+(S) \leq U^-(T)$ and two integer parameters $h \geq 1$ and $\phi \leq 1$, outputs an $h$-length $\phi$-sparse cutmatch of congestion $\gamma$ between $S$ and $T$, where $\gamma = \tilde{O}(\frac{1}{\phi})$. This algorithm runs in:*

1. *Deterministic parallel time $\tilde{O}(\gamma \cdot h^{18})$ with $m$ processors*
2. *Randomized CONGEST time $\tilde{O}(\gamma \cdot h^{18})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\gamma \cdot h^{18} + \gamma \cdot h^{17} \cdot (\rho_{CC})^{10}\right)$.*

*Proof.* We initialize the flow we return $f$ to be 0 on all arcs. We set our working capacities to be $\hat{U} = U$ initially. The algorithm runs for at most $O(h \cdot \gamma)$ iterations for a small hidden constant. In each iteration $i \in [1, O(\gamma)]$ we use Theorem 3.1 with $\varepsilon = .5$ (any constant would suffice) to find a length-constrained flow, moving cut pair, $(\hat{f}, \hat{w})$ where $\delta = \tilde{\Theta}(1)$, $k = \tilde{O}(h)$, $f = \delta \cdot \sum_{j=1}^{k} f_j$ and $f_j$ is an integral $h$-length flow from $S$ to $T$ using capacities $\hat{U}$. By averaging there must be some $f_j$ such that $\text{val}(f_j) \geq \text{val}(\hat{f})/k$. We let $\tilde{f}$ be this $f_j$.

If $\text{val}(\tilde{f}) > \Omega\left(\frac{\log n \cdot \hat{U}^+(S)}{h\gamma}\right)$ then we update our solution as $f = f + \tilde{f}$ and decrement $\hat{U}_a$ by $\tilde{f}(a)$ for every $a \in \delta^{\pm}(S, T)$. Otherwise, we return the pair $(f, \hat{w})$ as our solution.

In each iteration $i$ in which $\text{val}(\tilde{f}) > \Omega\left(\frac{\log n \cdot \hat{U}^+(S)}{h\gamma}\right)$, we have that $\hat{U}^+(S)$ decreases multiplicatively by at least a $1 - \frac{2\log n}{h\gamma}$ factor. Such a shrinking can happen at most $O(h \cdot \gamma)$ times until $\hat{U}^+(S)$ is reduced to 0. Thus, our algorithm requires at most $O(h \cdot \gamma)$ iterations until terminating. Furthermore, notice when we return a moving cut $\hat{w}$ we have

$$\begin{aligned}
\sum_a \hat{w}_a &\leq 2 \cdot \text{val}(\hat{f}) \\
&\leq h \cdot \text{val}(\tilde{f}) \\
&\leq \tilde{O}\left(\frac{\hat{U}^+(S)}{\gamma}\right) \\
&= \tilde{O}\left(\frac{U^+(S) - \text{val}(f)}{\gamma}\right)
\end{aligned}$$

as desired. Also, observe that $f$ is indeed an integral $S$-$T$ flow in $D$ with the stated capacities since we always have $\tilde{f}(a) \leq \hat{U}_a$.

The running time is exactly that of running at most $O(h \cdot \gamma)$ invocations of Theorem 3.1 with $\epsilon = .5$. □

# 16 Conclusion and Future Work

In this work we gave the first efficient randomized and deterministic algorithms for computing $(1-\epsilon)$-approximate length-constrained flows both in parallel and in the CONGEST model of distributed computation. We used these algorithms to give new results in maximal and maximum disjoint path problems, expander decompositions, bipartite $b$-matching and length-constrained cutmatches. We conclude with several open questions and directions for future work.

1. Our length-constrained flow algorithms have a dependence of poly($h$) which when plugged into the techniques of Haeupler et al. [23] give CONGEST algorithms for many distributed problems, e.g. MST, whose running time is poly(OPT) (up to sub-linear factors) where OPT is the optimal CONGEST running time for the input problem. It would be exciting to improve the dependence on $h$ of our algorithms to, say, $O(h)$ as this result when combined with those of Haeupler et al. [23] would give CONGEST algorithms running in time $O(\text{OPT})$ (up to sub-linear factors).

2. The running time of many of our algorithms depends on $\rho_{CC}$, the best quality of a CONGEST algorithm for cycle cover (as defined in Definition 5.8). It is known that $\rho_{CC} \leq 2^{O(\sqrt{\log n})}$ but it would be extremely interesting to show that $\rho_{CC} \leq \tilde{O}(1)$. Such an improvement would immediately improve the dependency on $n$ from $n^{o(1)}$ to $\tilde{O}(1)$ for our CONGEST algorithms for deterministic length-constrained flows, deterministic maximal and maximum disjoint paths, $(1-\epsilon)$-approximate $b$-matching and length-constrained cutmatches. Such a result does not seem to be known even for the randomized case.

3. Lastly, many classic problems can be efficiently solved by reducing to flows but, in particular, by reducing to length-constrained flows with a length-constraint $h = O(1)$. Indeed this is how we were able to give new algorithms for $b$-matching in this work. It would be interesting to understand which additional classic problems our length-constrained flow algorithms give new algorithms for in CONGEST.

# A Generalizing Our Results to Multi-Commodity

In this section we generalize our main result for computing length-constrained flows and moving cuts to the setting where have many source sink pairs and we are trying to maximize the total flow between corresponding pairs subject to congestion constraints.

## A.1 Multi-Commodity Definitions and Our Multi-Commodity Results

We now more formally define a multi-commodity length-constrained flow and moving cut. Suppose we are given a digraph $D = (V, A)$ with arc capacities $U$, lengths $\ell$ and $\kappa$ source set, sink set pairs $\{(S_i, T_i)\}_i$. Then, we have the following LP with a variable $f_P^{\{i\}}$ for each $i$ and path $P \in \mathcal{P}_h(S_i, T_i)$. We let $f^{\{i\}}$ gives the entire flow for commodity $i$.

$$\max \sum_i \sum_{P \in \mathcal{P}_h(S_i,T_i)} f_P^{\{i\}} \text{ s.t.} \qquad \text{(Multi Length-Constrained Flow LP)}$$

$$\sum_i \sum_{P \ni a} f_P^{\{i\}} \leq U_a \qquad \forall a \in A$$

$$0 \leq f_P^{\{i\}} \qquad \forall i \in [\kappa], P \in \mathcal{P}_h(S_i, T_i)$$

For a multi-commodity length-constrained flow $f$, we will use the shorthand $f(a) = \sum_i \sum_{P \ni a} f_P^{\{i\}}$. Likewise we let $\text{val}(f) = \sum_i \text{val}(f^{\{i\}})$ be the total flow we send. An $h$-length multi-commodity flow, then, is simply a feasible solution to this LP.

**Definition A.1** ($h$-Length Multi-Commodity Flow)**.** *Given digraph $D = (V, A)$ with lengths $\ell$, capacities $U$ and source, sink pairs $\{(S_i, T_i)\}_i$, an $h$-length $\{(S_i, T_i)\}_i$ flow is any feasible solution to Multi Length-Constrained Flow LP.*

With the above definition of multi-commodity length-constrained flows we can now define moving cuts as the dual of length-constrained flows. In particular, taking the dual of the above LP we get the multi-commodity moving cut LP with a variable $w_a$ for each $a \in A$ and a variable $y_i$ for every $i \in [\kappa]$.

$$\min \sum_{a \in A} U_a \cdot w_a \quad \text{s.t.} \qquad \text{(Multi Moving Cut LP)}$$
$$\sum_{a \in P} w_a \geq 1 \qquad \forall i \in [\kappa], P \in \mathcal{P}_h(S_i, T_i)$$
$$0 \leq w_a \qquad \forall a \in A, i \in [\kappa]$$

A multi-commodity $h$-length moving cut is simply a feasible solution to this LP.

**Definition A.2** ($h$-Length Moving Cut)**.** *Given digraph $D = (V, A)$ with lengths $\ell$, capacities $U$ and source, sink pairs $\{(S_i, T_i)\}_i$, a multi-commodity $h$-length moving cut is any feasible solution to Multi Moving Cut LP.*

We will use $f$ and $w$ to stand for solutions to Multi Length-Constrained Flow LP and Multi Moving Cut LP respectively. We say that $(f, w)$ is a feasible pair if both $f$ and $w$ are feasible for their respective LPs and that $(f, w)$ is $(1 \pm \epsilon)$-approximate for $\epsilon > 0$ if the moving cut certifies the value of the length-constrained flow up to a $(1 - \epsilon)$; i.e. if $(1 - \epsilon) \sum_a U_a \cdot w_a \leq \min_i \text{val}(f^{\{i\}})$.

When we are working in CONGEST we will say that $f$ is computed if each vertex $v$ stores the value $f_a^{(h',i)} := \sum_{P \in \mathcal{P}_{h,h'}(s,a,t)} f_P^{\{i\}}$. Here, we let $\mathcal{P}_{h,h'}(s, a, t)$ be all paths in $\mathcal{P}_h(s, t)$ of the form $P' = (a_1, a_2, \ldots a, b_1, b_2, \ldots)$ where the path $(a, b_1, b_2, \ldots)$ has length exactly $h'$ according to $l$. We say multi-commodity moving cut $w$ is computed in CONGEST if each vertex $v$ knows the value of $w_a$ for every arc incident to $v$. Likewise, we imagine that each node in the first round knows the capacities and lengths of its incident edges.

With the above notions, we can now state our main result for multi-commodity length-constrained flows and moving cuts which say that one can compute a feasible pair $(f, w)$ in parallel and distributedly. In the following we say that length-constrained flow $f$ is integral if $f_P^{\{i\}}$ is an integer for every path in $\mathcal{P}_h(S_i, T_i)$ for every $i$.

More generally than $\kappa$ commodities, we solve the problem provided our commodoties can be grouped into $\kappa$ batches that are far apart.

**Definition A.3** ($\kappa$-Batchable)**.** *Given digraph $D$ with lengths $\ell$ and source, sink set pairs $\{S_i, T_i\}_i$ we say that a $\{S_i, T_i\}_i$ is $\kappa$-batchable if the pairs of $\{S_i, T_i\}_i$ can be partitioned into batches $\{\mathcal{S}_j, \mathcal{T}_j\}_j$ if*

1. *For each $i$ there some $j$ such that $S_i \in \mathcal{S}_j$ and $T_i \in \mathcal{T}_j$;*

2. *For each $i$ and $i'$, if $v \in S_i \cup T_i$ and $v' \in S_{i'} \cup T_{i'}$ and $S_i, S_{i'} \in \mathcal{S}_j$ for some $j$ then $d_\ell(v, v') > 2h$.*

Observe that if the number of commodities is $\kappa$ then the set of source, sink pairs is trivially $\kappa$-batchable.

The following summarizes our main result for computing multi-commodity length-constrained flows and moving cuts.

**Theorem A.1.** *Given a digraph $D = (V, A)$ with capacities $U$, lengths $\ell$, length constraint $h \geq 1$, $0 < \varepsilon < 1$ and source and sink vertices $S, T \subseteq V$, and $\kappa$-batchable source, sink pairs $\{S_i, T_i\}_i$, one can compute a feasible multi-commodity $h$-length flow, moving cut pair $(f, w)$ that is $(1 \pm \epsilon)$-approximate in:*

1. *Deterministic parallel time $\tilde{O}(\kappa \cdot \frac{1}{\varepsilon^9} \cdot h^{17})$ with $m$ processors*

2. *Randomized CONGEST time $\tilde{O}(\kappa \cdot \frac{1}{\varepsilon^9} \cdot h^{17})$ with high probability;*

3. *Deterministic CONGEST time $\tilde{O}\left(\kappa \cdot \frac{1}{\varepsilon^9} \cdot h^{17} + \kappa \cdot \frac{1}{\varepsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.*

*Furthermore, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\kappa \cdot \frac{h}{\epsilon^4}\right)$ and $f_j$ is an integral $h$-length $S_i$-$T_i$ flow for some $i$.*

## A.2 Computing Multi-Commodity Length-Constrained Flows and Moving Cuts

We proceed to use our $(1+\epsilon)$-lightest path blockers and multiplicative weights to compute multi-commodity length-constrained flows and moving cuts. Our strategy is more or less that of Section 11 but now we iterate through our batches of commodities; our analysis is mostly unchanged but we include it here for completeness.

Formally, our algorithm is given in Algorithm 4. Throughout our analysis we will refer to the innermost loop of Algorithm 4 as one "iteration."

We begin by observing that $\lambda$ always lower bounds $d_w^{(h)}(S_i, T_i)$ for every $i$.

**Lemma A.4.** *It always holds that $\lambda \leq d_w^{(h)}(S_x, T_x)$ for every $x$ in Algorithm 4.*

*Proof.* Fix an $x$ and a value of $\lambda$ and let $S = S_x$ and $T = T_x$. Our proof is by induction. The statement trivially holds at the beginning of our algorithm.

Let $\lambda_i$ be the value of $\lambda$ at the beginning of the $i$th iteration. We argue that if $d_w^{(h)}(S, T) = \lambda_i$ then after $\Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ additional iterations we must have $d_w^{(h)}(S, T) \geq (1 + \varepsilon_0) \cdot \lambda_i$. Let $\lambda_i' = (1 + \epsilon_0) \cdot \lambda$ be $\lambda$ after these iterations. Let $\hat{f}_j$ be our lightest path blocker in the $j$th iteration for $(S_x, T_x)$.

Assume for the sake of contradiction that $d_w^{(h)}(S, T) < \lambda_i'$ after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations. It follows that there is some path $P \in \mathcal{P}_h(S, T)$ with weight at most $\lambda_i'$ after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations. However, notice that by definition of an $h$-length $(1+\epsilon_0)$-lightest path blocker $\hat{f}_j$ (Definition 10.1), we know that for every $j \in \left[i, i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)\right]$ there is some $a \in P$ for which $\hat{f}_j(a) = U_a$. By averaging, it follows that there is some single arc $a \in P$ for which $\hat{f}_j(a) = U_a$ for

**Algorithm 4** Multi-Commodity Length-Constrained Flows and Moving Cuts

**Input:** digraph $D = (V, A)$ with lengths $\ell$, capacities $U$, length constraint $h$ and $\kappa$-batchable source, sink pairs $\{S_i, T_i\}_i$ where $S_i, T_i \subseteq V$ for every $i$ and an $\varepsilon \in (0, 1)$.
**Output:** $(1 \pm \epsilon)$-approximate $h$-length multi-commodity flow $f$ and moving cut $w$.
Let $\epsilon_0 = \frac{\epsilon}{6}$, let $\zeta = \frac{1+2\varepsilon_0}{\varepsilon_0} + 1$ and let $\eta = \frac{\varepsilon_0}{(1+\varepsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$.
Initialize $w_a \leftarrow \left(\frac{1}{m}\right)^{\zeta}$ for all $a \in A$.
Initialize $\lambda \leftarrow \left(\frac{1}{m}\right)^{\zeta}$.
Initialize $f_P^{\{i\}} \leftarrow 0$ for all $i$ and $P \in \mathcal{P}_h(S_i, T_i)$.
**while** $\lambda < 1$ **do**:
  **for** $j \in [\kappa]$ and each batch $(\mathcal{S}_j, \mathcal{S}_j)$ **do**
    **for** each $(S_i, T_i)$ with $S_i \in \mathcal{S}_j$ and $T_i \in \mathcal{T}_j$ in parallel **do**
      **for** $\Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ repetitions **do**
        Compute an $h$-length $(1+\epsilon_0)$-lightest path blocker $\hat{f}$ (using Theorem 10.1 with $\lambda$).
        **Length-Constrained Flow (Primal) Update:** $f^{\{i\}} \leftarrow f^{\{i\}} + \eta \cdot \hat{f}$.
        **Moving Cut (Dual) Update:** $w_a \leftarrow (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a$ for every $a \in A$.
  $\lambda \leftarrow (1+\epsilon_0) \cdot \lambda$
**return** $(f, w)$.

at least $\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)$ of these $j \in [i, i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)]$. Since every such arc starts with dual value $(\frac{1}{m})^{\zeta}$ and multiplicatively increases by a $(1+\epsilon_0)$ factor in each of these updates, such an arc after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ many iterations must have $w_a$ value at least $(\frac{1}{m})^{\zeta} \cdot (1+\epsilon_0)^{\Theta\left(\frac{\log_{1+\epsilon_0} n}{\epsilon_0}\right)} \geq n^2$ for an appropriately large hidden constant in our $\Theta$. However, by assumption, the weight of $P$ is at most $\lambda_i'$ after $i + \Theta\left(\frac{h \log_{1+\epsilon_0} n}{\epsilon_0}\right)$ iterations and this is at most 2 since $\lambda_i < 1$ since otherwise our algorithm would have halted. But $2 < n^2$ and so we have arrived at a contradiction.

Repeatedly applying the fact that $\lambda_i' = (1+\epsilon_0)\lambda_i$ gives that $\lambda$ is always a lower bound on $d_w^{(h)}(S, T)$. □

We next prove the feasibility of our solution.

**Lemma A.5.** *The pair $(f, w)$ returned by Algorithm 4 are feasible for Multi Length-Constrained Flow LP and Multi Moving Cut LP respectively.*

*Proof.* First, observe that by Lemma A.4 we know that $\lambda$ is always a lower bound on $d_w^{(h)}(S_i, T_i)$ for every $i$ and so since we only return once $\lambda > 1$, the $w$ we return is always feasible.

To see that $f$ is feasible it suffices to argue that for each arc $a$, the number of times a path containing $a$ has its primal value increased is at most $\frac{U_a}{\eta}$. Notice that each time we increase the primal value on a path containing arc $a$ by $\eta$ we increase the dual value of this edge by a multiplicative $(1+\epsilon_0)^{1/U_a}$. Since the weight of our arcs according to $w$ start at $(\frac{1}{m})^{\zeta}$, it follows that if we increase the primal value of $k$ paths incident to arc $a$ then $w_a = (1+\epsilon_0)^{k/U_a} \cdot (\frac{1}{m})^{\zeta}$. On the other hand, by assumption when we increase the dual value of an arc $a$ it must be the case that $w_a < 1$ since otherwise $d_w^{(h)}(S, T) \geq 1$, contradicting the fact that $\lambda$ always lower bounds $d_w^{(h)}(S, T)$.

It follows that $(1+\epsilon_0)^{k/U_a} \cdot (\frac{1}{m})^{\zeta} \leq 1$ and so applying the fact that $\ln(1+\epsilon_0) \geq \frac{\epsilon_0}{1+\epsilon_0}$ for $\epsilon_0 > -1$ and our definition of $\zeta$ and $\eta$ we get

$$\begin{aligned} k &\leq \frac{\zeta \cdot (1+\varepsilon_0)}{\varepsilon_0} \cdot U_a \log m \\ &= \frac{U_a}{\eta} \end{aligned}$$

as desired. □

We next prove the near-optimality of our solution.

**Lemma A.6.** *The pair $(f, w)$ returned by Algorithm 4 satisfies $(1-\epsilon)\sum_a w_a \leq \sum_i \sum_{P \in \mathcal{P}_h(S_i,T_i)} f_P$.*

*Proof.* Fix an iteration $i$ which is an iteration for the $j$th batch and let $\hat{f}$ be the sum of all lightest path blockers that we compute in parallel for each $(S_i, T_i) \in (\mathcal{S}_j, \mathcal{T}_j)$ in this iteration. Let $k_i$ be $\text{val}(\hat{f})$, let $\lambda_i$ be $\lambda$ at the start of this iteration and let $D_i := \sum_a w_a$ be our total dual value at the start of this iteration. Notice that $\frac{1}{\lambda_i} \cdot w$ is dual feasible and has cost $\frac{D_i}{\lambda_i}$ by Lemma A.4. If $\beta$ is the optimal dual value then by optimality it follows that $\beta \leq \frac{D_i}{\lambda_i}$, giving us the upper bound on $\lambda_i$ of $\frac{D_i}{\beta}$. By how we update our dual, our bound on $\lambda_i$ and $(1+x)^r \leq 1 + xr$ for any $x \geq 0$ and $r \in (0,1)$ we have that

$$\begin{aligned} D_{i+1} &= \sum_a (1+\epsilon_0)^{\hat{f}(a)/U_a} \cdot w_a \cdot U_a \\ &\leq \sum_a \left(1 + \frac{\epsilon_0 \hat{f}(a)}{U_a}\right) \cdot w_a \cdot U_a \\ &= D_i + \epsilon_0 \sum_a \hat{f}(a) w_a \\ &\leq D_i + \epsilon_0 (1+2\varepsilon_0) \cdot k_i \lambda_i \\ &\leq D_i \left(1 + \frac{(1+2\varepsilon_0)\varepsilon_0 \cdot k_i}{\beta}\right) \\ &\leq D_i \cdot \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot k_i}{\beta}\right). \end{aligned}$$

Let $T-1$ be the index of the last iteration of our algorithm; notice that $D_T$ is the value of $w$ in our returned solution. Let $K := \sum_i k_i$. Then, repeatedly applying this recurrence gives us

$$\begin{aligned} D_T &\leq D_0 \cdot \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot K}{\beta}\right) \\ &= \left(\frac{1}{m}\right)^{\zeta-1} \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot K}{\beta}\right) \end{aligned}$$

On the other hand, we know that $w$ is dual feasible when we return it, so it must be the case that $D_T \geq 1$. Combining this with the above upper bound on $D_T$ gives us $1 \leq \left(\frac{1}{m}\right)^{\zeta} \exp\left(\frac{(1+2\varepsilon_0)\varepsilon_0 \cdot K}{\beta}\right)$. Solving for $K$ and using our definition of $\zeta$ gives us

$$\beta \log m \cdot \frac{\zeta - 1}{(1+2\varepsilon_0) \cdot \varepsilon_0} \leq K$$

$$\beta \log m \cdot \frac{1}{\varepsilon_0^2} \leq K.$$

However, notice that $K\eta$ is the primal value of our solution so using our choice of $\eta$ and rewriting this inequality in terms of $K\eta$ by multiplying by $\eta = \frac{\varepsilon_0}{(1+\varepsilon_0)\cdot\zeta} \cdot \frac{1}{\log m}$ and applying our definition of $\zeta = \frac{1+2\varepsilon_0}{\varepsilon_0} + 1$ gives us

$$\begin{aligned} \frac{\beta}{\varepsilon_0 \cdot (1+\varepsilon_0) \cdot \zeta} &\leq K\eta \\ \frac{\beta}{(1+\varepsilon_0)(1+3\varepsilon_0)} &\leq K\eta. \end{aligned} \tag{9}$$

Moreover, by our choice of $\varepsilon_0 = \frac{\varepsilon}{6}$ and the fact that $\frac{1}{1+x+x^2} \geq 1 - x$ for $x \in (0,1)$ we get

$$\begin{aligned} 1 - \varepsilon &\leq \frac{1}{1+\varepsilon+\varepsilon^2} \\ &\leq \frac{1}{(1+\frac{1}{2}\varepsilon)^2} \\ &\leq \frac{1}{(1+3\varepsilon_0)^2} \\ &\leq \frac{1}{(1+\varepsilon_0)(1+3\varepsilon_0)}. \end{aligned} \tag{10}$$

Combining Equation (9) and Equation (10) we conclude that

$$(1-\varepsilon)\cdot\beta \leq K\eta.$$

□

We conclude with our main theorem by proving that we need only iterate our algorithm $\tilde{O}\left(\kappa \cdot \frac{h}{\epsilon^4}\right)$ times.

**Theorem A.1.** *Given a digraph $D = (V, A)$ with capacities $U$, lengths $\ell$, length constraint $h \geq 1$, $0 < \varepsilon < 1$ and source and sink vertices $S, T \subseteq V$, and $\kappa$-batchable source, sink pairs $\{S_i, T_i\}_i$, one can compute a feasible multi-commodity $h$-length flow, moving cut pair $(f, w)$ that is $(1 \pm \epsilon)$-approximate in:*

1. *Deterministic parallel time $\tilde{O}(\kappa \cdot \frac{1}{\varepsilon^9} \cdot h^{17})$ with $m$ processors*
2. *Randomized CONGEST time $\tilde{O}(\kappa \cdot \frac{1}{\varepsilon^9} \cdot h^{17})$ with high probability;*
3. *Deterministic CONGEST time $\tilde{O}\left(\kappa \cdot \frac{1}{\varepsilon^9} \cdot h^{17} + \kappa \cdot \frac{1}{\varepsilon^7} \cdot h^{16} \cdot (\rho_{CC})^{10}\right)$.*

*Furthermore, $f = \eta \cdot \sum_{j=1}^{k} f_j$ where $\eta = \tilde{\Theta}(\epsilon^2)$, $k = \tilde{O}\left(\kappa \cdot \frac{h}{\epsilon^4}\right)$ and $f_j$ is an integral $h$-length $S_i$-$T_i$ flow for some $i$.*

*Proof.* By Lemma A.5 and Lemma A.6 we know that our solution is feasible and $(1 \pm \epsilon)$-optimal so it only remains to argue the runtime of our algorithm and that the returned flow decomposes in the stated way.

We argue that we must only run for $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ total iterations. Since $\lambda$ increases by a multiplicative $(1+\epsilon_0)$ after every $\Theta\left(\kappa \cdot \frac{h \log n}{\epsilon_0^2}\right)$ iterations and starts at least $\left(\frac{1}{m}\right)^{\Theta(\frac{1}{\epsilon_0})}$, it follows by Lemma A.4 that after $y \cdot \Theta\left(\kappa \cdot \frac{h \log n}{\epsilon_0^2}\right)$ total iterations the $h$-length distance between every $S_i$ and $T_i$ is at least $(1+\epsilon_0)^y \cdot \left(\frac{1}{m}\right)^{\Theta(1/\epsilon_0)}$. Thus, for $y \geq \Omega\left(\frac{\ln_{1+\epsilon_0} m}{\epsilon_0}\right) = \Omega\left(\frac{\ln n}{\epsilon_0^2}\right)$ we have that every $S_i$ and $T_i$ are at least 1 apart in $h$-length distance. Consequently, our algorithm must run for at most $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon_0^4}\right) = O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ many iterations.

Our running time is immediate from the the bound of $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ on the number of iterations of the while loop, the fact that commodities in the same batch can be updated in parallel and the running times given in Theorem 10.1 for computing our $h$-length $(1+\epsilon_0)$-lightest path blocker.

Lastly, the flow decomposes in the stated way because we have at most $O\left(\kappa \cdot \frac{h \log^2 n}{\epsilon^4}\right)$ iterations and each $f_j$ is an integral $S$-$T$ flow by Theorem 10.1. Thus, our final solution is $\eta \cdot \sum_{j=1}^{k} f_j$ and $k = \tilde{O}\left(\frac{h}{\epsilon^4}\right)$. □

## References


[1] Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 322–335, 2020. 1, 8

[2] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM (JACM)*, 62(5):1–25, 2015. 3

[3] Baruch Awerbuch and David Peleg. Sparse partitions. In *Symposium on Foundations of Computer Science (FOCS)*, pages 503–513. IEEE, 1990. 12

[4] Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondřej Pangrác, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. *ACM Transactions on Algorithms (TALG)*, 7(1):1–27, 2010. 1, 7

[5] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *Journal of the ACM (JACM)*, 68(5):1–30, 2021. 3

[6] Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 165–174, 2017. 11

[7] Alok Baveja and Aravind Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, 2000. 1

[8] Sebastian Brandt and Dennis Olivetti. Truly tight-in-$\delta$ bounds for bipartite maximal matching and variants. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 69–78, 2020. 3

[9] Andrei Z Broder, Alan M Frieze, and Eli Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM Journal on Computing*, 23(5):976–989, 1994. 1

[10] Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. *Distributed Computing*, 33(3):349–366, 2020. 11

[11] Yi-Jun Chang and Mohsen Ghaffari. Strong-diameter network decomposition. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 273–281, 2021. 12, 13

[12] Yi-Jun Chang and Thatchaphol Saranurak. Deterministic distributed expander decomposition and routing with applications in distributed derandomization. In *Symposium on Foundations of Computer Science (FOCS)*, pages 377–388. IEEE, 2020. 1, 3, 10, 11, 14, 15, 44, 48, 49

[13] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation via short cycle decompositions. In *Symposium on Foundations of Computer Science (FOCS)*. SIAM, 202. 13

[14] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020. 1

[15] Edith Cohen. Approximate max-flow on small depth networks. *SIAM Journal on Computing*, 24(3):579–597, 1995. 10, 14, 23, 24, 25, 27

[16] Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 33(3):279–291, 2020. 3

[17] Manuela Fischer. *Local Algorithms for Classic Graph Problems*. PhD thesis, ETH Zurich, 2021. 3

[18] Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In *International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 13

[19] Venkatesan Guruswami, Sanjeev Khanna, Rajmohan Rajaraman, Bruce Shepherd, and Mihalis Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67(3):473–496, 2003. 1, 7, 48

[20] Bernhard Haeupler, David Wajc, and Goran Zuzic. Network coding gaps for completion times of multiple unicasts. In *Symposium on Foundations of Computer Science (FOCS)*, pages 494–505. IEEE, 2020. 1

[21] Bernhard Haeupler, D Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 356–369, 2021. 1, 8

[22] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 1166–1179, 2021. 2

[23] Bernhard Haeupler, Harald Raecke, and Mohsen Ghaffari. Hop-constrained expander decompositions; oblivious routing, and distributed universal optimality. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2022. 4, 50, 52

[24] Yael Hitron and Merav Parter. General congest compilers against adversarial edges. In *International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. 7, 13, 14

[25] Richard M Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. 1989. 5, 22

[26] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *Journal of the ACM (JACM)*, 66(1):1–50, 2018. 3

[27] Jon M Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, Massachusetts Institute of Technology, 1996. 1

[28] Christos Koufogiannakis and Neal E Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *International Symposium on Distributed Computing*, pages 221–238. Springer, 2009. 3

[29] Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4): 441–454, 1993. 11

[30] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 129–136, 2008. 1, 9, 10, 14, 15

[31] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM journal on computing*, 15(4):1036–1053, 1986. 15

[32] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–203, 2013. 11

[33] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 950–961. IEEE, 2017. 3

[34] Merav Parter and Eylon Yogev. Optimal short cycle decomposition in almost linear time. In *International Colloquium on Automata, Languages and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. 7, 13, 14

[35] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000. 5

[36] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 755–764, 2010. 3

[37] Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 350–363, 2020. 13

[38] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2635. SIAM, 2019. 1

[39] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2004. 3

[40] Luca Trevisan. Approximation algorithms for unique games. In *Symposium on Foundations of Computer Science (FOCS)*, pages 197–205. IEEE, 2005. 3