Zuzic, G., Goranci, G., Ye, M., Haeupler, B. and Sun, X. (2022) Universally-Optimal Distributed Shortest Paths and Transshipment via Graph-Based ℓ1-Oblivious Routing. In: 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 9-12 Jan 2022, pp. 2549-2579. ISBN 9781611977073

(doi: 10.1137/1.9781611977073.100)

Universally-Optimal Distributed Shortest Paths and Transshipment via Graph-Based ℓ_1-Oblivious Routing[*]

Goran Zuzic[†]   ⓡ   Gramoz Goranci   ⓡ
ETH Zurich   School of Computing Science,
goran.zuzic@inf.ethz.ch   University of Glasgow
                                        gramoz.goranci@glasgow.ac.uk

Mingquan Ye   ⓡ   Bernhard Haeupler[‡]   ⓡ
University of Illinois at Chicago   ETH Zurich & Carnegie Mellon University
mye9@uic.edu   haeupler@cs.cmu.edu

Xiaorui Sun[§]
University of Illinois at Chicago
xiaorui@uic.edu

Abstract

We provide universally-optimal distributed graph algorithms for $(1+\varepsilon)$-approximate shortest path problems including shortest-path-tree and transshipment.

The universal optimality of our algorithms guarantees that, on any n-node network G, our algorithm completes in $T \cdot n^{o(1)}$ rounds whenever a T-round algorithm exists for G. This includes $D \cdot n^{o(1)}$-round algorithms for any planar or excluded-minor network. Our algorithms never require more than $(\sqrt{n} + D) \cdot n^{o(1)}$ rounds, resulting in the first sub-linear-round distributed algorithm for transshipment.

The key technical contribution leading to these results is the first efficient $n^{o(1)}$-competitive linear ℓ_1-oblivious routing operator that does not require the use of ℓ_1-embeddings. Our construction is simple, solely based on low-diameter decompositions, and—in contrast to all known constructions—directly produces an oblivious flow instead of just an approximation of the optimal flow cost. This also has the benefit of simplifying the interaction with Sherman's multiplicative weight framework [SODA'17] in the distributed setting and its subsequent rounding procedures.

Contents

1 Introduction

Computing single-source shortest paths (SSSP) is one of the most fundamental and well-studied problems in combinatorial optimization. Likewise, the distributed version of the SSSP problem has received wide-reaching attention in the distributed community [Nan14, EN16, HKN16, BKKL17, Elk17, FN18, GL18a, HL18, EN19, LPSP19, CM20]. For $(1 + \varepsilon)$-approximate SSSP, this effort culminated in a distributed algorithm that is guaranteed to complete in $\tilde{\Theta}(\sqrt{n} + D)$ rounds[1] [BKKL17] on every n-node network with hop-diameter D in the standard message-passing model (CONGEST).

This $\tilde{\Theta}(\sqrt{n} + D)$-round algorithm is *existentially-optimal*, in the sense that there exists a family of pathological networks where one cannot do better [SHK+12]. However, the barrier that precludes fast algorithms in these pathological networks does not apply to most networks of interest, and faster algorithms often exist in non-worst-case networks such as planar graphs [GH16]. This motivates a much stronger notion of optimality—called *universal optimality*—which requires a single (i.e., uniform) algorithm to be as fast as possible on *every* network G, i.e., (approximately) as fast as the running time of any other algorithm on G. In particular, for any (unknown) network G the universally-optimal algorithm must be competitive with whichever algorithm happens to be the fastest for G, including any algorithm explicitly designed to be fast on G (but potentially very slow on any other network). More formally, let $T_{\mathcal{A}}(G)$ be the running time on the network G of the longest-running (i.e., worst-case) problem-specific input to an algorithm $\mathcal{A}$. An algorithm $\mathcal{A}$ is ($n^{o(1)}$-) universally optimal if $T_{\mathcal{A}}(G)$ is $n^{o(1)}$-competitive with $T_{\mathcal{B}}(G)$ for every other correct algorithm $\mathcal{B}$, i.e., if $T_{\mathcal{A}}(G) \leq n^{o(1)} \cdot T_{\mathcal{B}}(G)$ [HWZ21].

This beyond-worst-network guarantee is therefore in some sense the strongest form in which an algorithm can adjust (on the fly) to the network topology it is run on. The concept of universal optimality was already (informally) proposed in 1998 by Garay, Kutten and Peleg [GKP98] but only over the last six years work towards universally-optimal distributed algorithms has found traction, in the form of the low-congestion shortcut framework [GH16, HIZ16a, HIZ16b, HLZ18, KKOI21, GH21, HWZ21]. This framework can be used to design universally-optimal distributed algorithms for problems that can be solved very fast using a communication primitive called partwise aggregation. For example, such universally-optimal algorithms exist for (exact) minimum spanning tree (MST) and $(1 + \varepsilon)$-minimum cut. The shortcut framework also implies fast algorithm with concrete guarantees for special graph classes, including any excluded-minor graph family. We refer to Section 2 and [HWZ21] for more details on universal optimality and the shortcut framework but remark that SSSP algorithms have been notoriously hard to achieve within the shortcut framework. The only non-trivial SSSP algorithm [HL18] within the framework has a bad super-constant approximation factor of at least polylog n (or even $n^{o(1)}$). An improvement using [BKKL17] towards better approximations was proposed and stated as the main open problem in the abstract and conclusion of [HL18], however this suggested approach turned out to be (provably) impossible (see below).

Our results. In this paper, we resolve the open question of [HL18] by giving a universally-optimal distributed algorithms for $(1 + \varepsilon)$-approximate single-source shortest path (SSSP) whenever there exists a $n^{o(1)}$-round algorithm. The running time of our algorithm is $n^{o(1)}$-competitive with the fastest possible correct algorithm and thus leads to ultra-fast sub-polynomial-round distributed algorithms on networks of interest. For example, our SSSP algorithm provably completes in $D \cdot n^{o(1)}$ rounds on any minor-free network.

---

[1] We use $\sim$, as in $\tilde{O}$, to hide $\text{poly}(\log n)$ factors, where n is the number of nodes in the network.

We also give the first universally-optimal distributed algorithms for $(1 + \varepsilon)$-approximate transshipment. Transshipment is a well-studied generalization of the shortest path problem also known as uncapacitated min-cost flow, earth mover's distance, or Wasserstein metric. No sub-linear-round algorithm was known for $O(1)$- or $(1 + \varepsilon)$-approximate transshipment in the distributed setting (CONGEST). Therefore even our worst-case running time of $(\sqrt{n} + D) \cdot n^{o(1)}$ rounds for $(1 + \varepsilon)$-approximate distributed transshipment improves over the state-of-the-art $\tilde{O}(n)$-round CONGEST algorithm [BKKL17] for this problem.

Challenges. All approaches to the distributed $(1 + \varepsilon)$-approximate SSSP problem that appear in the literature can be categorized as follows: (1) Either they select $\tilde{\Theta}(\sqrt{n})$ nodes (e.g., via random sampling), construct a backbone graph on the selected nodes in $\tilde{\Theta}(\sqrt{n})$ rounds, and finally solve some variant of the shortest path problem on the selected nodes. (2) Or they simulate all-to-all communications on a large number of nodes.

Algorithms using hop-sets and similar ideas like [Nan14, EN16, HKN16, Elk17, EN19, LPSP19, CM20] are of the first type. The algorithm of [BKKL17] actually fall into both categories simultaneously: [BKKL17] explicitly uses the $\tilde{\Theta}(\sqrt{n})$ reduction to a backbone graph from [Nan14] and then simulates a fast BROADCAST CONGESTED CLIQUE algorithm on the $\tilde{\Theta}(\sqrt{n})$ backbone nodes, which can be easily achieved in CONGEST with a slow-down of $\tilde{\Theta}(\sqrt{n})$ (via flooding). The algorithms from [GL18b] are the cleanest example of the last category. They simulate almost any near-linear time PRAM algorithm (for SSSP, transshipment, or almost any other problem) by implementing all-to-all communications between a linear number of nodes (in this case serving any communication which does not involve any node too often). This is possible in graphs with good expansion with a slowdown linear in the mixing time of the graph but impossible to do in sub-linear time in general graphs.

Once one looks at these algorithms in this way it is clear that these approaches fundamentally cannot *ever* lead to a $o(\sqrt{n})$ runtime, even when run on a simple network where one would/should expect $\tilde{o}(\sqrt{n})$-round algorithms to exist.

Our paper uses a different approach to SSSP, which requires us to solve the more-general transshipment problem. On a high level, transshipment asks us to find the minimum cost flow which satisfies a given demand $d \in \mathbb{R}^V$ without any capacity constraints on edges. For example, the s-t shortest path corresponds to transshipment with the demand $d(v) = (\mathbb{1}[v = s] - \mathbb{1}[v = t])$. We solve the transshipment problem using *Sherman's framework* [She17b] which yielded great success in the sequential and parallel settings [Li20, ASZ20]. In particular, we show that Sherman's framework for ℓ_1 can be implemented in CONGEST within the shortcut framework. The main technical barrier preventing Sherman's framework from being utilized in the distributed setting was the construction of a crucial object called *(linear) ℓ_1-oblivious routing (cost approximator)*. Concretely, the ℓ_1-oblivious routing is a linear operator R (i.e., a matrix) that takes the demand $d \in \mathbb{R}^V$ as input, and outputs a vector Rd such that $\|Rd\|_1$ is a good approximation to the (cost of the) optimal solution. All known approaches of constructing this matrix relied on first embedding the graph in ℓ_1 space and then routing the demand over this space. Unfortunately, while such approaches are well-suited for shared-memory settings, the approach is fundamentally broken in the distributed world where arbitrary data shuffling cannot be done efficiently. In more detail, all known oblivious routing algorithms route the demand of a vertex v to an intermediary vertex that depends on the ℓ_1-embedding coordinate of v; this seems infeasible to do at scale in a message-passing context.

Technical contribution. Our main technical contribution is a new construction of $n^{o(1)}$-approximate ℓ_1-oblivious routings that is graph-based and can be efficiently implemented in the distributed set-

ting. The construction is simple and relies on routing the demand along several low-diameter decompositions (LDDs) in a bottom-up way.

Our construction is particularly appealing in that it produces a *flow vector* that routes (i.e., satisfies) the given demand and whose cost is $n^{o(1)}$-competitive with the optimal flow that satisfies the demand. This is in contrast to previous "oblivious routings" which routed the demand over the fictional ℓ_1 space in a way that cannot be easily pulled back to a near-optimal flow in the underlying graph. In other words, previous constructions would be more aptly named *cost approximators*, rather than oblivious routings, since they do not produce a flow in the graph. This can be compared to $(1+\varepsilon)$-approximate maximum flow computation, where many fast algorithms construct a *congestion approximator* which approximates the solution [She13, Pen16]; it is known that fast construction of an oblivious routing with the same property (i.e., a flow with near-optimal congestion satisfying the demand) is a significantly harder task.

To simplify the presentation of distributed algorithms in the shortcut framework, we introduce a new interface to this framework called the *Distributed Minor-Aggregation model*. On one hand, the interface restricts the operations the nodes can perform in that they can only compute aggregates of adjacent nodes. On the other hand, the Distributed Minor-Aggregation model provides powerful high-level primitives like graph contractions which allow for succinct descriptions of otherwise complicated distributed algorithms. Our Minor-Aggregation model interface is our best attempt at making the most recent advancements in theoretical distributed computing and universal optimality [HWZ21, GHZ21] easily accessible. The idea is that any fast algorithm in the Minor-Aggregation model for a vast class of problems can be converted in a black-box way to a universally-optimal distributed algorithm in the standard CONGEST model. This gives rise to the following informal, but helpful, claim about (non-local) distributed algorithms: *An algorithm can be efficiently distributed if it can be written as only computing aggregates over a minor of the original network.* Our model connects nicely with similar intuitions or (implicit) models that have appeared in the distributed literature (e.g., [GKK+15, FGL+20]). The following informal theorem shows how to convert running times from the Minor-Aggregation model to the standard CONGEST model.

Theorem 1.1 (Informal, Minor-Aggregation model simulation)**.** *Any Minor-Aggregation algorithm on G that terminates in τ rounds for SSSP, transshipment, MST, etc., can be converted into a $\tau \cdot n^{o(1)}$-competitive universally-optimal algorithm in the CONGEST model on G. The algorithm is guaranteed to complete in $D \cdot \tau \cdot n^{o(1)}$ CONGEST rounds when G is a planar, genus-bounded, treewidth-bounded, excluded-minor graph, or an $n^{-o(1)}$-expander.*

Formal statements of our results. With the required preliminaries in place, we can now formally state our results. The following theorems concern $(1 + \varepsilon)$-approximate distributed SSSP and transshipment in the Minor-Aggregation model.

Theorem 1.2. (Distributed SSSP)**.** *Given an undirected and weighted graph G with edge weights in $[1, n^{O(1)}]$, there exists a distributed algorithm that computes $(1 + \varepsilon)$-approximate SSSP in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ Minor-Aggregation rounds.*

Theorem 1.3. (Distributed transshipment)**.** *Let G be a weighted graph with weights in $[1, n^{O(1)}]$ and let $\varepsilon > 0$ be a parameter. There exists an algorithm that computes $(1 + \varepsilon)$-approximate transshipment on G in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ Minor-Aggregation rounds.*

Combining the Minor-Aggregation model result with the simulation of Theorem 1.1, we immediately obtain the following set of results:

Corollary 1.4 (SSSP and transshipment in CONGEST)**.** *Let G be a weighted graph with weights*

3

in $[1, n^{O(1)}]$. We can solve both the $\left(1 + \frac{1}{n^{o(1)}}\right)$-approximate SSSP and $\left(1 + \frac{1}{n^{o(1)}}\right)$-approximate transshipment in $\mathrm{OPT}(G) \cdot n^{o(1)}$ CONGEST rounds, where OPT_G is the runtime of the fastest possible correct algorithm that works on G. Moveover, this algorithm is guaranteed to complete in $D \cdot n^{o(1)}$ CONGEST rounds if G is a planar, genus-bounded, treewidth-bounded, minor-free graph, or an $n^{-o(1)}$-expander.

The above set of results all rely on the distributed evaluation of ℓ_1-oblivious routing, which we state below for reference.

Theorem 1.5. (Distributed oblivious routing evaluation). *Let $G = (V, E)$ be a weighted graph with weights in $[1, n^{O(1)}]$. There exists an $\exp(O(\log n \cdot \log \log n)^{3/4})$-approximate ℓ_1-oblivious routing R such that we can perform the following computations in $\exp(O(\log n \cdot \log \log n)^{3/4})$ rounds of Minor-Aggregation. Given any distributedly stored node vector $d \in \mathbb{R}^V$ and edge vector $c \in \mathbb{R}^V$, we can evaluate and distributedly store $Rd \in \mathbb{R}^{\vec{E}}$ and $R^T c \in \mathbb{R}^V$.*

1.1 Related Work

Techniques from continuous optimization have brought breakthroughs for many problems in combinatorial optimization. Sherman [She13] obtained an approximate max-flow algorithm in undirected graphs with time complexity $O(m^{1+o(1)} \varepsilon^{-2})$ by constructing an $n^{o(1)}$-congestion approximator [Mad10] in nearly linear time. Peng [Pen16] improved the run-time to $O(m \mathrm{polylog}(n))$ by solving the approximate max-flow and constructing the cut-based hierarchical tree decompositions [RST14] recursively, and using ultra-sparsifiers [KMP10, ST14] to reduce the graph size. Sherman [She17b] designed a generalized preconditioner and applied that to produce an approximation algorithm for uncapacitated min-cost flow on undirected graphs with time complexity $O(m^{1+o(1)} \varepsilon^{-2})$. Kelner et al. [KLOS14] solved the approximate maximum concurrent multicommodity flow problem with k commodities in time $O(m^{1+o(1)} k^2 \varepsilon^{-2})$ by an $O(m^{o(1)})$-competitive oblivious routing scheme. Sherman [She17a] improved this time complexity to $O(mk\varepsilon^{-1})$ via the proposed area-convex regularization. In the distributed settings, Ghaffari et al. [GKK$^+$15] proposed an $(1+o(1))$-approximate algorithm that takes $(\sqrt{n} + D)n^{o(1)}$ rounds for max-flow in undirected weighted networks, where D is the hop-diameter of the communication network.

For the directed min-cost flow problem, Daitch and Spielman [DS08] provided an algorithm that runs in time $\widetilde{O}(m^{3/2} \log^2 U)$ by solving the linear equations efficiently; this time complexity was improved to $\widetilde{O}(m\sqrt{n} \log^2 U)$ by Lee and Sidford [LS14] by virtue of the new algorithm for solving linear programs. In terms of transshipment which is a generalization of the min-cost flow problem, Sherman [She17b] proposed an $(1 + \varepsilon)$-approximate algorithm with time complexity $m^{1+o(1)} \varepsilon^{-2}$ for uncapacitated and undirected min-cost flow problem using a generalized preconditioner obtained by ℓ_1 embedding and a hierarchical routing scheme in ℓ_1; Li [Li20] proposed a parallel algorithm with $m \mathrm{polylog}(n) \varepsilon^{-2}$ work and $\mathrm{polylog}(n) \varepsilon^{-2}$ time for the transshipment problem in [She17b] by utilizing the multiplicative weight update method to solve linear programs. A crucial ingredient tying together these approaches is a property of transshipment that solvers that return an approximate dual solution can be *boosted* to an $(1 + \varepsilon)$-approximate solver [Zuz21].

In the CONGEST model, substantial progress has been made on the SSSP problem in the past decade. For undirected graphs, Lenzen and Patt-Shamir [LPS13] proposed an $O(\varepsilon^{-1} \log \varepsilon^{-1})$-approximation algorithm that runs in $\widetilde{O}(n^{1/2+\varepsilon} + D)$ rounds for the weighted SSSP problem. This approximation factor was improved by [Nan14], in which Nanongkai presented an $\widetilde{O}(\sqrt{n}D^{1/4} + D)$-round $(1+o(1))$-approximation algorithm. Elkin and Neiman [EN16] gave an $(1+\varepsilon)$-approximation

algorithm that runs in $(\sqrt{n} + D)2^{\widetilde{O}(\sqrt{\log n})}$ rounds using their hopsets. Henzinger et al. [HKN16] proposed a deterministic $(1 + o(1))$-approximation algorithm that takes $n^{1/2+o(1)} + D^{1+o(1)}$ rounds. Haeupler and Li [HL18] proposed the first approximation algorithm for SSSP that adjusts to the topology of the communication network, and gave a $\widetilde{O}(\text{ShortcutQuality}(G)n^{o(1)})$-round algorithm with an approximation factor of $n^{o(1)}$, where $\text{ShortcutQuality}(G)$ is the quality of the best shortcut that can be constructed efficiently for the given topology. Ghaffari and Li [GL18b] designed distributed algorithms that run in $\tau_{mix} \cdot 2^{O(\sqrt{\log n})}$ rounds for $(1 + \varepsilon)$-approximate SSSP and transshipment, where τ_{mix} is the mixing time of the graph. More generally, [GL18b] proved that each parallel algorithm with efficient work that takes τ rounds can be simulated on any (expander) graph in $\tau_{mix} \cdot 2^{O(\sqrt{\log n})} \cdot \tau$ rounds in the CONGEST model. Besides these approximate algorithms, Elkin [Elk17] devised an exact SSSP algorithm for undirected graph that requires $O((n \log n)^{5/6})$ rounds when $D = O(\sqrt{n \log n})$, and $O((n \log n)^{2/3}D^{1/3})$ rounds for larger D.

For the directed SSSP problem, Ghaffari and Li [GL18a] presented an $\widetilde{O}(n^{3/4}D^{1/4})$-round algorithm, and an improved algorithm for large values of D with round complexity $\widetilde{O}(n^{3/4+o(1)} + \min\{n^{3/4}D^{1/6}, D^{6/7}\} + D)$. Forster and Nanongkai [FN18] gave two randomized algorithms that take $\widetilde{O}(\sqrt{nD})$ and $\widetilde{O}(\sqrt{n}D^{1/4}+n^{3/5}+D)$ rounds respectively, and obtained an $(1+\varepsilon)$-approximation algorithm with $\widetilde{O}((\sqrt{n}D^{1/4}+D)\varepsilon^{-1})$ rounds. Chechik and Mukhtar [CM20] proposed a randomized algorithm that takes $\widetilde{O}(\sqrt{n}D^{1/4}+D)$ rounds. Cao et al. [CFR21] provided an $(1+\varepsilon)$-approximation algorithm with round complexity $\widetilde{O}((\sqrt{n} + D + n^{2/5+o(1)}D^{2/5})\varepsilon^{-2})$. Censor-Hillel et al. [CHLP21] gave an exact SSSP algorithm with $(n^{7/6}/m^{1/2} + n^2/m) \cdot \tau_{mix} \cdot 2^{O(\sqrt{\log n})}$ rounds in the CONGEST model.

Besides CONGEST model, SSSP problem was also studied in the Congested Clique model [Nan14, HKN16, CHDKL20], Broadcast CONGEST model [BKKL17], Broadcast Congested Clique model [BKKL17], and Hybrid model [AHK+20, FHS20, KS20, CHLP21].

In addition, the all-pairs shortest paths (APSP) problem was also studied in various distributed models [HW12, CHKK+15, HP16, LG16, Elk17, HNS17, ARKP18, CHLT18, PR18, AR19, BN19, AHK+20, AR20, CHDKL20, DP20, KS20, CHLP21, DFKL21].

2 Preliminaries

Graph Notation Let $G = (V, E)$ be a simple undirected graph. We denote with $n := |V|$ the number of nodes, with $m := |E|$ the number of edges, and with D the hop-diameter of G. It is often convenient to direct E consistently. For simplicity and without loss of generality, we assume that the vertices $V = \{v_1, \ldots, v_n\}$ are numbered from 1 to n, and we define $\vec{E} = \{(v_i, v_j) \mid (v_i, v_j) \in E, i < j\}$. We identify E and $\vec{E}$ by the obvious bijection. We denote with $B \in \{-1, 0, 1\}^{V \times \vec{E}}$ the node-edge incidence matrix of G, which for any $e = (s, t) \in \vec{E}$ assigns $B_{s,e} = 1$, $B_{t,e} = -1$, and $B_{u,e} = 0$ for all other $u \in V$. In this paper, we typically assume the graph is weighted and the weights are polynomially bounded. For this, a weight or length function w assigns each edge $e \in E$ a weight $w_e \in [1, n^{O(1)}]$. The weight function can also be interpreted as a diagonal *weight matrix* $W \in [0, n^{O(1)}]^{E \times E}$ which assigns $W_{e,e} = w_e \geq 1$ for any $e \in \vec{E}$ (and 0 on all off-diagonal entries).

Flows and Transshipment A *demand* is a $d \in \mathbb{R}^V$. We say a demand is *proper* if $\mathbb{1}^T d = 0$. A *flow* is a vector $f \in \mathbb{R}^{\vec{E}}$. A flow f *routes* demand d if $Bf = d$. It is easy to see only proper demands are routed by flows. The cost $W(f)$ of a flow f is $\|Wf\|_1$. For a weighted graph G and a given proper demand d the *transshipment problem* asks to find a flow f_d^* of minimum cost among flows that route d. When the underlying graph is clear from the context, we let $\|d\|_{\text{OPT}} := W(f_d^*)$ denote

the cost of the optimal flow for routing demand d. For any $\alpha \geq 1$, we say a flow f is α-*approximate* if $W(f) \leq \alpha \cdot \|d\|_{\mathrm{OPT}}$ (such a flow does not necessarily need to route d). The transshipment problem naturally admits the following convex-programming formulation:

$$\min \ \|Wf\|_1 : Bf = d, \tag{2.1}$$

and its dual:

$$\max \ d^\top \phi : \left\|W^{-1}B^\top \phi\right\|_\infty \leq 1. \tag{2.2}$$

The entries in the vector $\phi \in \mathbb{R}^V$ are generally referred to as vertex *potentials*.

Distributed Algorithm (i.e., CONGEST Model). The distributed algorithms designed in this paper are message-passing algorithms following the standard CONGEST [Pel00] model of distributed computing. A network is given as an undirected graph $G = (V, E)$ in which nodes are individual computational units (i.e., have private memory and do their own computations). Communication between the nodes occurs in synchronous rounds. In each round, each pair of nodes adjacent in G exchanges an $O(\log n)$-bit message. Nodes perform arbitrary computation between rounds. Initially, nodes only know their unique $O(\log n)$-bit ID and the IDs of adjacent nodes as well as the weights of incident edges, for problems like SSSP with weights as input.

Asymptotic Notation. We use $\tilde{O}$ to hide polylogarithmic factors in n, i.e., $\tilde{O}(1) = \mathrm{polylog}\, n$ and $\tilde{O}(f(n, D)) = O(f(n, D) \cdot \mathrm{polylog}\, n)$. We use the term *with high probability* to denote success probabilities of at least $1 - n^{-C}$ where $C > 0$ is a constant that can be chosen arbitrarily large.

2.1   Low-Congestion Shortcut Framework and Part-wise Aggregation

Our universally-optimal distributed algorithms build on the *low-congestion shortcut framework* [GH16, HIZ16a, HIZ16b, KKOI21, HWZ21] which identifies the *part-wise aggregation task* as the crucial communication task in many optimization problems. The framework also introduced shortcuts as a near-optimal way of solving this communication problem. We summarize the facts needed in this paper and refer for more details to [HWZ21].

Aggregations. Let $\bigoplus$ be some function that combines two $O(\log n)$-bit messages into one (e.g., sum or max). We call such an $\bigoplus$ an *aggregation operator*. Given messages $x_1, x_2, \ldots, x_k$, their *aggregate* $\bigoplus_{i=1}^k x_i$ is the resulting message after iteratively taking two arbitrary messages m', m'', deleting them, and inserting $m' \bigoplus m''$ into the sequence until a single message remains (e.g., the sum-aggregation of $x_1, \ldots, x_k$ is $x_1 + \ldots + x_k$). Throughout this paper, $\bigoplus$ will be commutative and associative, making the value $\bigoplus_{i \in I} x_i$ unique and well-defined even if the order under which the messages are aggregated changes. However, it is often useful to consider more general aggregations, allowing us to use any *mergeable $O(\log n)$-bit sketches* [ACH+13] as an aggregation operator (examples of such operators include approximate heavy hitters, quantile estimation, frequency estimations, random sampling, etc.).

Part-wise aggregation (PA). The PA task is a central task used to solve many distributed operations problems. A formal definition follows.

Definition 2.1 (Part-wise aggregation (PA) task). Suppose that the nodes V of a graph $G = (V, E)$ are subdivided into a set of **connected** and **node-disjoint** parts $P_1, \ldots, P_k$. Initially, each node v chooses a private $O(\log n)$-bit input x_v and the task is to compute, for each part P_i, the aggregate over all private inputs belonging to that part, i.e., $\bigoplus_{v \in P_i} x_v$ (all nodes learn the same value).

Low-congestion shortcuts near-optimally solve the PA task. Specifically, [GH16, HWZ21] define a function ShortcutQuality($\cdot$) which assigns any undirected and unweighted graph G a positive integer ShortcutQuality(G) which characterizes (both as an upper and lower bound) the distributed complexity of solving PA. The lower bound is based on the network coding gaps of [HWZ20, HWZ21], while the upper bound is based on the very recent hop-constrained oblivious routings and efficient hop-constrained expander decompositions [GHZ21, GHR21].

Theorem 2.2. *Suppose A is a distributed CONGEST algorithm solving the part-wise aggregation task. For any G, the running time of A on G is at least $\tilde{\Omega}(\text{ShortcutQuality}(G))$ rounds [HWZ21]. Moreover, there is a randomized CONGEST algorithm that solves any PA instance in $\text{poly}(\text{ShortcutQuality}(G)) \cdot n^{o(1)}$ rounds [GHR21]. Specifically, if $\text{ShortcutQuality}(G) \le n^{o(1)}$, then we can solve PA in $n^{o(1)}$ rounds.*

We remark that on some special graph classes like excluded-minor graphs there exist algorithms that lose only $\tilde{O}(1)$ factors on the upper bound and can be made deterministic. [HIZ16a, GH21]

What makes part-wise aggregation and shortcuts so powerful is that they relate to many important (non-local) algorithmic problems, including, min-cut, MST, connectivity, etc. In particular, ShortcutQuality(G) is a lower bound on many important optimization problems, including all shortest-path problems considered in this paper.

Theorem 2.3 ([HWZ21]). *Suppose A is a (correct) distributed algorithm for SSSP or transshipment (or MST, min-cut, max-flow, etc.) with a sub-polynomial approximation ratio. For any G, the running time of A on G is at least $\tilde{\Omega}(\text{ShortcutQuality}(G))$ rounds.*

On the other hand, many problems can be solved with a small number of part-wise aggregations, giving universally-optimal algorithms with matching $\tilde{\Theta}(\text{ShortcutQuality}(G))$ lower and upper bounds on the round complexity (when $\text{ShortcutQuality}(G) \le n^{o(1)}$). The contribution of this paper is to show that this is true for approximate shortest-path problems.

We note that current solution to PA in $\text{poly}(\text{ShortcutQuality}(G)) \cdot n^{o(1)}$ only allows for $n^{o(1)}$-competitive universal optimality when there exists a $n^{o(1)}$-round solution. This is typically sufficient as many networks for interest have $\text{ShortcutQuality}(G) \le n^{o(1)}$. Moreover, a future results might improve the PA upper bound to $\tilde{O}(\text{ShortcutQuality}(G))$, in which case we would always get polylog-competitive universally-optimal algorithms.

Furthermore, on many special graph classes not only does the low-congestion shortcut framework guarantee that any universally-optimal algorithm is competitive with the fastest correct one, it also provides concrete upper bounds on this runtime. For example, when run on a planar graph, we know that $\text{ShortcutQuality}(G) = \tilde{O}(D)$ and we know how to solve the PA problem in $\tilde{O}(D)$ rounds, giving us a useful guarantee on the runtime on any algorithm that uses PA as a subroutine or any universally-optimal algorithm (e.g., the one presented in this paper). We provide a compiled list of concrete shortcut qualities and PA runtimes for specific graph classes.

Theorem 2.4 (Shortcut quality on special graphs). *Let G be a (undirected and unweighted) graph and let $D := \text{HopDiameter}(G)$. The following bounds hold:*

- *For all graphs G we have $\text{ShortcutQuality}(G) \le \tilde{O}(\sqrt{n}+D)$. PA can be solved in deterministic $\tilde{O}(\sqrt{n} + D)$ rounds. [GH16]*

- *When G is planar, then $\text{ShortcutQuality}(G) \le \tilde{O}(D)$. PA can be solved in deterministic $\tilde{O}(D)$ rounds. [GH16]*

- *When G has excluded minor, then* $\mathrm{ShortcutQuality}(G) \le \tilde{O}(D)$ *(the hidden constants depend on the excluded minor). The PA problem can be solved in deterministic $\tilde{O}(D)$ rounds. This generalizes the result for planar graphs. The same result holds even when G excludes $\tilde{O}(1)$-dense minors.* [GH21]

- *When G has treewidth at most k, then* $\mathrm{ShortcutQuality}(G) \le \tilde{O}(kD)$. *PA can be solved in randomized $\tilde{O}(kD)$.* [HIZ16b, HHW18]

- *When G has genus at most $g \ge 1$, then* $\mathrm{ShortcutQuality}(G) \le \tilde{O}(\sqrt{g}D)$. *PA can be solved in randomized $\tilde{O}(\sqrt{g}D)$ rounds.* [HIZ16b]

- *When G is an $n^{-o(1)}$-expander, then* $\mathrm{ShortcutQuality}(G) \le n^{o(1)}$. *PA can be solved in randomized $n^{o(1)}$ rounds.* [GL18b]

- *When G has hop-diameter D, then* $\mathrm{ShortcutQuality}(G) \le \tilde{O}(n^{(D-2)/(2D-2)} + D)$. *For $3 \le D = O(1)$, PA can be solved in randomized $\tilde{O}(n^{(D-2)/(2D-2)})$ rounds.* [KP21, KKOI21]

3 Graph-Based ℓ_1-Oblivious Routing via LDDs

In this section, we present our graph-based $n^{o(1)}$-approximate ℓ_1-oblivious routing R. We first define them in Section 3.1. Section 3.2 then presents an existential, model-oblivious, construction of R and proves its approximation guarantees. Section 3.4 presents a linear-algebraic interpretation of the constructed oblivious routing R.

3.1 Definition

For any k and $\alpha > 1$, Sherman [She17b] defined an α-approximate (linear) ℓ_1-preconditioner for a weighted graph G as a $k \times n$ matrix P, such that, for any proper demand d it holds that

$$\|d\|_{\mathrm{OPT}} \le \|Pd\|_1 \le \alpha \|d\|_{\mathrm{OPT}}.$$

Any such α-preconditioner which can be computed in τ rounds can be used to give a $\tilde{O}(\varepsilon^{-2}\alpha^2\tau)$ algorithm for transshipment [She17a]. We are constructing a strictly stronger object that is required to output a flow that routes any demand with α-approximate cost.

Definition 3.1. For a graph $G = (V, E)$, a matrix $R \in \mathbb{R}^{\vec{E} \times V}$ is a (linear) oblivious routing if for any proper demand $d \in \mathbb{R}^V$ the flow Rd routes d, i.e., $BRd = d$ for all $d \in \mathbb{R}^V$ with $\sum_{v \in V} d_v = 0$.

Definition 3.2. A routing R for a graph $G = (V, E)$ with weight matrix W is an α-approximate (linear) ℓ_1-oblivious routing if for any proper demand d the cost of the flow $f_{R,d} = Rd$ is at most $\alpha \|d\|_{\mathrm{OPT}}$, i.e., $W(f_{R,d}) = \|WRd\|_1 \le \alpha \|d\|_{\mathrm{OPT}}$.

Corollary 3.3. *If R is an α-approximate ℓ_1-oblivious routing for a graph G with weight matrix W then $P = WR$ is an α-approximate (linear) ℓ_1-preconditioner for G.*

Proof.
$$\|d\|_{\mathrm{OPT}} = \min_{f:Bf=d} \|Wf\|_1 \le \|Wf_{R,d}\|_1 = \|WRd\|_1 \le \alpha \|d\|_{\mathrm{OPT}}. \qquad \square$$

We remark that Li [Li20], like Sherman [She17a], constructs a sparse α-approximate (linear) ℓ_1-preconditioner P but at times calls such a matrix P an ℓ_1-oblivious routing even though Pd only produces a vector with the right norm and not a flow/routing. To our knowledge, this paper gives

the first matrix R that can be evaluated in almost linear time and is an α-approximate ℓ_1-oblivious routing with sub-polynomial α.

3.2 Construction

The main graph-theoretic tool we use to construct our ℓ_1-oblivious routing is the well-studied low-diameter decomposition or LDD [Awe85, AP90, Bar96]. Informally, an LDD decomposes a graph into disjoint node partitions such that each pair of close nodes has a large probability of ending up in the same part. A formal definition follows.

Definition 3.4 (Low-diameter decomposition). For a weighted graph $G = (V, E)$, a low-diameter decomposition (LDD) $\mathcal{P}$ of radius ρ and quality $\alpha \geq 1$ is a probability distribution over node disjoint partitions of V into (connected) components $S_1 \subseteq V, \ldots, S_k \subseteq V$ along with centers $c_1 \in S_1, \ldots, c_k \in S_k$ such that:

1. For each i, the center c_i is within distance ρ of every other node in the induced subgraph $G[S_i]$, w.h.p.

2. For every two vertices $x, y \in V$, if d is the distance between them in the original graph, then the probability that they do not belong to the same part $S_i \supseteq \{u, v\}$ is at most $\alpha \cdot \frac{d}{\rho}$.

Our ℓ_1-oblivious routing algorithm based on LDDs is given in Algorithm 1.

Algorithm 1: ℓ_1-oblivious routing

1. Let $\alpha := \exp(O(\sqrt{\log n \cdot \log \log n}))$ (representing the LDD quality).
2. Let $\rho := \exp(O(\log n \cdot \log \log n)^{3/4})$ (representing the LDD radius).
3. Let $d_0 \in \mathbb{R}^V$ by any demand vector.
4. For $i = 1, 2, \ldots, i_{\max} := O(\log n)^{1/4}$ repeat the following:
   (a) Sample $g := O(\log n) \cdot \frac{\rho}{\alpha}$ LDDs $\mathcal{P}_1^i, \mathcal{P}_2^i, \ldots, \mathcal{P}_g^i$ with radius ρ^i and quality α.
   (b) A node v sends $\frac{1}{g} d_{i-1}(v)$ to the center of its component in each $\mathcal{P}_j^i$ along any path of length at most ρ^i, for each $j \in \{1, \ldots, g\}$. This constructs the next-step demand d_i.
5. Compute an arbitrary spanning tree T of G and choose an arbitrary root r. Each node v sends its remaining flow $d_{i_{\max}}(v)$ to r along T.

Choosing the constants. Each hidden constant in the O-notation of Algorithm 1 can be replaced by a universal constant. Specifically, the O-constant in the definition of α is inherited from prior work [HL18] and explained in Lemma 5.1. Constants in the definition of ρ and $i_{\max}$ are arbitrary as long as $\rho^{i_{\max}} \geq n^{C+1}$, where n^C is the largest edge weight in the graph, with $C = O(1)$ as we assumed they are polynomially bounded. The constant in the definition of g makes the algorithm succeed with high probability, hence making it a sufficiently large constant drives the success probability to at least $1 - n^{-C'}$ for any chosen $C' = O(1)$.

Theorem 3.5. *With high probability, Algorithm 1 produces an $\exp(O(\log n \cdot \log \log n)^{3/4})$-approximate ℓ_1-oblivious routing.*

Remark. If one would use (optimal) LDDs of quality $O(\log n)$, the construction described would yield an $\exp(O(\sqrt{\log n \cdot \log \log n}))$-approximate ℓ_1-oblivious routing.

A guided tour of the analysis. We define a *pair-demand* $d_{s,t} \in \mathbb{R}^V$ as $d_{s,t}(x) := \mathbb{1}[x = s] - \mathbb{1}[x = t]$, i.e., requesting a unit flow from s to t. Due to linearity of our routing, it is sufficient to prove that the routing offers a good approximation only with respect to all pair demands $d_{s,t}$ in order to prove

it does the same for all (non-pair) demands. Therefore, the assumption that d is a pair-demand is without loss of generality. This greatly simplifies the analysis.

We analyze how the optimal solution changes when routed along a single LDD $\mathcal{P}$ (of quality approximately $2^{\sqrt{\log n}}$). Since $d = d_{s,t}$ the optimal solution $\|d\|_{\mathrm{OPT}}$ is initially equal to the distance between between s and t, namely $\|d\|_{\mathrm{OPT}} = \ell := \mathrm{dist}_G(s,t)$. After routing along a single LDD with radius ρ and routing each $d(x)$ to the center of the component of x, we pay a cost of at most 2ρ to route the demand and obtain a new (residual) demand $d' \in \mathbb{R}^V$ (which is supported at centers of components of $\mathcal{P}$). Note that if s and t are in the same component of $\mathcal{P}$, then $d' = \vec{0}$; if they are in different components we have $\|d'\|_{\mathrm{OPT}} \leq \|d\|_{\mathrm{OPT}} + 2\rho$ since s and t are both moved by distance at most ρ. However, in expectation, the increase is $\mathbb{E}[\|d'\|_{\mathrm{OPT}} - \|d\|_{\mathrm{OPT}}] \leq \Pr[\text{s, t in different components}] \cdot O(\rho) = 2^{\sqrt{\log n}} \cdot \frac{\ell}{\rho} \cdot O(\rho) \leq 2^{\sqrt{\log n}} \cdot \|d\|_{\mathrm{OPT}}$. In other words, the optimal solution increases in expectation by a manageable $2^{\sqrt{\log n}}$ factor. A naive way to use this result would be to a single hierarchy of LDD decompositions (i.e., find an LDD of radius $\rho = 2^{(\log n)^{3/4}}$, contract, and repeat). We showed this loses a $2^{\sqrt{\log n}}$ multiplicative factor in the value of the optimal solution in each level and, if done for $(\log n)^{1/4}$ levels, ultimately loses a $2^{(\log n)^{3/4}} = n^{o(1)}$ factor. This constructs an $n^{o(1)}$-approximate routing for any demand *in expectation*. It essentially corresponds to a low-quality tree embedding—an analogous (but better!) routing with a $O(\log n)$-approximation guarantee in expectation would be to simply sample an FRT tree [FRT04].

This, however, completely ignores the issue of concentration—we require our single constructed routing to be good with respect to *all demands*, and not just a single demand in expectation. This can typically be done by repeating the same process a sufficient number of times and taking the average until it works with high probability. For example, would need to repeat the above LDD-hierarchy routing at least $\mathrm{poly}(n)$ times until the averaged-out process succeeds for *each demand*. We address this by achieving concentration for each level of the LDD hierarchy. Most of the conceptual heavy-lifting comes from proving that Algorithm 1 works for all demands with high probability, since we have showed that designing a demand that only works in expectation is straightforward. To this end, our idea is to gradually increase the LDD radii as $\rho := 2^{(\log n)^{3/4}}, \rho^2, \rho^3, \ldots, \rho^{i_{\max}} = \mathrm{poly}(n)$ and showing concentration in-between each step. As we shown before, the optimal solution *in expectation* increases in each step by a manageable $2^{\sqrt{\log n}}$ factor, hence the total blow-up after $i_{\max} \leq (\log n)^{1/4}$ step is still $n^{o(1)}$, while the LDDs start consuming the entire graph, implying we are done (it is not hard to see why). To maintain concentration, in each step we sample $\tilde{O}(\rho) \approx 2^{(\log n)^{3/4}}$ LDDs to guarantee the optimal solution does not blow up by more than a $n^{o(1)}$ factor in each step. The crux of the analysis is in showing this averaging will keep the optimal solution small in every step and with respect to every demand.

It is fairly straightforward to show concentration when $i = 1$: the optimal solution is good in expectation and increases by at most an additive ρ factor, hence repeating it for $\tilde{O}(\rho)$ steps guarantees the result with high probability—this is a standard Chernoff bound result, we need to take the average over $\tilde{O}(M/\mu)$ independent random variables that have expectation μ and take up values in the range $[0, M]$ with probability 1. The issue, however, arises when $i > 1$: the optimal solution can grow by a factor of $\rho^i \gg \|d\|_{\mathrm{OPT}}$. Naively, this tells us we need to repeat the process for $\rho^i / \|d\|_{\mathrm{OPT}}$ times, a value which can be often as large as ρ^i. The trick, however, is to "artificially" increase the value of the optimal solution after routing it along an LDD of large radius. Specifically, after routing the demand along an LDD of radius ρ, we will increase the value of the optimal solution by an additive ρ. This does not influence the *newly-increased optimal solution* significantly—the expectation even

10

remains the same up to constant factors. Notably, this increase helps to prove the newly-increased solution concentrates: in step i, the optimal solution increases by an additive ρ^i factor, but is of size at least ρ^{i-1} after the previous step, hence repeating it $\tilde{O}(\rho^i/\rho^{i-1}) = n^{o(1)}$ times is sufficient to prove concentration (i.e., the newly-increased optimal solution does not blow up with respect to all demands). In the following formal proof, we simplify much of this conceptually complicated reasoning by introducing a potential $\phi_i := \|d_i\|_{\mathrm{OPT}} + \|d_i\| \cdot \rho^i$ which intuitively corresponds to the newly-increased optimal solution.

3.3 Proving the approximation guarantees of Algorithm 1

This section is dedicated to proving Theorem 3.5. The main technical insight that greatly simplifies the analysis is the following definition of a potential, which accounts for both the value of the optimal solution and the remaining ℓ_1 mass into account. Note that the subscript i corresponds to the ith step of Algorithm 1.

Definition 3.6 (Potential). $\Phi_i := \|d_i\|_{\mathrm{OPT}} + \|d_i\|_1 \cdot \rho^i$.

Our ultimate goal is to prove the potential increases only by a $n^{o(1)}$ factor over all $i_{\max}$ many steps; this can be shown to directly imply Theorem 3.5. We prove this in several steps. First, we show that in each step i the potential increases only by a multiplicative $O(\alpha) = n^{o(1)}$, but only *in expectation* and when the demand d_{i-1} is fixed to be a *pair-demand* $d_{i-1} = d_{s,t}$. A **pair-demand** $d_{s,t} \in \mathbb{R}^V$ is a demand of the form $d_{s,t}(x) := \mathbb{1}[x = s] - \mathbb{1}[x = t]$, i.e., routing a unit flow from s to t. Second, we show the same claim with high probability instead of in expectation. Third, we show the claim for all demands (not only pair-demands) with high probability. Together, they imply the result.

We start by showing the result in expectation and for pair-demands.

Lemma 3.7 (Expectation analysis). *For any $1 \le i \le i_{\max}$ and any two nodes $s, t \in V$ the following holds. Fix d_{i-1} to be a pair-demand $d_{s,t}$ (with potential $\Phi_{i-1} = d_G(s,t) + 2\rho^{i-1}$) and run the ith step of Algorithm 1. This induces a new random variable Φ_i. We always have that $\mathbb{E}[\max\{\Phi_i, \Phi_{i-1}\}] \le 6\alpha \cdot \Phi_{i-1}$ over the random choices of the ith step.*

Proof. Suppose that the (weighted) distance in G between s and t is ℓ. Since we fixed $d_{i-1} = d_{s,t}$, then $\Phi_{i-1} = \ell + 2\rho^{i-1} \ge \ell$.

We first analyze a *single* randomly sampled LDD with potential Φ_i^{single}. We define a random variable $M_i^{\mathrm{single}} := \max\{\Phi_i^{\mathrm{single}}, \Phi_{i-1}\}$ which tracks the (single) potential without ever decreasing it.

If both endpoints s, t are in the same component, this demand cancels out and $\Phi_i^{\mathrm{single}} = 0$, which in turn gives us $M_i^{\mathrm{single}} = \Phi_{i-1}$. On the other hand, if they end up in separate components (which happens with probability at most $\min\{1, \alpha \cdot \frac{\ell}{\rho^i}\}$), we route them to separate component centers (that are at most ρ^i away), which gives the following bound on the potential

$$\Phi_i^{\mathrm{single}} = \|d_i\|_{\mathrm{OPT}} + \|d_i\|_1 \cdot \rho^i \le (\ell + 2\rho^i) + 2 \cdot \rho^i = \ell + 4 \cdot \rho^i.$$

Note that the expectation does not change whether we take a single LDD or an average over g LDDs, hence we can drop the superscript on $\Phi_i^{\mathrm{single}}, M_i^{\mathrm{single}}$ and just write Φ_i and M_i. In expectation, we get:

$$\mathbb{E}[M_i] \le \Phi_{i-1} + \min\left\{\alpha \cdot \frac{\ell}{\rho^i}, 1\right\} \cdot (\ell + 4\rho^i).$$

11

We now consider two cases. First, if $\ell \leq \rho^i$, we get

$$\mathbb{E}[M_i] \leq \Phi_{i-1} + \alpha \cdot \frac{\ell}{\rho^i} \cdot 5\rho^i = \Phi_{i-1} + 5\alpha \cdot \ell \leq \Phi_{i-1} + 5\alpha \cdot \Phi_{i-1} \leq 6\alpha \cdot \Phi_{i-1}.$$

Otherwise, if $\ell > \rho^i$, we get

$$\mathbb{E}[M_i] \leq \Phi_{i-1} + \ell + 4\rho^i \leq \Phi_{i-1} + 5\ell \leq 6\Phi_{i-1}.$$

Therefore, it always holds that $\mathbb{E}[\Phi_i] \leq \mathbb{E}[M_i] \leq 6\alpha \cdot \Phi_{i-1}$. $\qquad\square$

Next, in Lemma 3.9, we show that the potential does not blow up significantly *with high probability*, instead of just being bounded in expectation. For this we use the following standard Chernoff bound:

Lemma 3.8 (Chernoff Bound). *Let $S = \sum_i X_i$ be a sum of non-negative independent random variables upper bounded by Z, i.e., $0 \leq X_i \leq Z$ with probability 1. For any $t \geq 2\mathbb{E}[S]$ it holds that $\Pr[S \geq t] \leq \exp(-\frac{1}{3} \cdot t/Z)$.*

Lemma 3.9 (Concentration analysis). *For any $1 \leq i \leq i_{\max}$ and any two nodes $s, t \in V$ the following holds. Fix d_{i-1} to be a pair-demand $d_{s,t}$ (with potential $\Phi_{i-1} = d_G(s,t) + 2\rho^{i-1}$) and run the ith step of Algorithm 1. This induces a new random variable Φ_i. With high probability, $\Phi_i \leq 13\alpha \cdot \Phi_{i-1}$ over the random choices of the ith step.*

Proof. We first analyze a *single* randomly sampled LDD with potential Φ_i^{single}. We define a random variable $M_i^{\text{single}} := \max\{\Phi_i^{\text{single}}, \Phi_{i-1}\}$ which tracks the (single) potential without decreasing it.

Let $\Delta_i^{\text{single}} := M_i^{\text{single}} - \Phi_{i-1} = \max\{0, \Phi_i^{\text{single}} - \Phi_{i-1}\}$ be the random variable denoting the change in potential between steps $i-1 \to i$ after sampling a single LDD (or 0 if the change is negative).

Naturally, the final difference in potentials is $\Phi_i - \Phi_{i-1}$ obtained by taking the average of g IID samples $\Delta_{i,1}, \Delta_{i,2}, \ldots, \Delta_{i,g} \sim \Delta_i^{\text{single}}$, i.e., $\Phi_i - \Phi_{i-1} \leq \frac{1}{g}\sum_{j=1}^{g} \Delta_{i,j}$.

We show that $\Delta_i^{\text{single}} \leq O(\rho^i)$ with probability 1. As before, d_{i-1} is some pair-demand $d_{s,t}$ for some endpoints s, t at distance ℓ apart. In a (single) randomly sampled LDD, if both endpoints s, t are in the same component, then the demand cancels out (i.e., $\Phi_i^{\text{single}} = 0$), giving us $\Delta_i^{\text{single}} = 0$. On the other hand, if they end up in separate components, we route them to separate component centers (that are at most ρ^i away), making

$$\begin{aligned}
\Phi_i^{\text{single}} &= \|d_i\|_{\text{OPT}} + \|d_i\|_1 \cdot \rho^i \leq (\ell + 2\rho^i) + 2 \cdot \rho^i \\
&= (\ell + 2\rho^{i-1}) + 4\rho^i - 2\rho^{i-1} \leq \Phi_{i-1} + 4\rho^i.
\end{aligned}$$

In other words, $\Delta_i^{\text{single}} \leq 4\rho^i$ with probability 1, as required.

Using the Chernoff bound (Lemma 3.8) and applying Lemma 3.7, we have that

$$\Pr\left[\sum_{j=1}^{g} \frac{1}{g}\Delta_{i,j} \geq 12\alpha \cdot \Phi_{i-1}\right] \leq \exp(-\frac{1}{3} \cdot 12\alpha \cdot \Phi_{i-1}/Z),$$

12

where $Z = 4\rho^i/g$, since $\frac{1}{g}\Delta_{i,j} \sim \frac{1}{g}\Delta_i^{\text{single}} \in [0, \frac{1}{g}4\rho^i] = [0, Z]$ with probability 1. Using this and the fact that $\Phi_{i-1} \geq \rho^{i-1}$ we have that

$$\Pr[\Phi_i - \Phi_{i-1} \geq 12\alpha \cdot \Phi_{i-1}] \leq \Pr\left[\sum_{j=1}^{g} \frac{1}{g}\Delta_{i,j} \geq 12\alpha \cdot \Phi_{i-1}\right]$$

$$\leq \exp\left(-\frac{1}{3} \cdot 12\alpha\Phi_{i-1}\frac{g}{4\rho^i}\right)$$

$$\leq \exp\left(-4\alpha\frac{\rho^{i-1} \cdot g}{\rho^i}\right)$$

$$= \exp\left(-\alpha\frac{g}{\rho}\right)$$

$$= \exp(-O(\log n)) = n^{-O(1)}.$$

Therefore, $\Phi_i \leq \Phi_{i-1} + 12\alpha \cdot \Phi_{i-1} = 13\alpha \cdot \Phi_{i-1}$ with high probability. $\square$

Combining the expectation and concentration analysis, we obtain that the potential does not blow up significantly over a single step.

Lemma 3.10 (Single-step analysis). *For any $1 \leq i \leq i_{\max}$ and any demand $d \in \mathbb{R}^V$ with $\sum_{v \in V} d_v = 0$ the following holds. Fix $d_{i-1} := d$ (with a fixed potential Φ_{i-1}) and run the ith step of Algorithm 1. This induces a new random variable Φ_i. With high probability, $\Phi_i \leq 13\alpha \cdot \Phi_{i-1}$ over the random choices of the ith step.*

Proof. We first note that the routing defined via Algorithm 1 is linear, i.e., there exists a linear operator $L = L_i$ (L depends on i, but we drop the subscript for ease of notation) such that $d_i = Ld_{i-1}$.

Using Lemma 3.9, the claim of this lemma holds for all pair-demands $d_{s,t}$ w.h.p. In other words,

$$\|Ld_{s,t}\|_{\text{OPT}} + \|Ld_{s,t}\|_1 \cdot \rho^i \leq 13\alpha \cdot [\|d_{s,t}\|_{\text{OPT}} + \|d_{s,t}\|_1 \cdot \rho^{i-1}].$$

Since there are only $|V|^2$ many pair-demands, it also holds for all demand-pairs w.h.p. We now show that it holds for an arbitrary demand d_{i-1}.

Consider the optimal flow $f \in \mathbb{R}^{\vec{E}}$ that routes d_{i-1} (for an edge $e \in E$, the sign of $f(e)$ denotes the direction of the flow along e). Any flow satisfying d_{i-1} can be decomposed into a positive combination of J paths $\{p_j\}_{j=1}^J$ where (1) each path p_j is *unit* in the sense that $p_j(e) \in \{0, 1, -1\}$ for all $e \in E$ (the sign depends on the arbitrary orientation of e), and (2) each path p_j starts at a node s where $d_{i-1}(s) > 0$ and ends in a node t where $d_{i-1}(t) < 0$. Therefore, we have that $f = \sum_{j=1}^J \beta_j p_j$ where $\beta_j > 0$ and $p_j \in \mathbb{R}^{\vec{E}}$ is a unit path. Since p_j is a unit path, we have that it is a feasible transshipment solution for some pair-demand q_j (namely, the pair-demand $q_j := d_{s,t}$ where s and t are the endpoints of p_j), hence $\|q_j\|_{\text{OPT}} \leq W(p_j)$. Due to the flow decomposition, $\|d_{i-1}\|_{\text{OPT}} = W(f) = \sum_{j=1}^J \beta_j W(p_j) \geq \sum_{j=1}^J \beta_j \|q_j\|_{\text{OPT}}$. Furthermore, due to the restriction on the endpoints of the paths we also have that $\|d_{i-1}\|_1 = \sum_{j=1}^J \beta_j \|q_j\|_1$.

Using this decomposition into paths, we have that

$$\Phi_{i-1} = \|d_{i-1}\|_{\text{OPT}} + \|d_{i-1}\|_1 \cdot \rho^{i-1}$$

$$\geq \sum_{j=1}^{J} \left(\beta_j \|q_j\|_{\text{OPT}} + \beta_j \rho^{i-1} \|q_j\|_1 \right)$$

$$\geq \frac{1}{13\alpha} \cdot \sum_{j=1}^{J} \left(\beta_j \|Lq_j\|_{\text{OPT}} + \beta_j \rho^i \|Lq_j\|_1 \right)$$

$$\geq \frac{1}{13\alpha} \cdot \left[\left\| L \sum_{j=1}^{J} \beta_j q_j \right\|_{\text{OPT}} + \rho^i \left\| L \sum_{j=1}^{J} \beta_j q_j \right\|_1 \right] \tag{3.1}$$

$$= \frac{1}{13\alpha} \cdot \left[\|d_i\|_{\text{OPT}} + \rho^i \|d_i\|_1 \right] = \frac{1}{13\alpha} \cdot \Phi_i.$$

Note that in (3.1) we used $\|a + b\| \leq \|a\| + \|b\|$ for $\|\cdot\|_{\text{OPT}}$ and $\|\cdot\|_1$. Rewriting, we have that $\Phi_i \leq 13\alpha \cdot \Phi_{i-1}$ as required, if one assumes the claim for all pair demand, with high probability. $\square$

Finally, with all intermediate steps in place, we prove the main result of this section.

Proof of Theorem 3.5. Fix a demand $d_0 := d$ and the value of the optimal solution $\text{OPT} := \|d\|_{\text{OPT}}$. Using Lemma 3.10, we have that w.h.p.

$$\Phi_0 = O(\text{OPT}),$$

$$\Phi_i \leq (13\alpha)^i \cdot \text{OPT}.$$

In the ith step, for $i \in \{1, \ldots, i_{\max}\}$, let f_i denote the flow which is constructed via Algorithm 1. The total movement $W(f_i)$ can be bounded in the following way:

$$W(f_i) \leq \rho^i \|d_{i-1}\|_1 \leq \rho \cdot \Phi_{i-1} \leq \rho \cdot (13\alpha)^{i-1} \cdot O(\text{OPT}) \leq \exp\left(O(\log n \cdot \log\log n)^{3/4} \right) \cdot \text{OPT}.$$

Summing up over all steps $i \in \{1, \ldots, i_{\max}\}$, the total movement of $f = \sum_{i=1}^{i_{\max}} f_i$ is at most

$$W(f) \leq i_{\max} \cdot \exp\left(O(\log n \cdot \log\log n)^{3/4} \right) \cdot \text{OPT} = \exp\left(O(\log n \cdot \log\log n)^{3/4} \right) \cdot \text{OPT}.$$

We now analyze the final aggregation along an arbitrary spanning tree T towards an arbitrarily chosen root r. The total movement of the final aggregation is $n^{C+1} \cdot \|d_{i_{\max}}\|_1$, where n^C is the largest edge weight (by assumption, $C > 0$ is a constant).

Lemma 3.10 implies that $\rho^{i_{\max}} \cdot \|d_{i_{\max}}\|_1 \leq \Phi_{i_{\max}} \leq (13\alpha)^{i_{\max}} \cdot O(\text{OPT})$. Remembering that $\rho^{i_{\max}} \geq n^{C+1}$, we have that the final aggregation contributes $n^{C+1} \cdot \|d_{i_{\max}}\|_1 \leq \exp\left(O(\log n \cdot \log\log n)^{3/4} \right) \cdot \text{OPT}$ to the cost. Adding all contributions together, we conclude that the ℓ_1-oblivious routing yields an $\exp\left(O(\log n \cdot \log\log n)^{3/4} \right)$ approximation. $\square$

3.4 Linear-algebraic interpretation of the routing

In this section, we develop an algebraic interpretation of the routing matrix R constructed by Algorithm 1 via simpler (LDD-induced) routing matrices.

Suppose that $G = (V, E)$ is an undirected graph. Edges are oriented arbitrarily by specifying a linear operator (i.e., matrix) $B \in \mathbb{R}^{V \times \vec{E}}$ which maps a flow $f \in \mathbb{R}^{\vec{E}}$ to the demand $d = Bf$ that f routes (defined in Section 2). On the other hand, an oblivious routing is a linear operator (i.e., matrix) $R \in \mathbb{R}^{\vec{E} \times V}$ that maps a demand $d \in \mathbb{R}^V$ to a flow $f = Rd \in \mathbb{R}^{\vec{E}}$.

Given an LDD $\mathcal{P}$, we define the oblivious routing with respect to $\mathcal{P}$ as follows. Let the components (partitions) of $\mathcal{P}$ be $S_1 \subseteq V, \ldots, S_k \subseteq V$ with corresponding centers $\{c_i \in S_i\}_i$, and let $\{T_i\}_i$ be the shortest path trees of S_i rooted at c_i (with all edges oriented from the root outwards). We define $R(\mathcal{P}) \in \mathbb{R}^{\vec{E} \times V}$ to be the routing matrix that routes the demand along T_i towards the root. More precisely, $R(\mathcal{P}) \cdot d$ is the flow that, for each node $s \in S_i$, routes $d(s)$ amount of flow via the leaf-to-root path of T_i (note that the output of $R(\mathcal{P}) \cdot d$ needs to match the orientation determined by the matrix B, hence some components might need to have their sign flipped).

With respect to Algorithm 1, let R denote the oblivious routing performed by the entire procedure, and R_i be the routing that is performed in the ith step, i.e., R_i is performed on d_{i-1} in order to produce the next-step demand d_i. We specifically define $R_{i_{\max}+1}$ to be routing in the final aggregation step, i.e., routing along the spanning tree T in Step 5 of the algorithm.

By definition of R_i, the flow routed in the ith step is $R_i d_{i-1}$. This routes the demand $BR_i d_{i-1}$, hence the residual demand that needs to be routed is $d_i = d_{i-1} - BR_i d_{i-1} = (I - BR_i)d_{i-1}$. On the other hand, the routing R_i is constructed by averaging out the routings with respect to g LDDs $\mathcal{P}_1^i, \ldots, \mathcal{P}_g^i$. In other words, $R_i = \frac{1}{g}R(\mathcal{P}_1^i) + \frac{1}{g}R(\mathcal{P}_2^i) + \ldots + \frac{1}{g}R(\mathcal{P}_g^i)$.

Lemma 3.11. *It holds that*

$$R = \sum_{i=1}^{i_{\max}+1} R_i \cdot (I - BR_{i-1}) \cdot (I - BR_{i-2}) \cdot \ldots \cdot (I - BR_1),$$

where

$$R_i = \frac{1}{g}R(\mathcal{P}_1^i) + \frac{1}{g}R(\mathcal{P}_2^i) + \ldots + \frac{1}{g}R(\mathcal{P}_g^i).$$

Proof. We already argued the second identity.

Fix some demand $d_0 \leftarrow d$. Let f be the total flow that is routed over all steps when routing d, and f_i be the flow that is routed during the ith step of Algorithm 1. Specially, let $f_{i_{\max}}$ be the flow routed in the final aggregation step (Step 5). By definition, we have that $f = \sum_{i=1}^{i_{\max}+1} f_i$. Furthermore, by definition of R_i we have that $f_i = R_i d_{i-1}$. Finally, as argued before, we have that

$d_i = (I - BR_i)d_{i-1}$. Combining all of these, we have that

$$Rd = f = \sum_{i=1}^{i_{\max}+1} f_i = \sum_{i=1}^{i_{\max}+1} R_i d_{i-1}$$

$$= \sum_{i=1}^{i_{\max}+1} R_i (I - BR_{i-1}) d_{i-2}$$

$$= \sum_{i=1}^{i_{\max}+1} R_i (I - BR_{i-1}) \dots (I - BR_1) d_0.$$

In other words, $R = \sum_{i=1}^{i_{\max}+1} R_i (I - BR_{i-1}) \dots (I - BR_1)$, as required. $\qquad\square$

4   The Distributed Minor-Aggregation Model

We contribute to the low-congestion shortcut framework by giving a simple yet powerful interface called the *Distributed Minor-Aggregation model* which makes the recent advances in theoretical distributed computing such as universal optimality [HWZ21] and oblivious shortcut constructions [HWZ21, GHZ21] more accessible. The Distributed Minor-Aggregation Model offers a high-level interface which can be used to succinctly describe many interesting distributed algorithms. In spite of this expressiveness, any algorithm in the Minor-Aggregation model can be efficiently simulated in the standard CONGEST model. To demonstrate this, we describe a simple universally-optimal algorithm for the minimum spanning tree (MST) in Example 4.4.

Definition 4.1 (Distributed Minor-Aggregation Model)**.** We are given an undirected graph $G = (V, E)$. Both nodes and edges are individual computational units (i.e., have their own processor and private memory). All computational units wake up at the same time and start communicating in synchronous rounds. Arbitrary local computation is allowed between (each step of) each round. Initially, nodes only know their unique $\tilde{O}(1)$-bit ID and edges know the IDs of their endpoint nodes. Each round of communication consists of the following three steps (in that order).

- **Contraction step.** Each edge e chooses a value $c_e = \{\bot, \top\}$. This defines a new *minor network* $G' = (V', E')$ constructed as $G' = G/\{e : c_e = \top\}$, i.e., we contract all edges with $c_e = \top$. Vertices V' of G' are called supernodes, and we identify supernodes with the subset of nodes V it consists of, i.e., if $s \in V'$ then $s \subseteq V$.

- **Consensus step.** Each individual node $v \in V$ chooses a $\tilde{O}(1)$-bit value x_v. For each supernode $s \in V'$ we define $y_s := \bigoplus_{v \in s} x_v$, where $\bigoplus$ is some pre-defined aggregation operator. All nodes $v \in s$ learn y_s.

- **Aggregation step.** Each edge $e \in E'$ connecting supernodes $a \in V'$ and $b \in V'$ learns y_a and y_b, and chooses two $\tilde{O}(1)$-bit values $z_{e,a}, z_{e,b}$ (i.e., one value for each endpoint). Finally, (every node of) each supernode $s \in V'$ learns the aggregate of its incident edges in E', i.e., $\bigotimes_{e \in \text{incidentEdges}(s)} z_{e,s}$ where $\bigotimes$ is some pre-defined aggregation operator. All nodes $v \in s$ learn the same aggregate value (this might be relevant if the aggregate is non-unique).

Distributing the input and output. Distributed Minor-Aggregation model is simply a communication models upon which one can run various algorithms. The goal is typically to consider a problem like transshipment, SSSP or MST, and design a Minor-Aggregation algorithm that provably terminates with a correct answer in the smallest possible number of rounds. At start, each

node/edge receives problem-specific input. This input is distributed among the network in the following way. Such problems are performed on a weighted graph, hence the weight are distributed in a way that each edge e initially knows its weight $w(e)$. Note: the weight does not influence the communication (i.e., always takes 1 round regardless of the weights). Furthermore, for transshipment, each node v additionally knows its demand value $d(v)$. For SSSP, all nodes and edges additionally know the ID of the source node. Similarly, upon termination, the output is also required to be *stored distributedly.* For example, for the minimum spanning tree (MST) problem, at termination, each edge should know whether it is a part of the MST or not. For SSSP, upon termination, each node should know its distance to the source and each edge whether it is a part of the SSSP tree. For transshipment, upon termination, each edge e should know its part of the flow $f(e)$ and each node v should know its potential $\phi(v)$.

Polylogarithmic factors. We note that the above definition extensively uses the $\tilde{O}$-notation, thereby ignoring polylogarithmic factors (unlike CONGEST which only ignores constant factors). This is due to the fact that the goal of the Minor-Aggregation model is to illuminate the influence of polynomial factors on the runtime of distributed algorithms, while at the same time keeping the framework as simple as possible. Ignoring poly$(\log n)$ factors is the standard in the literature for distributed global problems like MST or SSSP as the current state-of-the-art is also mostly focused on optimizing polynomial factors and the algorithmic descriptions of global distributed problems get significantly more complicated when one starts optimizing logarithmic factors.

Simulation in CONGEST. While this is not obvious, an algorithm in the Minor-Aggregation model can be efficiently simulated in the standard CONGEST model if one can efficiently solve the part-wise aggregation (PA) problem. Combining this with the very recent work [GHR21] which solves PA in ShortcutQuality$(G) \cdot n^{o(1)}$ rounds (see Theorem 2.2), any τ-round Minor-Aggregation algorithm can be compiled into a $\tau \cdot \text{poly}(\text{ShortcutQuality}(G)) \cdot n^{o(1)}$-round CONGEST algorithm. Moreover, on many graph classes this result can be improved down to $\tau \cdot \tilde{O}(\text{ShortcutQuality}(G))$ (see Theorem 2.4) and potential future improvements in hop-constrained oblivious routing constructions might make lead to unconditional $\tau \cdot \tilde{O}(\text{ShortcutQuality}(G))$ simulations (which would be near-optimal). The proof of the following simulation theorem is deferred to Appendix A. It implies many useful concrete bounds for specific graph classes (see Theorem 2.4 for a list).

Theorem 4.2. *Any τ-round Minor-Aggregation algorithm on G can be simulated in the (randomized) CONGEST model on G in $\tau \cdot \text{poly}(\text{ShortcutQuality}(G)) \cdot n^{o(1)}$ rounds. More generally, if one can solve PA in τ_{PA} rounds, any τ-round Minor-Aggregation algorithm can be simulated in $\tilde{O}(\tau \cdot \tau_{PA})$ (randomized) CONGEST rounds.*

Operating on minors. A particularly appealing feature of the Minor-Aggregation model is that any algorithm can be run on a minor of the original communication network in a black-box way. The framework allows the algorithm to be completely unchanged when ran on a minor of a graph rather than on the original graph. Several aspects of the Minor-Aggregation model make operating on minors possible. For instance, in a notable difference with CONGEST, nodes in the Minor-Aggregation model do not know a list of their neighbors[2]. The following corollary is immediate from the definition.

Corollary 4.3. *Let $G = (V, E)$ be an undirected graph, and let $F \subseteq E$ be a subset of edges. Any τ-round Minor-Aggregation algorithm on a minor $G' = G/F$ of G can be simulated via a τ-round Minor-Aggregation algorithm on G. Initially, each edge $e \in E$ needs to know whether $e \in F$ or*

---

[2]Moreover, it is not hard to see that a model which allows contractions and gives nodes a list of their neighbors cannot be simulated in CONGEST with $o(n)$ round blowup.

not. Upon termination, each node v in G learns all the information that the G'-supernode v was contained in learned.

An example: MST. The following example illustrates the expressive power and efficiency of the Minor-Aggregation model framework. We describe an $O(\log n)$-round Minor-Aggregation algorithm for MST. Combining with the simulation Theorem 4.2 and lower bound Theorem 2.3, this implies a $\tilde{O}(1)$-competitive universally-optimal distributed algorithm for MST in CONGEST. We note that, on a weighted planar graph, this algorithm provably completes in HopDiameter$(G) \cdot n^{o(1)}$ CONGEST rounds; similar results exist for many other graphs, see Theorem 2.4.

Example 4.4 (MST)**.** The minimum spanning tree of a weighted graph G can be computed via a $O(\log n)$-round Minor-Aggregation algorithm on G.

Proof. We can directly implement Boruvka's algorithm [NMN01]. In each round, every node v finds the minimum weight m_v of an edge incident to it. This is clearly doable in the the Minor-Aggregation model: no contractions and no consensus are necessary; each edge simply reports its weight to both of its endpoints and the aggregation operator $\bigotimes$ is simply the min operation.

We say that an edge $e = \{u, v\}$ is *marked* if its weight is equal to m_u or to m_v. Each edge can identify whether it is marked in a single aggregation step (by choosing $y_s := m_s$ for each node s). Finally, we add all marked edges to the MST and contract them. We repeat the algorithm on the contracted graph (Corollary 4.3). After $O(\log n)$ iterations, the graph shrinks to a single node and each edge knows whether it is in the MST or not. $\qquad\square$

We obtain a $\tilde{O}(\text{HopDiameter}(G))$-round CONGEST algorithm for weighted planar graphs by combining the above $O(\log n)$-round Minor-Aggregation algorithm with the CONGEST simulation Theorem 4.2, which stipulates that algorithms on planar (or more generally, minor-free) networks can be simulated with a $\tilde{O}(\text{HopDiameter}(G))$ overhead.

5 Distributed Computation of the ℓ_1-Oblivious Routing

In this section, we show how to distributedly implement Algorithm 1 using $n^{o(1)}$ Minor-Aggregation rounds. To this end, we first build a few necessary building blocks in which will simplify the distributed implementation of our algorithms in the remaining sections. We then proceed to prove Theorem 1.5 in Section 5.2.

5.1 Preliminary: distributed storage and basic linear algebra operations

It is often easier to describe algorithms a purely linear-algebraic setting without going into the low-level details of how and where to store individual values required to specify distributed algorithms. In order to streamline the description of (linear-algebraic) distributed algorithms we first define the notion of distributed storage for the Minor-Aggregation as follows.

1. We distributedly store a **node vector** $x \in \mathbb{R}^V$ by storing the value x_v in the node $v \in V$. Similarly, given an **edge vector** $x \in \mathbb{R}^{\vec{E}}$ we store the value x_e in the edge e (we remind the readers that edges are computational units in the Minor-Aggregation model).

2. We distributedly store a **spanning subgraph** $H \subseteq G$ (where G is the communication network) by storing distributedly storing the indicator edge vector $x_e = \mathbb{1}[e \in E(H)]$.

3. As explained in Section 4, the input to transhipment, namely the weights (edge vector) and the demand d (node vector), are distributedly stored. Each node v knows its part of the demand $d(v)$. Upon output, we require the flow f (edge vector) and potential ϕ (node vector) to be distributedly stored. The specification is analogous for SSSP: on input, we distributedly store the edge weights and require that all nodes know the ID of the source s. Upon output, we require the SSSP tree (spanning subgraph) and distances (node vector) to be distributedly stored.

Furthermore, we verify that the following linear-algebraic graph operations can be evaluated quickly. In the following suppose a, b are two distributedly stored node or edge vectors.

1. **Vector addition and component-wise transforms.** Without any extra communication we can compute and distributedly store (1) the sum vector $a + b$, or (2) the vector $(f(a_i))_i$, i.e., the vector a with $f : \mathbb{R} \to \mathbb{R}$ applied component-wise.

2. **Dot products and ℓ_p norms.** With a single Minor-Aggregation round we can compute and broadcast to all nodes the value of (1) the dot product $a^T b$, or (2) ℓ_p-norm of a vector, i.e., $(\sum_i |a_i|^p)^{1/p}$. We show why for the case of dot products. Suppose a, b are node vectors. Each node v evaluates $a_v \cdot b_v$ and stores it as its private input x_v. We contract all edges of the graph and apply the consensus step with the plus-operator, which broadcasts the dot product $\sum_{v \in V} a_v \cdot b_v = a^T b$ to all nodes. For the case of edge vectors, we contract all edges, and perform an aggregation step with the $+$-operator where each edge $e = \{a, b\}$ computes $a_e \cdot b_e$ and sets $z_{e,a} = a_e \cdot b_e$, $z_{e,b} = 0$. This informs all nodes about $a^T b$. Computing ℓ_p norms is analogous.

3. **Multiplication with B and B^T.** With a single Minor-Aggregation round we can compute and distributedly store $B \cdot a$ and $B^T \cdot b$, where $a \in \mathbb{R}^{\vec{E}}$ is an edge vector and $b \in \mathbb{R}^V$ is a node vector. To compute $B \cdot a$, we leave all edges uncontracted and perform an aggregation step with the $+$-operator, where each edge $e = \{u, v\}$ with orientation $\vec{e} = (u, v)$ sets $z_{e,u} = +a_u$ and $z_{e,v} = -a_v$. Upon completion of the step, a node $w \in V$ learns $(B \cdot a)_w$, as required for distributed storage. To compute $B^T \cdot b$, we perform a consensus step without any contracted edges with each node v setting $y_v = x_v = b_v$ as its private input. Each edge $\vec{e} = (u, v)$ learns b_u and b_v and computes $(B^T b)_e = b_u - b_v$, as required.

4. **Multiplication with W.** With a single Minor-Aggregation round we can compute and distributedly store $W \cdot a$, where $a \in \mathbb{R}^{\vec{E}}$ is an edge vector. Since for each edge $e = (u, v)$, a_e and w_e are stored at both vertex u and v, every node $u \in V$ learns $W \cdot a$ for all the edges incident to u.

5.2 Distributed evaluation of R and R^T

In this section we show the following result.

Theorem 1.5. (Distributed oblivious routing evaluation). *Let $G = (V, E)$ be a weighted graph with weights in $[1, n^{O(1)}]$. There exists an $\exp(O(\log n \cdot \log \log n)^{3/4})$-approximate ℓ_1-oblivious routing R such that we can perform the following computations in $\exp(O(\log n \cdot \log \log n)^{3/4})$ rounds of Minor-Aggregation. Given any distributedly stored node vector $d \in \mathbb{R}^V$ and edge vector $c \in \mathbb{R}^V$, we can evaluate and distributedly store $Rd \in \mathbb{R}^{\vec{E}}$ and $R^T c \in \mathbb{R}^V$.*

The rest of the section is dedicated to proving Theorem 1.5. Inspecting Algorithm 1, we require distributed implementations of a few subroutines. First, we require a way to efficiently sample from

the LDD distribution. To this end, we leverage the following theorem from prior work [HL18].

Lemma 5.1. *[LDD sampling [HL18]] Suppose G is a weighted graph G with weights in $[1, n^{O(1)}]$. For any $\rho \geq 1$, there exists a distributed algorithm which samples an LDD from a distribution of radius ρ and quality $\exp(O(\sqrt{\log n \cdot \log \log n}))$ in $\exp(O(\sqrt{\log n \cdot \log \log n}))$ rounds of Minor-Aggregation.*

Upon termination, each node v knows the center node c_i of its LDD component S_i, and v's parent edge in the shortest path tree of $G[S_i]$ that is centered at c_i.

Proof. Deferred to Appendix A. $\qquad\square$

In order to compute the flows obtained by routing the demand d_{i-1} using the shortest path trees in each LDD component, we need to be able to quickly compute subtree sums of rooted trees. This is furnished by the following result from [DG19, Theorem 5.1, full version].

Lemma 5.2 (Subtree sum [DG19]). *Let $G = (V, E)$ be a graph and let $F \subseteq G$ be a collection of node-disjoint rooted trees (i.e., a rooted forest) that are subgraphs of G. Initially, F is stored distributedly (each edge know whether $e \in F$ and its orientation). Furthermore, each node v has a $O(\log n)$-bit private input x_v. There exists a $\tilde{O}(1)$-round Minor-Aggregation algorithm such that, upon termination, each node v learns both the sum of private values of all of its descendents and the sum of private values of all of its ancestors.*

Lemma 5.3. *Suppose an LDD $\mathcal{P}$ over a weighted graph G is distributedly stored (in the sense of Lemma 5.1). There exists a $\tilde{O}(1)$-round Minor-Aggregation algorithm that, given distributedly stored $d \in \mathbb{R}^V$ and $c \in \mathbb{R}^{\vec{E}}$, evaluates and distributedly stores $R(\mathcal{P}) \cdot d \in \mathbb{R}^{\vec{E}}$ and $R(\mathcal{P})^T \cdot c \in \mathbb{R}^V$.*

Proof. We consider the shortest path trees $T_1, \ldots, T_k$ for each component of the LDD. To compute $R(\mathcal{P}) \cdot d$, we need to output $(R(\mathcal{P}) \cdot d)_e = 0$ for all non-tree edges. For each tree-edge edge $e = \{v, u\} \in E(T_i)$ where u is closer to the root of T_i, we output $(R(\mathcal{P}) \cdot d)_e$ to be the total sum of demands $d(w)$ over all w that are descendents of v. This number needs to possibly be negated if $\vec{E}$ contains the edge in the opposite orientation (i.e., if $(u, v) \in \vec{E}$). The correctness of this procedure is immediate. We can directly implement this in $\tilde{O}(1)$ rounds of Minor-Aggregation via Lemma 5.2.

Similarly, to compute $R(\mathcal{P})^T \cdot c$, each node $v \in V(T_i)$ needs to compute the sum of $c(e)$ over all edges e (with some elements possibly negated) on the root-to-v path (in T_i). Again, we can directly implement this in $\tilde{O}(1)$ rounds of Minor-Aggregation via Lemma 5.2. $\qquad\square$

Finally, we are ready to prove the main theorem of this section.

Proof of Theorem 1.5. The $\exp(O(\log n \cdot \log \log n)^{3/4})$-approximation guarantee of the ℓ_1-oblivious routing constructed by Algorithm 1 is ensured by Theorem 3.5.

We examine Algorithm 1 and show we can implement each ingredient required for the procedure. Firstly, we sample and distributedly store all LDDs $\mathcal{P}_j^i$ for $1 \leq i \leq i_{\max}$, $1 \leq j \leq g$. This is provided by Lemma 5.1 in $g \cdot i_{\max} \cdot \exp(\sqrt{\log n \cdot \log \log n}) = \exp(O(\log n \cdot \log \log n)^{3/4})$ rounds of Minor-Aggregation. Furthermore, we compute and store an arbitrary spanning tree T of G. A simple choice is to use the MST, whose construction can be computed in $\tilde{O}(1)$ rounds of Minor-Aggregation, as stipulated by Example 4.4.

Let $R_i = \frac{1}{g}R(\mathcal{P}_1^i) + \frac{1}{g}R(\mathcal{P}_2^i) + \ldots + \frac{1}{g}R(\mathcal{P}_g^i)$ (Lemma 3.11). We note we can evaluate (and distributedly store) $R_i \cdot d$ (which we call *multiplication from right*) and $R_i^T \cdot c$ (called *multiplication from left*) in $\exp(O(\log n \cdot \log\log n)^{3/4})$ rounds for any distributedly stored vectors c, d. We simply evaluate each term in the sum (when multiplied from both left and right), and add them together. This contributes $\tilde{O}(1) \cdot g$ rounds of communication by utilizing Lemma 5.3 and basic operations of Section 5.1. Furthermore, we can evaluate and distributedly store $(I - BR_i) \cdot d$ and $(I - BR_i) \cdot d$ for any vector $d \in \mathbb{R}^V$ in $\tilde{O}(g)$ rounds using the basic operations of Section 5.1. Specially, we define $R_{i_{\max}+1}$ to be the routing with respect to a spanning tree T. Since routing along T can be seen as routing along a simple LDD, multiplication of $R_{i_{\max}+1}$ from left and right in $\tilde{O}(1)$ rounds is furnished by Lemma 5.3.

Finally, we remember from Lemma 3.11 that $R = \sum_{i=1}^{i_{\max}+1} R_i \cdot (I - BR_{i-1}) \cdot (I - BR_{i-2}) \cdot \ldots \cdot (I - BR_1)$. We already showed that we can multiply each factor from both left and right in $\tilde{O}(g)$ steps, hence we can compute Rd and $R^T c$ in $i_{\max}^2 \cdot \tilde{O}(g) = \exp(O(\log n \cdot \log\log n)^{3/4})$, as required. $\qquad\square$

6   Distributed $(1 + \varepsilon)$-Transshipment

In this section, we present our distributed algorithm for approximating the transshipment problem. The result is shown in Theorem 1.3, and for the sake of completeness, we restate it below.

Theorem 1.3. (Distributed transshipment). *Let G be a weighted graph with weights in $[1, n^{O(1)}]$ and let $\varepsilon > 0$ be a parameter. There exists an algorithm that computes $(1 + \varepsilon)$-approximate transshipment on G in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log\log n)^{3/4})$ Minor-Aggregation rounds.*

We follow Sherman's framework [She17b] via the multiplicative weights update (MWU) paradigm [AHK12] based on the construction of efficient ℓ_1-oblivious routing presented in the last section, and then give the round complexity of the implementation in the CONGEST model. Instead of exactly performing Sherman's method [She17b], our algorithm does not apply Bourgain's ℓ_1-embedding [Bou85] when constructing the ℓ_1-oblivious routing operator.

Given an ℓ_1-oblivious routing, we solve transshipment by utilizing its *boosting* property: the ℓ_1-oblivious routing yields an $n^{o(1)}$-approximate solution for transshipment, which we can then *boosting* to an $(1+\varepsilon)$-approximate one using multiplicative weights (or gradient descent). This boosting property was implicitly used in many papers [She17b, BKKL17, Li20, ASZ20], and was explicitly isolated in a recent writeup [Zuz21] that shows any black-box dual approximate solution can be boosted to an $(1 + \varepsilon)$-approximate one. The following lemma gives an end-to-end interface to boosting: given an ℓ_1-oblivious routing, we can construct $(1 + \varepsilon)$-approximate solutions to transshipment.

Lemma 6.1. *Let R be an α-approximate ℓ_1-oblivious routing with respect to a transshipment instance on a weighted graph G. Suppose we can compute matrix-vector products with R and R^T in M Minor-Aggregation rounds. Then we can compute and distributedly store a flow $f \in \mathbb{R}^{\vec{E}}$ and a vector of potentials ϕ in $\tilde{O}(\alpha^2 \varepsilon^{-2} M)$ rounds such that*

1. $\|Wf\|_1 \le (1+\varepsilon)d^\top \phi \le (1+\varepsilon)\|d\|_{\mathrm{OPT}}$;

2. $\|Bf - d\|_{\mathrm{OPT}} \le \varepsilon\|d\|_{\mathrm{OPT}}$;

3. $\|W^{-1}B^T\phi\|_\infty \le 1$.

Before proving Lemma 6.1, we give the following lemma which returns the rough flow f and potential ϕ affected by the input parameter t. The algorithm we simulate is summarized in Algorithm 2. In

Algorithm 2, MWU is used to determine if one region is approximately feasible. It returns a flow f or a potential ϕ that satisfy the two conditions in the following lemma.

Lemma 6.2 (Lemma C.2 in [Li20]). *Consider a transshipment instance with demand vector d and a parameter $t \geq \|d\|_{\mathrm{OPT}}/2$. Let R be an α-approximate ℓ_1-oblivious routing operator. Suppose we can compute matrix-vector products with R and R^T in M Minor-Aggregation rounds. Then there is an $\tilde{O}(\alpha^2 \varepsilon^{-2} M)$-round Minor-Aggregation algorithm outputs either*

1. *an acyclic flow f satisfying $\|Wf\|_1 \leq t$ and $\|WRBf - WRd\|_1 < \varepsilon t$, or*

2. *a potential ϕ with $d^\top \phi = t$.*

Proof. We can show an algorithm for the above lemma applying the MWU method [AHK12] for (approximately) checking the feasibility of the following region

$$\left\{ y \in \mathbb{R}^{\vec{E}} \,\middle|\, \left\| y^\top WRBW^{-1} \right\|_\infty + \frac{1}{t} y^\top WRd \leq -\varepsilon \text{ and } \|y\|_\infty \leq 1 \right\}. \tag{6.1}$$

Observe that this problem is tightly connected to the dual convex-programming formulation of the transshipment problem. Building upon the presentation of Li [Li20], we give a self-contained procedure summarized in Algorithm 2 for checking the feasibility of the region. The correctness of Algorithm 2 is proved in [Li20]. Furthermore, one can easily check that all the operations required to implement Algorithm 2 in the Minor-Aggregation model are either matrix-vector multiplications or basic operations presented in Section 5.1. $\qquad\square$

Algorithm 2: MWU for transshipment using ℓ_1-oblivious routing

1. Set $\delta \leftarrow \varepsilon/(2\alpha)$.
2. Set $p_0^+(e) \leftarrow 1/(2m)$ and $p_0^-(e) = 1/(2m)$ for all $e \in E$.
3. Set $\phi_0^+(e) = 1$ and $\phi_0^-(e) = 1$ for all $e \in E$.
4. Define $\chi_e := (\mathbb{1}_u - \mathbb{1}_v)$, where $e = (u, v) \in E$, and $\mathbb{1}_u, \mathbb{1}_v$ are indicator vectors.
5. For $t = 1, 2, \ldots, T$ where $T = O(\alpha^2 \varepsilon^{-2} \log m)$:
   (a) If $\left\| WRB \sum_{e \in E} w_e^{-1}(p_{t-1}^+(e)\chi_e - p_{t-1}^-(e)\chi_e) + \frac{1}{t} WRd \right\|_1 \geq \varepsilon$
      i. Set $y(t) \leftarrow -\mathrm{sign}\left(WRB \sum_{e \in E} w_e^{-1}(p_{t-1}^+(e)\chi_e - p_{t-1}^-(e)\chi_e) + \frac{1}{t} WRd \right)$
   (b) Otherwise, set $f \leftarrow -t \sum_e w_e^{-1}(p_e^+ \chi_e - p_e^- \chi_e)$ and output f.
   (c) For each $e \in E$:
      i. Set $\phi_t^+(e) \leftarrow \frac{1}{2}\phi_{t-1}^+(e) \cdot \exp\left(\delta \cdot (y_t^\top WRBw_e^{-1}\chi_e + \frac{1}{t} y_t^\top WRd) \right)$.
      ii. Set $\phi_t^-(e) \leftarrow \frac{1}{2}\phi_{t-1}^-(e) \cdot \exp\left(\delta \cdot (-y_t^\top WRBw_e^{-1}\chi_e + \frac{1}{t} y_t^\top WRd) \right)$.
   (d) For each $e \in E$, set $p_t^+(e) \leftarrow \phi_t^+(e)/\sum_{e \in E} \phi_t^+(e)$, $p_t^-(e) \leftarrow \phi_t^-(e)/\sum_{e \in E} \phi_t^-(e)$.
6. Set $x \leftarrow \frac{1}{T} \cdot \sum_{i \in [T]} y_t$.
7. Set ϕ to be the vector $-(x^\top WR)^\top$ scaled up so that $\phi^\top d = t$.
8. Output ϕ.

Now we prove Lemma 6.1 based on Lemma 6.2 by doing a careful binary search on the value of t.

Proof of Lemma 6.1. We first describe our algorithm. Begin with $t = \mathrm{poly}(n)$, which is an upper bound on $\|d\|_{\mathrm{OPT}}$. As long as Algorithm 2 with parameter t returns a flow f, we decrease the value of t by setting $t \leftarrow \frac{1+\varepsilon}{2} t$ and invoke Algorithm 2 with this new value of t. At some point Algorithm 2 must return a potential ϕ for which $\|d\|_{\mathrm{OPT}} \geq d^\top \phi = t$. This means that $\|d\|_{\mathrm{OPT}} \in (t, 2t)$ and we can run binary search in this interval to compute two values t_ℓ, t_r such that (1) $t_r - t_\ell \leq \varepsilon \|d\|_{\mathrm{OPT}}$ and (2) $\|d\|_{\mathrm{OPT}} \in (t_\ell, t_r)$. Finally, we run Algorithm 2 with parameter $t = t_\ell/(1 + \varepsilon)$ to obtain

potentials ϕ and then run Algorithm 2 with parameter $t = t_r$ to obtain a flow f. The algorithm returns the flow-potential pair (f, ϕ). The property (3) in Lemma 6.1 can be found in Claim C.3 of [Li20].

We next give the round complexity. By Lemma 6.2, for each demand vector d and a parameter t, Algorithm 2 takes $\tilde{O}(\alpha^2 \varepsilon^{-2} M)$ Minor-Aggregation rounds. By the above analysis, the binary search of t takes $O(\log(n/\varepsilon))$ times. Since each matrix-vector multiplication takes M Minor-Aggregation rounds, we conclude that the round complexity of Lemma 6.1 is $\widetilde{O}(\alpha^2 \varepsilon^{-2} M)$. $\qquad\square$

Finally, combining Sherman's transshipment framework with our distributed graph-based ℓ_1-oblivious routing, we can approximate transshipment. To make sure that the demand is satisfied exactly, we select a carefully-chosen approximation factor within the framework, followed by routing the residual demand via the ℓ_1-oblivious routing.

Proof of Theorem 1.3. Our algorithm first computes an ℓ_1-oblivious routing operator R with approximation factor $\alpha = \exp(O(\log n \log \log n)^{3/4})$, and then computes an $(1 + \varepsilon/(2\alpha))$-approximate transshipment solution f and ϕ based on R. To make sure that demand is satisfied, we use the ℓ_1-oblivious routing operator R to route the residual demand $d - Bf$ to obtain f'.

We show that flow vector $f + f'$ and potential vector ϕ satisfying the requirement. Since $Bf' = d - Bf$, we have $B(f + f') = Bf + d - Bf = d$. In addition, we have

$$
\begin{aligned}
\left\| W(f + f') \right\|_1 &\leq \|Wf\|_1 + \|Wf'\|_1 \\
&\leq \left(1 + \frac{\varepsilon}{2\alpha}\right) \|d\|_{\mathrm{OPT}} + \alpha \|d - Bf\|_{\mathrm{OPT}} \\
&\leq \left(1 + \frac{\varepsilon}{2\alpha}\right) \|d\|_{\mathrm{OPT}} + \alpha \cdot \frac{\varepsilon}{2\alpha} \|d\|_{\mathrm{OPT}} \\
&< (1 + \varepsilon) \|d\|_{\mathrm{OPT}}.
\end{aligned}
$$

The round complexity is obtained by our parameter setting, Theorem 1.5, and Lemma 6.1. $\qquad\square$

7 Distributed $(1 + \varepsilon)$-SSSP

In this section, we present our distributed algorithm to construct a single source shortest path (SSSP) tree and prove the following result.

Theorem 1.2. (Distributed SSSP). *Given an undirected and weighted graph G with edge weights in $[1, n^{O(1)}]$, there exists a distributed algorithm that computes $(1 + \varepsilon)$-approximate SSSP in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ Minor-Aggregation rounds.*

Our algorithm is obtained by a distributed implementation of a simplified version of the SSSP algorithm presented in [Li20] and mainly included in this paper for completeness.

We first review the SSSP algorithm of [Li20] in Section 7.1, and then give its distributed implementation in Section 7.2.

7.1 SSSP Algorithm

In [Li20], Li presented an algorithm to construct an approximate SSSP tree given an algorithm that approximate the transshipment problem. We start by defining the notion of *expected* single source shortest path (ESSSP) tree.

Definition 7.1 (Definition D.1 in [Li20]). Given a graph $G = (V, E)$, a source $s \in V$ and a demand vector $d \in \mathbb{R}^V$ with $d_v \geq 0$ for each $v \neq s$, an α-approximate **expected SSSP (ESSSP)** tree is a tree T such that

$$\mathbb{E}\left[\sum_{v:d_v>0} d_v \cdot \text{dist}_T(s, v)\right] \leq \alpha \sum_{v:d_v>0} d_v \cdot \text{dist}_G(s, v),$$

where $\text{dist}_T(s, v)$ and $\text{dist}_G(s, v)$ denote the distances between s and v in tree T and graph G respectively.

In [Li20], Li gave an algorithm to compute an ESSSP tree with respect to a given graph, a source, and a demand vector. The algorithm is summarized in Algorithm 3. On a high level, the ESSSP algorithm first obtains a flow vector, that is an approximate transshipment solution for the given graph and demand vector, and then samples an outgoing edge (with respect to the flow vector) for each vertex with probability proportional to the outgoing flow value. If the flow vector is acyclic, then the sampled edges form a ESSSP tree. But if the flow vector contains some directed cycles, then the sampled edges form some connected components such that each connected component contains exactly one directed cycle. For this case, the algorithm iteratively contracts connected components of the sampled graph in the input graph, and recurses on the new graph until the result is a directed tree. In the end, the algorithm uses sampled edges in all the recursions to construct an ESSSP tree. The correctness of Algorithm 3 is proved in Claim D.8 of [Li20].

Lemma 7.2 (Correctness of ESSSP, Claim D.8, [Li20]). *Given a graph $G = (V, E)$, a source $s \in V$, a demand vector $d \in \mathbb{R}^V$ with $d_v \geq 0$ for each $v \neq s$, and a parameter ε, Algorithm 3 computes an $(1 + \varepsilon)$-approximate ESSSP tree T. Furthermore, the recursion depth of the algorithm is $O(\log n)$.*

Converting ESSSP to an SSSP tree. The SSSP construction algorithm is summarized in Algorithm 4 (SSSP), which is simplified version of Algorithm 6 from [Li20]. On a high level, the algorithm tries to finds an $(1 + \varepsilon)$-ESSSP tree T^*, which can be shown to provide a $(1 + \varepsilon)$-approximate path to at least half of the nodes (appropriately weighted). We can identify which half of the nodes the path induces via the transshipment potentials ϕ. When a good path to a node v is found, we can remove v from the *target set* of nodes V' (which initially starts with $V' \leftarrow V$) and start a new iteration with a smaller V'.

This procedure intuitively produces $O(\log n)$ trees such that for each node v at least one tree offers an $(1 + \varepsilon)$-optimal s-to-v path. There is a simple "trick" to obtain a single tree that is good with respect to all nodes (described in [HL18], Algorithm ExpectedSPDistance and proven in Lemma 14): each node keeps track of its parent pointer $\text{p}(v)$. If a new tree T^* offers a better path to v than previously known, we reassign $\text{p}(v)$ to point to the parent of v in T^*.

Algorithm 4 (SSSP) is a simplified version [Li20, Algorithm 6], due to the fact that we do not require an ℓ_1-embedding to solve a transshipment instance. The correctness of Lemma 7.3 follows from Claim E.1 and Claim E.2 in [Li20].

Lemma 7.3 (Correctness of SSSP, Claim E.1 and Claim E.2, [Li20]). *Given a graph $G = (V, E)$, a source $s \in V$, and a parameter ε, Algorithm 4 computes an $(1 + \varepsilon)$-approximate SSSP tree T. Furthermore, the number of iterations of the while loop is $O(\log n)$ with high probability.*

7.2 Distributed implementation of SSSP

We give our distributed implementation of Algorithm 4 (SSSP), and prove Theorem 1.2. We start with some useful subroutines that follow from prior work [DG19]. The following lemma roots (i.e.,

Algorithm 3: ESSSP($G = (V, E, w)$, s, d, ε)   (Algorithm 4 of [Li20], restated)

1. Set $\varepsilon' = c\varepsilon/\log n$ for some constant c, and compute the flow f and the potential ϕ of an $(1 + \varepsilon')$-approximate transshipment on G with demand vector d.
2. Each vertex $u \in V \setminus \{s\}$ samples an edge (u, v) such that $f(u, v) > 0$ with probability $f(u, v)/\sum_{f(u,v)>0} f(u, v)$. Let H be the directed graph consisting of the sampled edges and directed self-loop (s, s).
3. For each connected component C of H (ignoring edge directions when computing connected components):
   (a) Let $w(C)$ be the total weights of the edges in the (unique) cycle in C.
   (b) Let T_C be the graph formed by contracting the (unique) cycle in C into a supervertex x_C.
   (c) Let v_C be a supervertex for C with demand $d'_{v_C} \leftarrow \sum_{v \in V(C)} d_v$.
4. For each edge $(u, v) \in E$:
   (a) Let C and C' be the connected components of H containing u and v respectively.
   (b) If $C \neq C'$:
      i. $e' \leftarrow (u, v)$.
      ii. $w'(u, v) \leftarrow w(u, v) + \text{dist}_{T_C}(u, x_C) + \text{dist}_{T_{C'}}(v, x_{C'}) + w(C) + w(C')$.
5. Let $s' \leftarrow v_{C_s}$, where C_s is the component in H containing s.
6. Denote graph $G' = (V', E', w')$ with $V' = \{v_C\}$ and $E' = \{e'\}$, where C is the component in H.
7. $T' \leftarrow$ ESSSP($G', s', d', (1 + \varepsilon)/(1 + 3\varepsilon') - 1$).
8. Initialize $T \leftarrow \emptyset$.
9. For each edge $(u, v) \in T'$:
   (a) $T \leftarrow T \cup (u, v)$.
10. For each connected component C of H:
   (a) Remove an arbitrary edge from the (unique) cycle in C and merge the resulting tree with T.
11. Output T.

orients the edges) given an unrooted forest and roots for each connected component.

Lemma 7.4 (Rooting a tree, [DG19]). *Let G be a graph, and $G' \subseteq G$ be a distributedly stored undirected forest and suppose that in each connected component all nodes agree on a (so-called) root node. There is a $\tilde{O}(1)$-round Minor-Aggregation algorithm to root the each tree in G' at its root (i.e., each edge computes its direction towards its component's root), and to compute the distances between any vertex and its root in G'.*

Proof. Note that we can assume without loss of generality that G' is a tree (i.e., there is a single connected component) since we can independently run algorithms on different connected components. Connected components can be easily identified via a single round by contracting all edges and using min-aggregation.

To root a single tree we utilize Theorem 5.3 in the full version of [DG19] (which can be reinterpreted as constructing a heavy-light decomposition in $\tilde{O}(1)$ rounds of Minor-Aggregation). We only use the fact that, upon exit, each edge learns the orientations towards the root. Computing the distance to the root is performed via Lemma 5.2. $\square$

Algorithm 4: SSSP($G = (V, E, w)$, $s \in V$, $\varepsilon > 0$)

1. Initialize $V' \leftarrow V \setminus \{s\}$, potential vector $\phi \leftarrow \vec{0}$, and *parent pointers* $p : V \to V$ initially $p(v) \leftarrow v$.
2. While $V' \neq \emptyset$:
   (a) Let $d \leftarrow \sum_{v \in V'}(\mathbb{1}_v - \mathbb{1}_s)$.
   (b) Obtain a $(1 + \frac{\varepsilon}{10})$-apx flow-potential pair (f^*, ϕ^*) for transshipment.
   (c) Obtain $T^* \leftarrow \text{ESSSP}(G, s, d, \Theta(\varepsilon / \log n))$.
   (d) Root T^* at s and compute distances $\text{dist}_{T^*}(s, v)$ for all v.
   (e) Let $T = (V, \{(v, \text{p}(v))\}_v)$ be the tree defined by parent pointers p.
   (f) Compute the distances $\text{dist}_T(s, v)$ for all v (with weights w).
   (g) For each vertex $v \in V$:
       i. If $\text{dist}_{T^*}(s, v) \leq \text{dist}_T(s, v)$:
          A. set $\text{p}(v) \leftarrow$ parent of v in T^*.
       ii. $\phi(v) \leftarrow \max\{\phi^*(v), \phi(v)\}$ where ϕ^* is translated so that $\phi^*(s) = 0$.
       iii. If $\text{dist}_T(s, v) \leq (1 + \varepsilon)\phi(v)$:
          A. $V' \leftarrow V' \setminus \{v\}$.
3. Output tree $T = (V, \{(v, \text{p}(v))\}_v)$ defined by parent pointers p.

We give a distributed algorithm to find all the directed cycles in a directed minor of the communication network if every vertex in the minor has one outgoing edge. The algorithm is summarized in Algorithm 5.

The algorithm iteratively samples vertices with constant probability, and contracts other vertices to the sampled vertices until a cycle can be easily identified. Then the algorithm backtracks the identified cycle through the contraction process.

Lemma 7.5 (Finding cycles). *Let G be a graph, and $G' \subseteq G$ be a distributedly stored directed subgraph (each edge knows whether $e \in E(G')$ and its direction) such that every vertex of G' has one outgoing edge (the outgoing edge can be a self-loop). There is an $\tilde{O}(1)$ round Minor-Aggregation algorithm to find all the directed cycles in G' (i.e., every vertex learns the incident edges belonging to a cycle of G').*

Proof. We first show that Algorithm 5 finds all the directed cycles in G'. Since every vertex has one outgoing edge, every connected component of G' identified in Step 1 contains exactly one cycle. For each connected component C of G', C_i is a contracted graph of C_{i-1}, and thus for each i, C_i is a directed minor of G' such that each vertex has one outgoing edge, and C_i only contains a cycle. Note that E_j forms a cycle of C_j by induction. The algorithm outputs the directed cycle for each connected component found in Step 1.

Now we show that Algorithm 5 can be implemented in $\tilde{O}(1)$ Minor-Aggregation rounds. Finding all the connected components of G' and identifying the connected components with a self-loop can be done in a single round by contracting all edges and doing a min-aggregation to find the minimum ID in the component. By the Chernoff bound, for any C_i, each directed path of length $O(\log n)$ contains a sampled vertex with probability $1 - O(1/n^{10})$. By the union bound, with probability at least $1 - O(1/n)$, for each C_i, each walk on C_i either finds a cycle or hits a sampled vertex after $O(\log n)$ steps of the walk. Note that if two walks visit the same vertex, then their following walks are the same. Hence, if a vertex is visited by different walks, we only keep the walk initiated by the vertex with the smallest ID. Thus, one can determine if there is a walk from a sampled vertex

26

Algorithm 5: FindCycles($G' = (V, E)$) where G' is a directed graph where each node has one outgoing edge
---
1. Find all the connected components of G' (ignoring edge directions only in this step).
2. For each connected component C of G', if C contains a self-loop, then return the self-loop, otherwise
    (a) Let i be 0 and C_0 be C. Every vertex in C_i samples itself with probability $1/10$.
    (b) Repeat until there is a sampled vertex v in C_i such that the walk from v along outgoing edges finds a cycle before visits another sampled vertex.
        i. For each directed edge $e = (u, v)$ of C_i such that u is sampled and v is not sampled, set $c_e = \bot$. Set $c_e = \top$ for all the other edges.
        ii. Set $C_{i+1} = C_i/\{e : c_e = \top\}$, and $i = i + 1$.
        iii. Every vertex in C_i samples itself with probability $1/10$.
    (c) Let v be an arbitrary vertex whose walk finds a cycle along outgoing edges, and E_i be the edges of the cycle obtained from the walk from v.
    (d) For $j = i - 1$ down to 0
        i. Let E_j be E_{j+1}'s corresponding edges in C_j, and V_j be the endpoints of E_j in C_j.
        ii. For each vertex of C_{j+1} which corresponds to an induced subgraph of C_j that contains two vertices u, v of V_j, add the edges on the directed path between u and v in C_j to E_j.
    (e) Return E_0.
---

finding a cycle before hitting another sampled vertex in $O(\log n)$ Minor-Aggregation rounds, and return such a sampled vertex if exists.

By Corollary 4.3, the vertex sampling and the minor construction can be implemented in $O(1)$ rounds, and thus Step 2(a) can be implemented in $O(1)$ rounds. Step 2(b) can be implemented in $\tilde{O}(1)$ rounds using the observation that every iteration of Step 2(b) reduces the number of vertices in the connected component by a constant factor with constant probability. Step 2(c) also can be implemented in $O(\log n)$ rounds by walking for $O(\log n)$ steps along the outgoing edges from a given vertex.

Note that for any $j \geq 1$, a vertex of C_j belongs to V_j if and only if the corresponding induced subgraph of C_{j-1} contains two vertices of V_{j-1}. For each induced subgraph of C_{j-1} that corresponds to a vertex of C_j, it takes $O(1)$ rounds to identify the vertices in V_{j-1}, and another $O(1)$ rounds to identify the path connecting the two vertices in V_{j-1} by Lemma 5.2. Hence Step 2(d) can be simulated in $O(\log n)$ rounds. $\qquad \square$

Now we are ready to show that Algorithm 3 can be implemented in an efficient distributed manner. Roughly speaking, besides obtaining the solution to the transshipment problem, each step of Algorithm 3 can be implemented in polylogarithmic number of rounds. Together with the round complexity of approximating transshipment (Theorem 1.3) and the fact that Algorithm 3 has a recursion depth $O(\log n)$ (Lemma 7.2), we obtain the following lemma.

Lemma 7.6 (Round complexity of ESSSP). *Suppose $G = (V, E, w)$ is a weighted graph, $s \in V$ is a distinguished node, $d \in \mathbb{R}^V$ is a demand vector satisfying $d_s \geq 0$ and $d_v \leq 0$, and $\varepsilon > 0$ is a parameter. There exists a distributed implementation of Algorithm 3 which outputs an $(1 + \varepsilon)$-approximate ESSSP in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ rounds of Minor-Aggregations.*

27

Proof. Let $G_0 = (V_0, E_0)$ be the initial input graph of Algorithm 3, and $G_i = (V_i, E_i)$ be the input graph of i-th recursion for $i > 0$. Note that G_0 is the same as G, and each G_i is a minor of G_0. Hence, all the G_i are minors of G. Throughout our implementation, for each G_i, we maintain a rooted forest F_i on G such that for each vertex u of G_i, there is a tree of F_i corresponding to a spanning tree of u's corresponding induced subgraph in G. Since each vertex of G_0 is a vertex of G, F_0 only contains isolated vertices.

We analyze the round complexity to implement Algorithm 3 (ESSSP) with respect to input graph G_i. By Theorem 1.5, Corollary 4.3 and Lemma 6.1, Line 1 can be implemented in $\varepsilon^{-2} \exp(O(\log n \cdot \log \log n)^{3/4})$ rounds.

Now we show that Line 2 can be implemented in $O(1)$ rounds for each G_i. First, for each vertex u' in G, letting u denote the vertex of G_i whose corresponding induced subgraph in G consisting u', we compute

$$f_i(u') \stackrel{\text{def}}{=} \sum_{v \in V_i : f(u,v) > 0 \text{ and } (u,v) \text{ corresponds to an edge of } G \text{ incident to } u'} f(u,v)$$

in $O(1)$ rounds. Second, we compute

$$g_i(u') \stackrel{\text{def}}{=} \sum_{v' \in V : v' \text{ is a descendant of } u' \text{ in } F_i} f_i(v')$$

in $O(1)$ rounds by Lemma 5.2. Third, each vertex u' of G samples a child v' in F_i with probability proportional to $g_i(v')/g_i(u')$, and no edge with probability $f_i(u')/g_i(u')$ in $O(1)$ rounds. Forth, for each tree of F_i, we identify the vertex where the walk from the root along edges sampled in third step stops. For each tree T in F_i, each vertex u' in T is sampled with probability $f_i(u')/\sum_{v' \in T} f_i(v')$. And this step can be simulated in $O(1)$ rounds by Lemma 5.2. At the end, in $O(1)$ rounds, each vertex u' of G identified in the last step samples an edge (u,v) with probability proportional to $f(u,v)$ among all the edges with positive flow value and corresponding to edges incident to u'. Since each vertex u of G_i, edge (u,v) with positive flow value is sampled with probability proportional to $f(u,v)$. Hence, Line 2 of Algorithm 3 can be implemented in $O(1)$ rounds.

Let H_i be the graph constructed in Line 2 of i-th recursion for $i > 0$. The connected components of H_i can be identified in a single round by contracting all edges and doing a min-aggregation, and the directed cycles of H_i can be found in $\tilde{O}(1)$ rounds by Corollary 4.3 and Lemma 7.5. Line 3 to 12 be implemented in $O(1)$ rounds by Corollary 4.3 and Lemma 5.2. The construction of G_{i+1} can be done in $O(1)$ rounds by Corollary 4.3. The edges of F_{i+1} can obtained by removing an arbitrary edge for each cycle in H_i, and taking the union with F_i, which can be simulated in $O(1)$ rounds. By Corollary 4.3 and Lemma 7.4, the edge directions of F_{i+1} can be obtained in $\tilde{O}(1)$ rounds. Line 15 to 19 can also be implemented in $O(1)$ rounds.

Hence, besides the recursion, Algorithm 3 on each input graph G_i takes $\varepsilon^{-2} \exp(O(\log n \cdot \log \log n)^{3/4})$ rounds. Since the recursion depth of Algorithm 3 is $O(\log n)$ by Lemma 7.2, Algorithm 3 can be implemented in $\varepsilon^{-2} \exp(O(\log n \cdot \log \log n)^{3/4})$ Minor-Aggregation rounds. $\square$

Now we give the distributed implementation of Algorithm 4 (SSSP) and prove the main result of this section.

Theorem 1.2. (Distributed SSSP). *Given an undirected and weighted graph G with edge weights in $[1, n^{O(1)}]$, there exists a distributed algorithm that computes $(1 + \varepsilon)$-approximate SSSP in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ Minor-Aggregation rounds.*

Proof. By Lemma 7.3, Algorithm 4 outputs an SSSP with appropriate guarantees. In this proof, we show that Algorithm 4 can be simulated in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ Minor-Aggregation rounds.

Note that the while loop on lines 2–14 executes $O(\log n)$ w.h.p. (Lemma 7.3), hence it is sufficient to bound the complexity of a single iteration of the loop. Line 4 can be implemented in $\varepsilon^{-2} \cdot \exp(O(\log n \cdot \log \log n)^{3/4})$ rounds via Theorem 1.3. Line 5 has the same round complexity via Lemma 7.6. Lines 6–8 are implemented in $\tilde{O}(1)$ rounds via Corollary 4.3 and Lemma 7.4. Other operations are trivially implementable in the Minor-Aggregation model. □

References

[ACH+13] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):1–28, 2013.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.*, 8(1):121–164, 2012.

[AHK+20] John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1280–1299, 2020.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proceedings of 31st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[AR19] Udit Agarwal and Vijaya Ramachandran. Distributed weighted all pairs shortest paths through pipelining. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 23–32, 2019.

[AR20] Udit Agarwal and Vijaya Ramachandran. Faster deterministic all pairs shortest paths in congest model. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, page 11–21, 2020.

[ARKP18] Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in õ (n 3/2) rounds. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 199–205, 2018.

[ASZ20] Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 322–335, 2020.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985.

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

[BKKL17] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models. In *31st International Symposium on Distributed Computing (DISC)*, volume 91, pages 7:1–7:16, 2017.

[BN19] Aaron Bernstein and Danupon Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, page 334–342, 2019.

[Bou85] Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[CFR21] Nairen Cao, Jeremy T Fineman, and Katina Russell. Brief announcement: An improved distributed approximate single source shortest paths algorithm. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 493–496, 2021.

[CHDKL20] Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, pages 1–25, 2020.

[CHKK+15] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, page 143–152, 2015.

[CHLP21] Keren Censor-Hillel, Dean Leitersdorf, and Volodymyr Polosukhin. On sparsity awareness in distributed computations. *arXiv preprint arXiv:2105.06068*, 2021.

[CHLT18] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse Matrix Multiplication and Triangle Listing in the Congested Clique Model. In *22nd International Conference on Principles of Distributed Systems (OPODIS)*, pages 4:1–4:17, 2018.

[CM20] Shiri Chechik and Doron Mukhtar. Single-source shortest paths in the congest model with improved bound. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 464–473, 2020.

[DFKL21] Michal Dory, Orr Fischer, Seri Khoury, and Dean Leitersdorf. Constant-round spanners and shortest paths in congested clique and mpc. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, page 223–233, 2021.

[DG19] Michal Dory and Mohsen Ghaffari. Improved distributed approximations for minimum-weight two-edge-connected spanning subgraph. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 521–530, 2019.

[DP20] Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 59–68, 2020.

[DS08] Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC)*, pages 451–460, 2008.

[Elk17] Michael Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, page 757–770, 2017.

[EN16] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 128–137, 2016.

[EN19] Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in rnc. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, page 333–341, 2019.

[FGL+20] Sebastian Forster, Gramoz Goranci, Yang P Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. Minor sparsifiers and the distributed laplacian paradigm. *arXiv preprint arXiv:2012.15675*, 2020.

[FHS20] Michael Feldmann, Kristian Hinnenthal, and Christian Scheideler. Fast hybrid network algorithms for shortest paths in sparse graphs. In *24th International Conference on Principles of Distributed Systems (OPODIS)*, pages 31:1–31:16, 2020.

[FN18] Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 686–697, 2018.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[GH16] Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 202–219, 2016.

[GH21] Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 213–221, 2021.

[GHR21] Mohsen Ghaffari, Bernhard Haeupler, and Harald Räcke. Hop-constrained expander decompositions, oblivious routing, and universally-optimal distributed algorithms. *arXiv preprint*, 2021.

[GHZ21] Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1208–1220, 2021.

[GKK+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 81–90, 2015.

[GKP98] Juan A Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

[GL18a] Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 431–444, 2018.

[GL18b] Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC)*, volume 121, pages 31:1–31:16, 2018.

[HHW18] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. Round-and message-optimal distributed graph algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 119–128, 2018.

[HIZ16a] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016.

[HIZ16b] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *International Symposium on Distributed Computing (DISC)*, pages 158–172, 2016.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, page 489–498, 2016.

[HL18] Bernhard Haeupler and Jason Li. Faster distributed shortest path approximations via shortcuts. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC)*, volume 121, pages 33:1–33:14, 2018.

[HLZ18] Bernhard Haeupler, Jason Li, and Goran Zuzic. Minor excluded network families admit fast distributed algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 465–474, 2018.

[HNS17] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed exact weighted all-pairs shortest paths in õ ($n^{5/4}$) rounds. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 168–179, 2017.

[HP16] Stephan Holzer and Nathan Pinsker. Approximation of Distances and Shortest Paths in the Broadcast Congest Clique. In *19th International Conference on Principles of Distributed Systems (OPODIS)*, pages 1–16, 2016.

[HW12] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing (PODC)*, pages 355–364, 2012.

[HWZ20] Bernhard Haeupler, David Wajc, and Goran Zuzic. Network coding gaps for completion times of multiple unicasts. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 494–505, 2020.

[HWZ21] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1166–1179, 2021.

[KKOI21] Naoki Kitamura, Hirotaka Kitagawa, Yota Otachi, and Taisuke Izumi. Low-congestion shortcut and graph parameters. *Distributed Comput.*, 34(5):349–365, 2021.

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multi-commodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 217–226, 2014.

[KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 235–244, 2010.

[KP21] Shimon Kogan and Merav Parter. Low-congestion shortcuts in constant diameter graphs. *arXiv preprint arXiv:2106.01894*, 2021.

[KS20] Fabian Kuhn and Philipp Schneider. Computing shortest paths and diameter in the hybrid network model. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 109–118, 2020.

[LG16] François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In *International Symposium on Distributed Computing (DISC)*, pages 57–70, 2016.

[Li20] Jason Li. Faster parallel algorithm for approximate shortest path. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 308–321, 2020.

[LPS13] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: Extended abstract. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, page 381–390, 2013.

[LPSP19] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Computing*, 32(2):133–157, 2019.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in Õ(vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 424–433, 2014.

[Mad10] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 245–254, 2010.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 565–573, 2014.

[NMN01] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete mathematics*, 233(1-3):3–36, 2001.

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

[Pen16]   Richard Peng. Approximate undirected maximum flows in $O(mpolylog(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 1862–1867, 2016.

[PR18]   Matteo Pontecorvi and Vijaya Ramachandran. Distributed algorithms for directed betweenness centrality and all pairs shortest paths. *arXiv preprint arXiv:1805.08124*, 2018.

[RST14]   Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 227–238, 2014.

[She13]   Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 263–269, 2013.

[She17a]   Jonah Sherman. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 452–460, 2017.

[She17b]   Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 772–780, 2017.

[SHK+12]   Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

[ST14]   Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.

[Zuz21]   Goran Zuzic. A simple boosting framework for transshipment. *arXiv preprint arXiv:2110.11723*, 2021.

A   Deferred proofs

Theorem 4.2. *Any τ-round Minor-Aggregation algorithm on G can be simulated in the (randomized) CONGEST model on G in $\tau \cdot \mathrm{poly}(\mathrm{ShortcutQuality}(G)) \cdot n^{o(1)}$ rounds. More generally, if one can solve PA in τ_{PA} rounds, any τ-round Minor-Aggregation algorithm can be simulated in $\tilde{O}(\tau \cdot \tau_{PA})$ (randomized) CONGEST rounds.*

Proof. The computation of an edge $e = \{u, v\}$ in the Minor-Aggregation model is simulated by both of its endpoints (they will always agree on the state of the edge). Naturally, the computation of a node in Minor-Aggregation model is simulated by the node itself in CONGEST. It is sufficient to show that we can simulate a single round of the Minor-Aggregation model in $\tilde{O}(\tau_{PA})$ rounds of the CONGEST model.

Leader election in each supernode. We directly follow the argument laid out in [GH16]. Initially, each node starts as its own *cluster*. In subsequent iterations, we grow the clusters until the set of clusters matches the set of supernodes (i.e., there are no outgoing contracted edges from each cluster). Every node in each cluster $C \subseteq V$ maintains the minimum ID of the node in C, a node which we call the "leader" of C.

We merge clusters together over contracted edges (i.e., $e \in E$ with $c_e = \top$). We do this by restricting the merges to be star shaped in the following way. The leader of each cluster C throws a fair random coin and labels itself "heads" or "tails". This information is then propagated (via solving a PA task) to all nodes in C in $\tilde{O}(\tau_{PA})$ rounds—following Definition 2.1, the leader sets its private input to be 1 or 0 (corresponding to heads/tails), the aggregate operation is max, each node uses the leader's ID as their part ID.

Each tails cluster C chooses an arbitrary edge connecting it to a heads cluster: (1) each node sends its leader ID to all neighbors, (2) each node sets its private input to be an arbitrary incident edge satisfying the condition (or $\bot$ if none), (3) solving a PA task inside each cluster we choose an arbitrary (e.g., one with minimal ID) such edge and inform each node in C about the leader ID of the chosen heads cluster. Each node inside a tails cluster which has found a neighboring tails cluster takes over the part ID of the heads cluster.

Consider a cluster C that has an incident contracted edge e (whose endpoints are in clusters C and D). With probability at least $1/4$, if C is tails and D is heads, the cluster C will dissapear (be merged into D). Therefore, using a standard argument, after $O(\log n)$ iterations, the clusters will exactly match the supernodes with high probability.

Contraction + Consensus step. First, we elect a leader in each supernode. In other words, each node v in a supernode $s \in V'$ agrees on some unique ID of that supernode. Then solving the distributed PA task with the leader's ID as the part ID, we can directly inform each node $v \in s$ of the aggregate $\bigoplus_{v \in s} x_v$ in τ_{PA} CONGEST rounds (Definition 2.1).

Aggregation step. Suppose that, after performing the consensus step, each node $v \in s$ is informed about y_s and v's supernode ID which we denote as $\mathrm{ID}(s)$. Each node $v \in s$ sends $\mathrm{ID}(s)$ and y_s to all v's neighbors. Consider an edge $\{a, b\} = e \in E$ and suppose they are in supernodes $a \in s_a, b \in s_b$. Both endpoints can (equally) simulate the computation on edge e: assuming $s_a \neq s_b$, e corresponds to an edge $e \in E'$, in which case a and b compute z_{e,s_a} and z_{e,s_b}. Now, each node $v \in s$ can compute the aggregate $\bigotimes_e z_{e,s}$ over all edges e incident to v. Solving the distributed PA task, for each supernode s we can compute the final aggregate $\bigoplus_{e \in \mathrm{incidentEdges}(s)} z_{e,s}$ and inform all $v \in s$ of this value. $\qquad\blacksquare$

Lemma 5.1. *[LDD sampling [HL18]] Suppose G is a weighted graph G with weights in $[1, n^{O(1)}]$. For any $\rho \geq 1$, there exists a distributed algorithm which samples an LDD from a distribution of radius ρ and quality $\exp(O(\sqrt{\log n \cdot \log \log n}))$ in $\exp(O(\sqrt{\log n \cdot \log \log n}))$ rounds of Minor-Aggregation.*

Upon termination, each node v knows the center node c_i of its LDD component S_i, and v's parent edge in the shortest path tree of $G[S_i]$ that is centered at c_i.

Proof. The algorithm is presented in [HL18], in Section 2, Algorithm LDDSubroutine and Algorithm ExpectedSPForest. In order to avoid recreating large parts of [HL18], we will assume and reuse the notation of that paper in this proof.

First, we observe that LDDSubroutine can be implemented in the Minor-Aggregation model. LDDSubroutine consists of (1) contracting 0-weighted nodes, and (2) performing multiple steps of ball-growing, where in each step each supernode becomes active if an neighboring supernode was active in the previous step. Contractions can be clearly performed via Corollary 4.3. Ball-growing can also be simulated in the Minor-Aggregation model: each edge between supernode where one side is active and the other one is not will inform the inactive side to become active. Second, we also observe that ExpectedSPForest can be implemented using Minor-Aggregations in a trivial way since it simply calls LDDSubroutine multiple times.

By reinterpreting Lemma 13, the result guarantees an LDD of quality $\frac{1}{\beta} n^{O(\log \log n)/\log(1/\beta)}$, and it runs in $\tilde{O}(\frac{1}{\beta})$ Minor-Aggregation rounds. Choosing $\beta = \exp(-O(\sqrt{\log n \cdot \log \log n}))$, we obtain the stated result.

Specifically, we run the Algorithm ExpectedSPForest while $\rho^{(t)} = (\frac{O(\log n)}{\beta})^t$ is bounded by our desired radius ρ_{target}. In each step we find an LDD of radius $\frac{O(\log n)}{\beta}$ and quality $O(\log n)$; contract the components, increase non-contracted edges by $\frac{O(\log n)}{\beta}$ (so that the distance nodes in different components never decreases), and repeat the process on the contracted graph (one can operate on the contracted graph Corollary 4.3). The returned decomposition is the one obtained in the final step of the process. This makes the radius property trivial since the final radius is at most $\rho^{(t)} \leq \rho$, where $t \leq \sqrt{\log n}$ is the number of iterations performed. The quality property is proven as follows. Let G_i be the contracted graph in step i. Consider some nodes x, y. They are separated in the final components if they are separated at every step along the way. In the penultimate step $t-1$, the expected distance between them is $\mathbb{E}[d_{G_{t-1}}(x,y)] \leq O(\log n)^{t-1} d_G(x,y)$ (Lemma 12). Therefore, the probability of them being cut in the final step is at most $\mathbb{E}[\frac{d_{G_{t-1}}(x,y)}{\rho^{(t)}} \cdot O(\log n)] \leq \frac{\mathbb{E}[d_{G_{t-1}}(x,y)]}{\rho^{(t)}} \cdot O(\log n) \leq \frac{d_G(x,y) \cdot O(\log n)^{t-1}}{\rho^{(t)}} \cdot O(\log n) = \frac{d_G(x,y)}{\rho^{(t)}} \cdot O(\log n)^t \leq \frac{d_G(x,y)}{\rho} \cdot \frac{O(\log n)}{\beta} \cdot O(\log n)^t \leq \frac{d_G(x,y)}{\rho} \cdot \exp(O(\sqrt{\log n \cdot \log \log n}))$. Therefore, by definition of quality, it is at most $\exp(O(\sqrt{\log n \cdot \log \log n}))$. $\qquad\square$