Round- and Message-Optimal Distributed Graph Algorithms^{*†}

Bernhard Haeupler
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
haeupler@cs.cmu.edu

D. Ellis Hershkowitz
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
dhershko@cs.cmu.edu

David Wajc
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
dwajc@cs.cmu.edu

ABSTRACT

Distributed graph algorithms that separately optimize for either the number of rounds used or the total number of messages sent have been studied extensively. However, algorithms simultaneously efficient with respect to both measures have been elusive. For example, only very recently was it shown that for Minimum Spanning Tree (MST), an optimal message and round complexity is achievable (up to polylog terms) by a single algorithm in the CONGEST model of communication.

In this paper we provide algorithms that are simultaneously round- and message-optimal for a number of well-studied distributed optimization problems. Our main result is such a distributed algorithm for the fundamental primitive of computing simple functions over each part of a graph partition. From this algorithm we derive round- and message-optimal algorithms for multiple problems, including MST, Approximate Min-Cut and Approximate Single Source Shortest Paths, among others. On general graphs all of our algorithms achieve worst-case optimal $\tilde{O}(D + \sqrt{n})$ round complexity and $\tilde{O}(m)$ message complexity. Furthermore, our algorithms require an optimal $\tilde{O}(D)$ rounds and $\tilde{O}(n)$ messages on planar, genus-bounded, treewidth-bounded and pathwidth-bounded graphs.¹

CCS CONCEPTS

• **Theory of computation** → **Distributed algorithms**; • **Mathematics of computing** → *Graph algorithms*; Approximation algorithms;

KEYWORDS

Distributed graph algorithms; Low-congestion shortcuts; CONGEST; Part-wise Aggregation

ACM Reference Format:

Bernhard Haeupler, D. Ellis Hershkowitz, and David Wajc. 2018. Round- and Message-Optimal Distributed Graph Algorithms. In *PODC '18: ACM*

^{*}All authors supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

[†]Full version available at [18].

¹Throughout this paper, n , m and D denote respectively the number of nodes, number of edges and the graph diameter respectively. In addition, we use tilde notation, $\tilde{O}$, $\tilde{\Omega}$ and $\tilde{\Theta}$, to suppress polylogarithmic terms in n .

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODC '18, July 23–27, 2018, Egham, United Kingdom

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-5795-1/18/07...\$15.00

<https://doi.org/10.1145/3212734.3212737>

Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3212734.3212737>

1 INTRODUCTION

Over the years, a great deal of research has focused on characterizing the optimal runtime for distributed graph algorithms in the CONGEST model of communication. Fundamental problems that have been studied include Shortest Paths [10, 23, 24, 28, 29, 31], MST [13, 25, 27, 36], Min-Cut [17, 32], and Max Flow [16]. Runtime is measured by the number of synchronous rounds of communication, and for these problems $\tilde{\Theta}(D + \sqrt{n})$ rounds are known to be necessary and sufficient [5, 7, 17, 36].

Another common performance metric optimized for in the CONGEST model is the total number of messages sent. For MST, an $\tilde{\Omega}(m)$ lower bound is known [2].² However, for several decades the only MST algorithms known to match this message lower bound had sub-optimal round complexity [1–3, 9, 11, 12]. The question of whether algorithms attaining both optimal round and message complexity has been a long-standing problem. For instance, Peleg and Rubinfeld [36] asked whether it might be achievable for MST. In a recent breakthrough work Pandurangan et al. [34] answered this question in the affirmative, providing a randomized MST algorithm with simultaneously optimal round and message complexities (up to polylog terms). Shortly thereafter Elkin [8] provided the same result without randomization. However, simultaneously round- and message-optimal algorithms for other fundamental problems have remained elusive.

1.1 Our Main Result

In this paper we advance the study of simultaneously round- and message-optimal distributed algorithms. In particular, we provide such algorithms for multiple distributed graph problems. Underlying these contributions is our main result – a round- and message-optimal algorithm for a fundamental distributed problem, which we refer to as Part-Wise Aggregation (or PA for short). We elaborate on some applications of this algorithm in Section 1.2. Informally, Part-Wise Aggregation is the problem of computing the result of a function applied to each part of a graph partition. Formally, the problem is as follows.

Definition 1.1 (Part-Wise Aggregation). *The input to Part-Wise Aggregation (PA) is:*

- (1) a graph $G = (V, E)$;

²Strictly speaking, the $\tilde{\Omega}(m)$ message lower bound for MST only holds if the algorithm is (1) deterministic (2) in the KT0 model, or (3) “comparison-based”. (Our deterministic algorithm satisfies (1), (2) and (3).) If these conditions are not met it is possible to solve MST using $\tilde{O}(n)$ messages – beating the $\tilde{\Omega}(m)$ bound for sufficiently dense graphs. For more see Mashreghi and King [30].

- (2) a partition $(P_i)_{i=1}^N$ of V , where each P_i induces a connected subgraph on G . Each node $v \in P_i$ knows an $O(\log n)$ -bit value associated with it, $\text{val}(v)$, and which of its neighbors are in P_i ;
- (3) a function f that takes as input two $O(\log n)$ -bit inputs, outputs an $O(\log n)$ -bit output and is commutative and associative.

The problem is solved if for every P_i every $v \in P_i$ knows its part's aggregate value $f(P_i) := f(\text{val}(v_1), f(\text{val}(v_2), \dots))$, where $P_i = \{v_1, v_2, \dots\}$.

The performance of our algorithm is determined by the quality of the shortcuts that the input graph admits. Shortcuts, as well as the parameters which determine their quality – termed the *block parameter*, b , and *congestion*, c – are formally defined in Section 2. For now, we note only that every graph admits a shortcut with $b = 1$ and $c = \sqrt{n}$. Our main result is as follows.

Theorem 1.2. *There exists a Part-Wise Aggregation on a graph G admitting a tree-restricted shortcut with congestion c and block parameter b w.h.p.³ in $\tilde{O}(bD + c)$ rounds and $\tilde{O}(m)$ messages, and deterministically in $\tilde{O}(b(D + c))$ rounds and $\tilde{O}(m)$ messages.*

1.2 Applications of Our Main Result

The power of Part-Wise Aggregation – and by extension Theorem 1.2 – is that numerous distributed primitives can be cast as instances of this problem. For example, it is not hard to see that electing a leader, computing the number of nodes in each tree in a forest or having every part of a graph partition agree on a minimum value are all instances of this problem. Consequently, many previous algorithms rely on subroutines which are implementable using Part-Wise Aggregation [5, 6, 12–17, 21, 27, 32]. Perhaps unsurprisingly then, using our new PA algorithm as a subroutine in some of these previous works' algorithms, we obtain round- and message-optimal solutions to numerous problems.

In the following three corollaries we highlight three such applications of our algorithm: round- and message-optimal algorithms for MST, Approximate Min-Cut and Approximate SSSP. We defer the proofs of these corollaries (and other corollaries of our main result and algorithmic approach) to the paper's full version. For a flavor of these corollaries' proofs, we note that Borůvka's MST algorithm [33] can be implemented easily using $O(\log n)$ applications of Part-Wise Aggregation, implying Corollary 1.3. Corollaries 1.4 and 1.5 are obtained by using our PA algorithm in the algorithms of [15] and [21], respectively.

The input to all three problems is an undirected weighted graph with edge weights in $[1, \text{poly}(n)]$. Initially, every node knows the weight associated with each of its incident edges. Recall that since every graph has a shortcut with $b = 1$ and $c = \sqrt{n}$, our algorithms simultaneously achieve a message complexity of $\tilde{O}(m)$ and a runtime of essentially worst-case optimal $\tilde{O}(D + \sqrt{n})$.

MST. MST is solved when every node knows which of its incident edges are in the MST.

Corollary 1.3. *Given a graph G admitting a tree-restricted shortcut with congestion c and block parameter b , one can solve MST w.h.p*

in $\tilde{O}(bD + c)$ rounds and $\tilde{O}(m)$ messages and deterministically in $\tilde{O}(b(D + c))$ rounds with $\tilde{O}(m)$ messages.

Approximate Min-Cut. Min-cut is $(1 + \epsilon)$ -approximated when every node knows whether or not it belongs to a set $S \subset V$ such that the size of the cut given by $(S, V \setminus S)$ is at most $(1 + \epsilon)\lambda$, where λ is the size of the minimum cut of G with the prescribed weights.

Corollary 1.4. *For any $\epsilon > 0$ and graph G admitting a tree-restricted shortcut with congestion c and block parameter b , one can $(1 + \epsilon)$ -approximate min-cut w.h.p. in $\tilde{O}(bD + c) \cdot \text{poly}(1/\epsilon)$ rounds and $\tilde{O}(m) \cdot \text{poly}(1/\epsilon)$ messages.*

Approximate SSSP. An instance of α -approximate single source shortest paths (SSSP) consists of an undirected weighted graph G as above and a source node s , which knows that it is the source node. For $v \in V$ we denote by $d(s, v)$ the shortest path length between s and v in G and $L = \max_{u, v} d(u, v)$. The problem is solved once every node v knows d_v such that $d(s, v) \leq d_v \leq \alpha \cdot d(s, v)$.

Corollary 1.5. *For any $\beta = O(1/\text{poly} \log n)$, given a graph G admitting a tree-restricted shortcut with congestion c and block parameter b , one can $L^{O(\log \log n) / \log(1/\beta)}$ -approximate SSSP w.h.p in $\tilde{O}\left(\frac{1}{\beta}(bD + c)\right)$ rounds and $\tilde{O}\left(\frac{m}{\beta}\right)$ messages.*

The value of β determines a tradeoff between the quality of the SSSP approximation and the round and message complexity of our algorithm. Taking $\beta = \log^{-\Theta(1/\epsilon)} n$, Corollary 1.5 yields an $O(L^\epsilon)$ -approximation algorithm using $\tilde{O}(bD + c)$ rounds and $\tilde{O}(m)$ messages.

1.3 Discussion of Our Results

There are a few salient points worth noting regarding our results, on which we elaborate below.

Round- and Message-Optimality of our Algorithms. As all graphs admit a tree-restricted shortcut with block parameter $b = 1$ and congestion $c = \sqrt{n}$, our algorithms all terminate within $\tilde{O}(D + \sqrt{n})$ rounds, which is optimal for all our applications of our PA algorithm, by [5]. As for message complexity, our $\tilde{O}(m)$ bound is tight for MST in the KT_0 model by [2]. For the other problems an $\Omega(n)$ message lower bound is trivial; for sparse graphs, then, our message complexity bound is tight for these problems. Finally, we note that our proof of Corollary 1.3 relies on solving Part-Wise Aggregation $O(\log n)$ times to solve MST, which implies that our algorithms for PA are both round- and message-optimal (again, up to polylog terms).

Beyond Worst-Case Optimality. As stated above, every graph admits tree-restricted shortcuts with block parameter $b = 1$ and congestion $c = \sqrt{n}$, which implies an $\tilde{O}(D + \sqrt{n})$ bound for our algorithms' round complexity on general graphs. However, as observed in prior work, a number of graph families of interest – planar, genus-bounded, bounded-treewidth and bounded-pathwidth graphs – admit shortcuts with better parameters [15, 19, 20]. As a result, our algorithms have a round complexity of only $\tilde{O}(D)$ times the relevant parameter of interest (e.g., genus, treewidth or pathwidth). Provided these parameters are constant or even polylogarithmic, our algorithms run in $\tilde{O}(D)$ rounds. Another recent

³Throughout the paper, by w.h.p. we mean with probability $1 - \frac{1}{\text{poly}(n)}$.

result [22] implies that our algorithms run in $\tilde{O}(D^2)$ time on minor-free graphs. We also note that our algorithms need not know the optimal values of block parameter and congestion, as a simple doubling trick can be used to approximate the best values (see [19]). In particular, our algorithms perform as well as the parameters of the best shortcut that the input graph admits.

Future Applications of This Work. Non-trivial shortcuts likely exist for graph families beyond those mentioned above. As such, demonstrating even better runtimes for our algorithms on many networks may be achieved in the future by simply proving the *existence* of efficient shortcuts on said networks. Moreover, given the pervasiveness of PA in distributed graph algorithms, the applications of our PA algorithm we present are likely non-exhaustive. We are hopeful that our PA algorithm will find applications in deriving round- and message-optimal bounds for even more problems.

Due to space constraints, we give several proof sketches rather than full proofs. Full proofs anywhere a proof sketch is given and further discussion of applications of our main result are available in the paper full version of our paper [18].

2 PRELIMINARIES

Before moving onto our formal results, we explicitly state the model of communication we consider and then review relevant concepts from previous work in low-congestion shortcuts.

2.1 CONGEST Model of Communication

Throughout this paper we work in the classic CONGEST model of communication [35]. In this model, the network is modeled as a graph $G = (V, E)$ of diameter D with $n = |V|$ nodes and $m = |E|$ edges. Communication is conducted over discrete, synchronous rounds. During each round each node can send an $O(\log n)$ -bit message along each of its incident edges. Every node has an arbitrary and unique ID of $O(\log n)$ bits, first only known to itself (this is the KT_0 model of Awerbuch et al. [2]).

2.2 (Tree-Restricted) Shortcuts

Low-congestion shortcuts were originally introduced by Ghaffari and Haeupler [15] to solve PA. These shortcuts allow high-diameter parts to communicate efficiently, by using edges outside of parts; this effectively decreases the diameter of the parts. Ghaffari and Haeupler [15] showed how, given a simple low-congestion shortcut, PA can be solved in an optimal number of rounds – i.e. $\tilde{O}(D + \sqrt{n})$ – w.h.p. Formally, a low congestion shortcut is defined as follows.

Definition 2.1 (Low-Congestion Shortcuts [15]). *Let $G = (V, E)$ be a graph and $(P_i)_{i=1}^N$ be a partition of G 's vertex set. $\mathcal{H} = H_1, \dots, H_N$ where $H_i \subseteq E$ is a c -congestion shortcut with dilation d with respect to $(P_i)_{i=1}^N$ if it satisfies*

- (1) *Each $e \in E$ belongs to at most c of the H_i .*
- (2) *The diameter of $(P_i \cup V(H_i), E[P_i] \cup H_i)$ for any i is at most d .*⁴

⁴Here $V(H_i)$ denotes all endpoints of edges in H_i and $E[P_i]$ denote the edges of G with both endpoints in P_i .

Ghaffari and Haeupler [15] also showed how to compute near-optimal $\tilde{O}(D)$ -congestion and $\tilde{O}(D)$ -dilation shortcuts for planar graphs, given an embedding of such a graph. This allowed them to obtain $\tilde{O}(D)$ -round MST algorithms for this problem, among other results. However, it was not until the work of Haeupler, Izumi, and Zuzic [19] that it was demonstrated that shortcuts could be efficiently computed in general. This work showed that high quality instances of a certain type of shortcut – *tree-restricted shortcuts* – can be efficiently approximated. These types of shortcuts are defined as follows.

Definition 2.2 (Tree-Restricted Shortcuts [19]). *Let $G = (V, E)$ be a graph and $(P_i)_{i=1}^N$ be a partition of G 's vertex set. A shortcut $\mathcal{H} = (H_i)_{i=1}^N$ is a T -restricted shortcut with respect to $(P_i)_{i=1}^N$ if there exists a rooted spanning tree T of G with $H_i \subseteq E[T]$ for all $i \in [N]$.*

Since a rooted BFS tree has minimal depth, and the $\tilde{O}(D)$ -round $\tilde{O}(m)$ -message deterministic leader election algorithm of Kutten et al. [26] allows us to compute a BFS tree in the same bounds, throughout this paper T will be a rooted BFS tree. The same work that introduced tree-restricted shortcuts also introduced a convenient alternative to dilation, block parameter.

Definition 2.3 (Block Parameter [19]). *Let $\mathcal{H} = (H_i)_{i=1}^n$ be a T -restricted shortcut on the graph $G = (V, E)$ with respect to parts $(P_i)_{i=1}^n$. For any part P_i , we call the connected components of $(P_i \cup V(H_i), H_i)$ the blocks of P_i , and the number of blocks of P_i its block parameter. The block parameter of $\mathcal{H}$, b , is the maximum block parameter of any part P_i .*

As shown in [19], if T is a depth- D tree, the dilation of a T -restricted shortcut with block parameter b is at most $O(bD)$. As such, block parameter is a convenient alternative to dilation. See Figure 1 for an example of a T -restricted shortcut.

Figure 1: An example of a T -restricted shortcut on 4 parts. $e \in E[T]$ labeled with $\{i : e \in H_i\}$. Edges directed towards T 's root. Here $c = 3$ and $b = 2$.

3 TECHNIQUES

In this section we outline our general algorithmic approach. We begin by demonstrating the message sub-optimality of previous shortcut algorithms for Part-Wise Aggregation on a particular example. We then give a workaround for this example and sketch how we develop this workaround into a full-fledged algorithm.

3.1 Bad Example for Previous Shortcut-Based Algorithms

Several prior round-optimal randomized algorithms for PA used tree-restricted shortcuts [19, 20]. To solve PA, these algorithms repeatedly aggregate within blocks. To aggregate within a block, every node in the block transmits its value up the block (along the tree's edges); when values from the same part arrive at a node in the block, they are aggregated by applying f and then forwarded up the block as a single value. By the end of this process, the root of the block has computed f of the block and can broadcast the result back down. This approach can be implemented using an optimal $\tilde{O}(D + \sqrt{n})$ rounds.

Unfortunately, there exist PA instances for which the above approach requires $\omega(m)$ messages. For example, consider the $D \times (n-1)/D$ grid graph with an additional node, r , neighboring all of the top row's nodes. Suppose each row is its own part, and all the column edges are shortcut edges, forming a single block rooted at r . See Figure 2. Aggregating within this block requires $\Omega(nD)$ messages: a message cannot be combined with other messages in its part until it has at least reached r and so each node is responsible for sending a unique message to r along a path of length $D/2$ on average. Thus, aggregating in blocks in this way to solve PA requires $\Omega(nD)$ messages, which is sub-optimal for any $D = \omega(1)$, since $m = O(n)$ for this network.

Figure 2: $\Omega(nD)$ message example. Parts are indicated by dashed rectangles. Tree edges are directed towards the root.

A Workaround. We can improve the poor message complexity of aggregating within blocks on this particular network as follows. Partition each of the D parts into *sub-parts*, each with $O(D)$ connected nodes; we have $O(n/D)$ sub-parts in total. See Figure 3. First, sub-parts aggregate: the right-most node in the sub-part broadcasts its value left and every other node broadcasts left the aggregation of its own value and what it receives from its neighbor to the right. The leftmost node of a sub-part then uses the block's edges to transmit the sub-part's aggregate value to r , which then computes the aggregate value for each part. Symmetrically to the above, r then broadcasts to every node the aggregate value for its part.

Aggregating within each sub-part requires $O(n)$ messages, as it requires each node to broadcast at most once. Moreover, there are $O(n/D)$ sub-parts, each responsible for broadcasting up and down the block once and so using the shortcut requires $O(n/D) \cdot O(D) =$

Figure 3: A workaround. Sub-parts are indicated by (smaller) dotted rectangles.

$O(n)$ messages. Therefore, for this network, our workaround requires an optimal $O(m) = O(n)$ messages.

3.2 Overview of Our Approach

The workaround of the previous subsection is heavily tailored to the particular example of Figure 2. Moreover, it requires that nodes know significantly more about the network topology than we allow. However, the above example and workaround motivate and highlight some of the notable strategies of our algorithm for Part-Wise Aggregation.

Sub-Part Divisions. As illustrated in the example, having all nodes use a shortcut in order to send their private information to their part leader rapidly exhausts our $\tilde{O}(m)$ message budget. To solve this issue, we refine the partition of our network into what we call a *sub-part division*. In a sub-part division each part P_i containing more than D nodes is partitioned into $\tilde{O}(|P_i|/D)$ sub-parts each with a spanning tree rooted at a designated node termed the *representative* of the sub-part. In the preceding example the representatives are the left-most nodes of each sub-part. Each sub-part uses its spanning tree to aggregate towards its representative, who then alone is allowed to use shortcut edges to forward the result toward the part leader. This decreases the number of nodes that use the shortcut from $O(n)$ to $\tilde{O}(n/D)$, thereby reducing the message complexity of aggregating within a block from $O(nD)$ to $\tilde{O}(n)$. Applying this observation and some straightforward random sampling ideas to previous work on low-congestion shortcuts to solve PA almost immediately implies our message-efficient randomized solutions to PA.

Message-Efficient (and Deterministic) Shortcut Construction. If our algorithms are to use shortcuts as we did in the preceding example, they must construct them message efficiently; i.e., with $\tilde{O}(m)$ messages. No previous shortcut construction algorithm achieves low message complexity. We show that not only do sub-part divisions allow us to use shortcuts message efficiently, but they also allow us to construct shortcuts message efficiently. In particular, we give both randomized and deterministic message-efficient shortcut construction algorithms. The latter is the first round-optimal deterministic shortcut construction algorithm and is based on a divide-and-conquer strategy that uses heavy path decompositions [37]. Though the general structure of our deterministic shortcut

construction algorithm is similar to that used in previous low-congestion shortcut work—nodes try to greedily claim the shortcut edges they get to use—the techniques used to deterministically implement this structure are entirely novel with respect to past work in low-congestion shortcuts.

Star Joinings. To use sub-part divisions as above, we must demonstrate how to compute them within our bounds. To do so, we begin with every node in its own sub-part and repeatedly merge sub-parts until the resulting sub-parts are sufficiently large. However, it is not clear how many sub-parts can be efficiently merged together at once, as obtained sub-parts can have arbitrarily large diameter, rendering communication within a sub-part infeasible. We overcome this issue by always forcing sub-parts to merge in a star-like fashion; this limits the diameter of the new sub-part, enabling the new sub-part to adopt the representative of the center of the star. We call this behavior *star joinings*. As we show, enforcing this behavior is easily accomplished with random coin flips. We also accomplish the same behavior deterministically but with significantly more technical overhead, drawing on the coloring algorithm of Cole and Vishkin [4].

4 SOLVING PA

In this section we show how to solve PA, given shortcuts and a sub-part division. The subroutines necessary to compute shortcuts and sub-part divisions randomly and deterministically within our time and message bounds are given in Section 5 and Section 6, respectively. Those subroutines together with our algorithm for PA given a sub-part division and shortcuts imply our main result, Theorem 1.2.

For our purposes it is convenient to assume that in our PA instance each part P_i also has a *leader* $l_i \in P_i$ where every $v \in P_i$ knows the ID of l_i . As we show in the full version of our paper, we can dispense with this assumption at the cost of logarithmic overhead in round and message complexity. As we ignore multiplicative polylogarithmic terms, for the remainder of the paper we assume that a leader for each part is always known in our PA instances.

One of the crucial ingredients we will rely on to solve PA instances as above is sub-part divisions.

Definition 4.1 (Sub-part division). *Given partition $(P_i)_{i=1}^N$ of V , a sub-part division is a partition of every part P_i into $\tilde{O}\left(\frac{|P_i|}{D}\right)$ sub-parts $S_1, \dots, S_{k_i}$. Each sub-part S_j also has a spanning tree of diameter $O(D)$ rooted at a node $r \in S_j$, termed the sub-part's representative.*

We note that sub-parts are not necessarily related to blocks in any way; e.g. a single sub-part might span multiple blocks and blocks need not contain sub-part representatives.

4.1 Aggregating on Families of Sub-trees

The second ingredient we rely on is tree-restricted shortcuts, along which we will route (some of) our messages. To do so, we must first restate an algorithm of Haeupler, Izumi, and Zuzic [19] which we refer to as BLOCKROUTE, that convergecasts/broadcasts within shortcut blocks. As convergecast and broadcast are symmetric, we only discuss convergecast.

Lemma 4.2. ([19, Lemma 2]) *Let T be a tree of depth D . Given a family of subtrees such that any edge of T is contained in at most c subtrees, there is a deterministic algorithm that can perform a convergecast/broadcast on all of the subtrees in $O(D + c)$ CONGEST rounds.*

Specifically, for convergecasts, if multiple messages are scheduled over the same edge, the algorithm forwards the packet with the smallest depth of the subtree root, breaking ties with the smallest ID of the subtree.

One observation we make about this algorithm, and which will prove crucial since we only allow representatives to use shortcuts, is the following.

Observation 4.3. *Let S be the set of nodes with a value to be convergecasted in the algorithm described in Lemma 4.2. Then the number of messages used by the algorithm is $O(|S|D)$.*

4.2 Solving PA and Verifying Block Parameter

We now show how given a sub-part division and a T -restricted shortcut, we can round- and message-efficiently solve PA with and without randomization. Our method is given by Algorithm 1 (which contains both our deterministic and randomized algorithm), and works as follows. First, each leader l_i of part P_i broadcasts an arbitrary message m_i to all nodes in P_i . Then, symmetrically to how m_i was broadcast, each l_i computes $f(P_i)$ and then broadcasts $f(P_i)$ to all nodes of P_i . The most technically involved aspect of our algorithm is how l_i broadcasts m_i to all nodes in P_i . If $|P_i|$ is smaller than D , broadcast can be trivially performed along the spanning tree of the single sub-part of P_i in $O(D)$ rounds with $O(|P_i|)$ messages. However, if $|P_i|$ is larger than D , we use shortcuts, as follows.

For our deterministic algorithm, we repeat the following b times: every representative in a block which received the message m_i spreads m_i to other representatives in its block using BLOCKROUTE along the shortcut. Next, representatives with m_i spread m_i to nodes in their sub-part. Lastly, nodes with m_i spread m_i to neighboring nodes in adjacent sub-parts. Crucially, only our representatives use shortcuts, thereby limiting our message complexity, by Observation 4.3. We illustrate the broadcast of m_i in Figure 4.

Our randomized algorithm works similarly, with the following modification: each part leader independently delays itself – and subsequently, its entire part – before sending its first message at the beginning of the algorithm, by a delay chosen uniformly in the range $[c]$ (here c is the shortcut's congestion). This limits the number of parts which would use any given edge during any round to $O(\log n)$ w.h.p. As only one message can be sent along an edge, we execute BLOCKROUTE as before, but rather than break ties as in Lemma 4.2, we simply spend $O(\log n)$ rounds between each “meta-round”, to allow each node to forward all of its $O(\log n)$ messages. This broadcast within blocks requires $O(D \log n)$ CONGEST rounds. The following lemma states the performance of our algorithms.

Lemma 4.4. *Given a sub-part division and a T -restricted shortcut with congestion c and block parameter b , Algorithm 1 uses $\tilde{O}(m)$ messages to solve PA either w.h.p in $\tilde{O}(bD + c)$ rounds or deterministically in $\tilde{O}(b(D + c))$ rounds.*

PROOF. We first prove our round complexities. We start by proving the stated round complexity for broadcasting m_i . Any part that

Algorithm 1 PA given shortcut and sub-part division.

Input: PA instance;
sub-part division;
 T -restricted shortcut

Output: solves PA

Notation: $S(v) \subseteq V := v \in V$'s sub-part;
 $r(v) \in V := v \in V$'s representative;
 $S(U) := \bigcup_{u \in U} S(u)$ for $U \subseteq V$;
 $R(U) := \{r(u) \mid u \in U\}$ for $U \subseteq V$;
 $R_i := r(P_i)$;
 $B_i(v) \subseteq V :=$ the block of part P_i containing $v \in V$

```

1: for Part  $P_i$  do
2:   if  $|P_i| < D$  then
3:     Broadcast  $m_i$  from  $l_i$  to all of  $P_i$  on  $P_i$ 's spanning tree.
4:   else
5:     if Randomized algorithm then
6:       Delay part  $P_i$  by (independent)  $\sim U(c)$ ;
7:       Blow up calls to BLOCKROUTE by  $O(\log n)$ .
8:     Route  $m_i$  from  $l_i$  to  $r(l_i)$  using BLOCKROUTE.
9:      $\mathcal{A} \leftarrow \{r(l_i)\}$ ,  $\mathcal{I} \leftarrow \{\}$ . ▷ "active"/"inactive" reps.
10:    for  $b$  iterations do
11:      BLOCKROUTE( $\mathcal{A}$ ) to send  $m_i$  to  $\bigcup_{r \in \mathcal{A}} B_i(r) \cap R_i$ .
12:       $\mathcal{A} \leftarrow \mathcal{A} \cup \bigcup_{r \in \mathcal{A}} B_i(r) \cap R_i$ .
13:      for all  $r \in \mathcal{A}$  do
14:        Broadcast  $m_i$  from  $r$  to  $S(r)$  along  $S(r)$ 's tree.
15:      Broadcast  $m_i$  over edges in  $E$  that exit  $S(\mathcal{A})$ .
16:      for all Vertex  $v \notin S(\mathcal{A}) \cup S(\mathcal{I})$  do
17:        if  $v$  received a message in line 15 then
18:           $v$  routes  $m_i$  to  $r(v)$ .
19:       $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{A}$ .
20:       $\mathcal{A} \leftarrow$  reps. that received a message in line 18.
21: Symmetrically to lines 1-20, compute  $f(P_i)$  at  $l_i$ .
22: Symmetrically to lines 1-20, b'cast result of  $f(P_i)$  from  $l_i$  to  $P_i$ .

```

is of fewer than D nodes clearly only requires $O(D)$ rounds. For any part of more than D nodes, we argue that each of the b iterations requires only $\tilde{O}(D + c)$ rounds or $O(D \log n)$ if a random delay of $U(c)$ is added. Routing m_i from l_i to $r(l_i)$ requires $O(D)$ rounds. Running BLOCKROUTE only requires $O(D + c)$ rounds by Lemma 4.2. Moreover, if a random delay is added, a Chernoff and union bound show that w.h.p an edge never has more than $O(\log n)$ distinct parts' aggregate messages that should be transmitted along it. By allowing each node to send up to $O(\log n)$ parts' aggregate message in each meta-round, BLOCKROUTE requires $O(D \log n)$ rounds, and therefore this approach requires $\tilde{O}(bD + c)$ rounds overall. Next, broadcasting m_i within any sub-part requires $\tilde{O}(D)$ rounds as sub-parts are of diameter $\tilde{O}(D)$. Broadcasting m_i to adjacent subparts requires only a single round. Lastly, computing $f(P_i)$ and broadcasting $f(P_i)$ symmetrically require $\tilde{O}(b(D + C))$ rounds.

We now prove a message complexity of $\tilde{O}(m)$. We start by proving this message complexity for broadcasting m_i . Message complexity is trivial if the part is of fewer than D nodes. Next consider parts of more than D nodes. Notice that nodes in a given sub-part only send messages in those rounds where the sub-part is active.

Figure 4: Nodes with m_j (yellow circles) and (in)active representatives at the end of each broadcast iteration for a part with 3 blocks b_1, b_2 and b_3 . The leader l_i is indicated by a black square, l_i , while sub-part representatives are indicated by stars; solid lines and dotted black lines correspond to intra and inter-sub-part edges.

Moreover, once a sub-part becomes inactive, it never again becomes active. Routing m_i from l_i to $r(l_i)$ requires $O(D)$ messages. Moreover, each of the $\tilde{O}(\frac{|P_i|}{D})$ sub-parts in part P_i use BLOCKROUTE at most once, using $O(D)$ messages per sub-part by Observation 4.3; as a result this step uses $\tilde{O}(|P_i|)$ messages for part P_i and so $\tilde{O}(n)$ messages total. Broadcasting within all sub-parts requires $O(n)$ messages since each sub-part only does so once and has a spanning tree with which to do so. Broadcasting across sub-parts uses each edge at most twice and so uses $O(m)$ messages. Lastly, computing $f(P_i)$ and broadcasting $f(P_i)$ symmetrically require $O(m)$ messages.

Correctness of broadcasting m_i is trivial if $|P_i| < D$. Moreover, if $|P_i| \geq D$, a simple argument by induction over blocks shows that b iterations suffices for parts of more than D nodes. Correctness of computing $f(P_i)$ and broadcasting $f(P_i)$ symmetrically follow. $\square$

Because our PA algorithm is essentially the same algorithm we use to verify that our shortcuts have good block parameter, we now describe this second algorithm, Algorithm 2. We verify the block parameter of a fixed part P_i as follows. Run Algorithm 1 to broadcast an arbitrary message m_i . If our block parameter is sufficiently small then every node will receive m_i and assume it as such. Moreover, if our block parameter is too large but Algorithm 1 succeeds we can still use Algorithm 1 to inform all nodes in P_i of P_i 's block parameter symmetrically to how m_i was broadcast. However, if Algorithm 1 fails –i.e. some node does not receive m_i – then we must somehow inform all nodes that the block parameter is too large. We do so by having each node that does not receive m_i inform its neighbors in P_i that it did not receive m_i . There must be some such neighbor in P_i that did receive m_i . By one additional call to Algorithm 1 this neighbor can inform all nodes that did receive m_i that the block parameter is, in fact, too large. This algorithm gives the following lemma.

Algorithm 2 Block parameter verification.

Input: partition of V , $(P_i)_{i=1}^N$, where $v \in P_i$ knows leader l_i ;
sub-part division;
 T -restricted shortcut, $\mathcal{H}$;
desired block parameter b

Output: for all i , $v \in P_i$ learns if P_i has block param. b in $\mathcal{H}$

```

1: for part  $P_i$  do
2:   Run Alg. 1 to broadcast arbitrary  $m_i$  from  $l_i$ .
3:   for  $v \in P_i$  that did not receive  $m_i$  do
4:      $v$  broadcasts  $\bar{m}_i$  to neighbors in  $P_i$ .
5:   Run Alg. 1 to broadcast if  $\exists v \in P_i$  with  $m_i$  and  $\bar{m}_i$ .
6:   for every  $i$  and  $v \in P_i$  do
7:     if  $v$  did not receive  $m_i$  or received  $\bar{m}_i$  then
8:        $v$  decides block parameter of  $P_i$  exceeds  $b$ .
9:     else Run Alg. 1 to compute the block number of  $P_i$ .

```

Lemma 4.5. *Given parts $(P_i)_{i=1}^N$, a sub-part division, a c -congestion T -restricted shortcut, $\mathcal{H}$, and desired block parameter b , one can deterministically (resp., w.h.p.) inform every node whether its part's block parameter in $\mathcal{H}$ exceeds b in $\tilde{O}(b(D + c))$ (resp. $\tilde{O}(bD + c)$) rounds with $\tilde{O}(m)$ messages.*

PROOF. Round and message complexities follow trivially from Lemma 4.4 and Lemma 4.2. We now argue correctness. If a node does not receive m_i when Algorithm 1 is first run then the block parameter of P_i is certainly larger than b . When this occurs, all nodes will either be told by l_i that the block parameter is larger than b or they will not receive m_i , implicitly informing them that the block parameter of P_i is larger than b . If all nodes receive m_i , then l_i clearly distributes to all nodes in P_i the number of blocks incident to P_i and so the block number of P_i is correctly determined to be above or below b as desired. $\square$

5 RANDOMIZED SUBROUTINES

In this section we outline how we construct sub-part divisions and shortcuts round- and message-optimally using randomization.

5.1 Computing Sub-Part Divisions

We first show how a sub-part division can be computed with randomization, by randomly sampling sub-part representatives in Algorithm 3. In particular, for large parts ($|P_i| \geq D$), every node decides to be a representative with probability $\min\{1, \frac{\log n}{D}\}$ and then representatives claim balls of radius D around them as their sub-part. Algorithm 3's properties are given below.

Lemma 5.1. *Algorithm 3 computes a sub-part division of a part with a known leader w.h.p in $O(D)$ rounds with $O(m)$ messages.*

PROOF. Runtime and message complexity are trivial. Correctness is trivial for parts of fewer than D nodes, so consider parts of more than D nodes. By construction, each claimed sub-part has diameter $O(D)$. It remains to show that every node has a representative and there are $\tilde{O}(\frac{|P_i|}{D})$ sub-parts in P_i . Fix node v and consider the ball of radius D around v . Since P_i has at least D nodes, this ball is of size at least D and so a Chernoff bound shows that w.h.p $\Theta(\log n)$ nodes in this ball will elect themselves a representative, meaning v will

Algorithm 3 Randomized sub-part division.

Input: partition of V , $(P_i)_{i=1}^N$, where $v \in P_i$ knows leader l_i
Output: sub-part division

```

1: for part  $P_i$  do
2:   if  $|P_i| \leq D$  then
3:     Let  $P_i$  have one sub-part with representative  $l_i$ .
4:     Compute sub-part spanning tree by a BFS in  $P_i$  from  $l_i$ .
5:   else
6:     for  $v \in P_i$  do
7:       if Coin flip with bias  $\min\{1, \frac{\log n}{D}\}$  is heads: then
8:          $v$  is a representative.
9:          $v$  sends its ID to  $P_i$  neighbors.
10:    for  $O(D)$  rounds do
11:       $v$  transmits first ID it hears to  $P_i$  neighbors.
12:       $v$ 's sub-part parent  $\leftarrow$  first neighbor  $v$  hears.
13:       $v$  computes for which neighbors it is a parent.

```

have a representative. A union bound over all v shows this to hold for every node. Moreover, the expected number of representatives in part P_i is $\frac{|P_i| \log n}{D}$ and so a Chernoff bound shows that w.h.p there are $\tilde{O}(\frac{|P_i|}{D})$ sub-parts in P_i . A union bound shows this holds for all parts w.h.p. $\square$

5.2 Computing Shortcuts

We now show in Algorithm 4 how we message-efficiently construct a T -restricted shortcut with randomization. We rely on the COREFAST shortcut construction algorithm of Haeupler et al. [19]. In COREFAST, a sub-sampled set of vertices broadcast up T , attempting to “claim” edges; edges with too many vertices trying to claim them are discarded. To control the message complexity, we only have the $\tilde{O}(n/D)$ sub-part representatives attempt to claim edges. The correctness and runtime of Algorithm 4 is given by Lemma 5.2.

Algorithm 4 Randomized shortcut construction.

Input: partition of V , $(P_i)_{i=1}^N$ where $v \in P_i$ knows leader l_i ;
BFS tree T ;
sub-part division

Output: T -restricted shortcut with congestion $\tilde{O}(c)$ and block parameter $< 3b$

```

1: Set all  $P_i$  active.
2: for  $O(\log n)$  iterations do
3:   Run COREFAST [19] on representatives in active parts.
4:   Run Alg. 2 on line 3 result to check if block params.  $< 3b$ .
5:   for each  $P_i$  do
6:     if block parameter of  $P_i$  is  $< 3b$  in line 4 then
7:       Set  $P_i$  inactive.
8:   for each newly inactive  $P_i$  do
9:      $P_i$  uses the edges line 3 gives it as shortcut edges.

```

Lemma 5.2. *Given partition partition of V , $(P_i)_{i=1}^N$ where $v \in P_i$ knows leader l_i , a sub-part division, spanning tree T and the existence of a T -restricted shortcut with congestion c and block parameter b , Algorithm 4 computes a T -restricted shortcut with congestion at most*

$\tilde{O}(c)$ and block parameter at most $3b$ in $\tilde{O}(bD + c)$ rounds with $O(n)$ messages w.h.p.

PROOF (SKETCH). Since only the $\tilde{O}(n/D)$ sub-part representatives broadcast up a tree of depth $O(D)$, each uses $O(D)$ messages and so we use $O(n)$ messages. Round complexity is trivial. Congestion of the shortcut follows by construction. The block parameter can be shown by arguing that we can only discard so many extra edges relative to the optimal shortcut and each extra discarded edge induces a new block. $\square$

6 DETERMINISTIC SUBROUTINES

In this section we show how to construct sub-part divisions and shortcuts deterministically.

6.1 Computing Star Joinings

Our algorithm for constructing sub-part divisions repeatedly merges together sub-parts until they are of sufficient size. However, if sub-parts are allowed to merge arbitrarily, the resulting sub-parts may have prohibitively large diameter. The diameter of resulting sub-parts can be limited by forcing sub-parts to always join in a star-like fashion. As such, we begin by providing a deterministic algorithm to enable such behavior. We term this behavior a *star joining*.

Definition 6.1 (Star joining). Let $(P_i)_{i=1}^N$ partition V . We say a star joining is computed over parts $(P_i)_{i=1}^N$ if the following holds: a constant fraction of the parts P_i are designated as receivers, and the other parts P_i are designated as joiners. For every joiner part P_i , all $v \in P_i$ knows some (common) edge with one endpoint in P_i and another end-point in some receiver part P_j .

We now show how a star joining can be computed deterministically, given a deterministic PA solution. We use as a sub-routine the 3-coloring algorithm of Cole and Vishkin [4]. Roughly, the Cole and Vishkin [4] algorithm works as follows. Every node begins with its ID as its color, meaning there are initially n colors. Next, every node updates its color based on its neighbors' colors, logarithmically reducing the number of possible colors. This is then repeated $\log^* n$ times. For more, see Cole and Vishkin [4]. The properties of this algorithm are as follows.

Lemma 6.2. ([4, Corollary 2.1]) An oriented n -vertex graph with maximum out-degree of one can be 3-colored in $O(\log^* n)$ rounds with $O(m \log^* n)$ messages.

We give our algorithm for deterministically computing star joinings in Algorithm 5 which works as follows. Take the super-graph whose nodes are parts and whose edges are the chosen (directed) edges. First, designate parts with at least two incoming edges receivers and all parts with an outgoing edge into one such part a joiner. These parts constitute all trees in our super-graph and so we next remove them from the super-graph, leaving only (directed) paths and (directed) cycles. On the remaining paths and cycles, simulate the Cole-Vishkin algorithm to compute a 3-coloring of the remaining nodes in the super-graph. For colors $k = 1, 2, 3$ make all k -colored parts receivers, their neighbors joiners and remove these parts from this process. The properties of our deterministic star joining algorithm are given by the following lemma.

Algorithm 5 Deterministic star joining.

Input: $(P_i)_{i=1}^N$ s.t. $v \in P_i$ knows $e_i \in E$ exiting P_i and leader l_i ;
PA algorithm $\mathcal{A}$

Output: a star joining on $(P_i)_{i=1}^N$

```

1:  $\mathcal{J}, \mathcal{R} \leftarrow \emptyset$  ▷ Initialize joiners and receiver
2:  $G' \leftarrow ((P_i)_{i=1}^N, \{e_i\}_{i=1}^N)$ 
3:  $\mathcal{R} \leftarrow \mathcal{R} \cup \{P_i : \delta_{G'}^-(P_i) \geq 2\}$  by  $\mathcal{A}$ .
4:  $\mathcal{J} \leftarrow \mathcal{J} \cup \{P_i : P_i \notin \mathcal{R} \wedge e_i = (P_{i'}, P_i) \text{ s.t. } P_{i'} \in \mathcal{R}\}$  by  $\mathcal{A}$ .
5:  $G' \leftarrow G' \setminus (\mathcal{R} \cup \mathcal{J})$ .
6: Run the 3-coloring algorithm of [4] on  $G'$ .
7: for color  $k = 1, 2, 3$  do
8:    $\mathcal{R} \leftarrow \mathcal{R} \cup \{P_i : P_i \text{ colored } k\}$  by  $\mathcal{A}$ .
9:    $\mathcal{J} \leftarrow \mathcal{J} \cup \{P_i : P_i \notin \mathcal{R} \wedge e_i = (P_{i'}, P_i) \text{ s.t. } P_{i'} \in \mathcal{R}\}$  by  $\mathcal{A}$ .
```

Lemma 6.3. Let $(P_i)_{i=1}^N$ partition V and suppose every $v \in P_i$ knows some edge $e_i \in E$ exiting P_i . If algorithm $\mathcal{A}$ solves PA over $(P_i)_{i=1}^N$, then Algorithm 5 computes a star joining over $(P_i)_{i=1}^N$ with $O(\log^* n)$ calls to $\mathcal{A}$.

PROOF (SKETCH). Standard graph-theoretic arguments show that a constant fraction of nodes are receivers. Moreover, it is not too hard to see that the Cole-Vishkin algorithm can be efficiently simulated within our bounds. $\square$

6.2 Computing Sub-Part Divisions

We now use star joinings to deterministically compute sub-part divisions in Algorithm 6 as follows: start with each node in its own sub-part; compute star joinings and merge stars of joiners centered around receivers $O(\log n)$ times, fixing sub-parts once they have at least D nodes. Correctness and runtime of Algorithm 6 are given by the following lemma.

Algorithm 6 Deterministic sub-part division.

Input: partition of V , $(P_i)_{i=1}^N$

Output: a sub-part division

```

1: for part  $P_i$  do
2:    $\mathcal{I}_i \leftarrow \{\{v\} : v \in P_i\}$  ▷ Initialize incomplete sub-parts
3:    $\mathcal{C}_i \leftarrow \{\}$  ▷ Initialize complete sub-parts
4:   for  $O(\log n)$  iterations do
5:     for  $F_j \in \mathcal{I}_i$  do
6:       if  $\exists$  edge  $(u, v) \in F_j \times (\bigcup_{F_{j'} \in \mathcal{I}_i} F_{j'} \setminus F_j)$  then
7:          $e_j \leftarrow e$  for such edge  $e = (u, v)$  by PA.
8:       else
9:          $e_j \leftarrow$  any  $e = (u, v) \in F_j \times (\bigcup_{F_{j'} \in \mathcal{C}_i} F_{j'})$  by PA.
10:      Run Alg. 5 on  $\mathcal{I}_i$  sub-parts with edges  $\{e_j\}$ .
11:      for Joiner  $F_j$  with edge  $e_j = (u, v)$  with  $v \in F_{j'}$  do
12:         $F_j$  merges with  $F_{j'}$  by PA.
13:         $u$  remembers  $v$  as its parent.
14:         $F_j$  orients its tree edges to  $v$  by PA.
15:       $\mathcal{C}'_i \leftarrow \{F_j \in \mathcal{I}_i : |F_j| \geq D\}$  by PA.
16:       $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \mathcal{C}'_i$ .
17: return Division given by  $\{\mathcal{C}_i\}_{i=1}^N$ .
```

Lemma 6.4. Given partition $(P_i)_{i=1}^N$ of V , Algorithm 6 computes a sub-part division of $(P_i)_{i=1}^N$ in $\tilde{O}(D)$ rounds with $\tilde{O}(n)$ messages.

6.3 Computing Shortcuts

Having shown how sub-part divisions can be computed in a deterministic fashion, we now turn to our deterministic shortcut construction. We rely on heavy path decompositions [37].

Definition 6.5 (Heavy Path Decomposition [37]). Given a directed tree T , an edge (u, v) of T is heavy if the number of v 's descendants is more than half the number of u 's descendants; otherwise, the edge is light. A heavy path decomposition of T consists of all the heavy edges in T .

It is immediate from the definition that each leaf-to-root path on an n -node tree T intersects at most $\lceil \log_2 n \rceil$ different paths of T 's heavy path decomposition. Given a rooted tree T of depth D , a heavy path decomposition of T can be easily computed in $O(D)$ rounds using $O(n)$ messages.

Our deterministic shortcut construction algorithm, Algorithm 8, first computes a heavy path decomposition and then computes shortcuts on the obtained paths in a bottom-up order. Thus, we first provide a sub-routine, Algorithm 7, that computes shortcuts of congestion $O(c \log D)$ on a path P . Algorithm 7 assumes every node v begins with a set $S(v)$ of part IDs that would like to use v 's parent edge in the path. For simplicity, we assume vertices of $\mathcal{P}$ are numbered by their height, $v = 1, 2, \dots$ (i.e., the source of the path is number 1, its parent is numbered 2, etc'). Algorithm 7 iteratively extends paths used for shortcuts, repeatedly doubling them in length, unless too much congestion results. See Figure 5. This algorithm's properties are as follows.

Algorithm 7 Deterministic shortcut construction for paths.

Input: Path $P \subseteq V$;

Mapping $S : V \rightarrow 2^{(P_i)_{i=1}^N}$;
Desired congestion c

Output: Mapping $S_f : V \rightarrow 2^{(P_i)_{i=1}^N}$

```

1: for  $v \in V$  do
2:   Set  $S_0(v) \leftarrow S(v)$ .
3: for  $i = 0, 1, 2, \dots, \log_2 D - 1$  do
4:   for every node  $v \equiv 2^i \pmod{2^{i+1}}$  do
5:     if  $|S_i(v)| \geq 2c$  then
6:       Break  $(v, v+1)$  and set  $S_i(v) \leftarrow \emptyset$ .
7:     else
8:        $u \leftarrow v + 2^i$ 
9:       if no broken edges between  $v$  and  $u$  then
10:        Transmit  $S_i(v)$  from  $v$  to  $u$  along  $P$ .
11:        Set  $S_{i+1}(u) \leftarrow S_i(u) \cup S_i(v)$ .
12: return  $S_f = S_{\log_2 D}$ .
```

Lemma 6.6. Given directed path P of length D , desired congestion c and $S : V \rightarrow 2^{(P_i)_{i=1}^N}$ which denotes for each vertex v which parts want to use v 's parent edge in P , Algorithm 7 returns $S_f : V \rightarrow 2^{(P_i)_{i=1}^N}$ s.t. for every $v \in P$ it holds that $|S_f(v)| = O(c \log D)$ in $O(c \log D + D)$ rounds.

Figure 5: Illustration of Algorithm 7. Source of colored edge is vertex v that transmits $S_i(v)$ and sink is vertex u that updates $S_{i+1}(u)$ (assuming no edges broken). Black edges are path edges.

PROOF. To bound the running time, observe that iteration i of the algorithm can be implemented in $c + 2^i$ rounds. Summing over all iterations $i = 0, 1, \dots, \log_2 D - 1$, the bound on the number of rounds follows. To bound the congestion of the output shortcuts, we prove by induction that before the i -th iteration the congestion on any edge is at most $2ci$. This clearly holds for $i = 0$. Assume as an inductive hypothesis that before iteration i all edges are used by at most $2ci$ parts. In iteration i the only edges whose congestion are potentially increased are those edges exiting u (i.e. edge $(u, u+1)$) such that $u \equiv 0 \pmod{2^{i+1}}$. The congestion on this edge increases by $|S_i(v)|$ where $v \equiv 2^i \pmod{2^{i+1}}$. Applying our inductive hypothesis we get that the total congestion on such an edge is at most $|S_i(v)| + |S_i(u)| = 2ci$, implying the claimed bound on the congestion. $\square$

We now turn to describing the overall shortcut construction algorithm, Algorithm 8, and analyze the resulting block parameter there. We limit message complexity by only allowing sub-part representatives to send a message requesting that an edge be used in their part's shortcut. As we show, each bottom-up computation yields good shortcuts for a constant fraction of parts. Thus, after each bottom-up computation, we can use our block parameter verification algorithm – Lemma 4.5 – to identify the parts for which our shortcut construction succeeded and freeze the shortcut edges of said parts. The correctness and runtime of Algorithm 8 is given by Lemma 6.7.

Lemma 6.7. Given: partition $(P_i)_{i=1}^N$ where $v \in P_i$ knows leader $l_i \in P_i$; a tree T of depth D which admits a T -restricted shortcut of congestion c and block parameter b ; and a sub-part division, Algorithm 8 deterministically computes in $\tilde{O}(b(c + D))$ rounds and $\tilde{O}(m)$ messages, a shortcut with congestion $O(c \log n)$ and block parameter $O(b)$.

PROOF (SKETCH). The stated round complexity comes from summing sub-routines. The message complexity follows similarly and requires noting that only the $\tilde{O}(n/D)$ sub-part representatives forward their part ID up at most D nodes when running Algorithm 7. Lastly, there can only be a constant number of parts with bad block parameter in each iteration. Each extra block we introduce corresponds to a broken edge that is not broken by the original shortcut. However, each such edge also corresponds to some number of parts that tried to use it and since a part can only try to use so many edges in our shortcut we have that not that many extra edges can be broken. $\square$

Algorithm 8 Deterministic shortcut construction.

Input: partition of V , $(P_i)_{i=1}^N$ with leader l_i known by $v \in P_i$;
sub-part division
BFS-tree T
Output: T -restricted shortcut with congestion $\tilde{O}(c)$ and block
parameter $< 3b$

- 1: Initially all P_i are *active*.
- 2: Compute heavy path decomposition of T .
- 3: **for** $j = 1, 2, \dots, O(\log n)$ **do**
- 4: **for all** $v \in V$ **do**
- 5: **if** v is representative in active part P_i **then**
- 6: $S_j(v) \leftarrow \{l_i\}$.
- 7: **else**
- 8: $S_j(v) \leftarrow \emptyset$.
- 9: Set each heavy path with no incoming light edges *active*.
- 10: **for** $\lfloor \log n \rfloor$ repetitions **do**
- 11: $S_f \leftarrow$ output of Alg. 7 run on all active heavy paths.
- 12: **for** active path sink node v and light edge (v, u) **do**
- 13: $S_j(u) \leftarrow S_{j-1}(u) \cup \bigcup_v S_f(v)$.
- 14: Set active paths inactive.
- 15: Set heavy paths with source u as in line 13 *active*.
- 16: Run Alg. 2 on active parts to check if block params. $< 3b$.
- 17: Set parts with block parameter $< 3b$ *inactive*.
- 18: **return** $\bigcup_j S_j(v)$ as v 's shortcut edges

REFERENCES

- [1] Baruch Awerbuch. 1987. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*. 230–240.
- [2] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. 1990. A Trade-off Between Information and Communication in Broadcast Protocols. *Journal of the ACM (JACM)* 37, 2 (1990), 238–256.
- [3] F Chin and HF Ting. 1985. An almost linear time and $O(n \log n + e)$ messages distributed algorithm for minimum-weight spanning trees. In *Proceedings of the 26th Symposium on Foundations of Computer Science (FOCS)*. 257–266.
- [4] Richard Cole and Uzi Vishkin. 1986. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control* 70, 1 (1986), 32–53.
- [5] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing (SICOMP)* 41, 5 (2012), 1235–1265.
- [6] Michael Elkin. 2006. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. System Sci.* 72, 8 (2006), 1282–1308.
- [7] Michael Elkin. 2006. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal on Computing (SICOMP)* 36, 2 (2006), 433–456.
- [8] Michael Elkin. 2017. A Simple Deterministic Distributed MST Algorithm, with Near-Optimal Time and Message Complexities. *Proceedings of the 36th ACM Symposium on Principles of Distributed Computing (PODC)* (2017).
- [9] Michalis Faloutsos and Mart Molle. 2004. A linear-time optimal-message distributed algorithm for minimum spanning trees. *Distributed Computing* 17, 2 (2004), 151–170.
- [10] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. 2012. Networks cannot compute their diameter in sublinear time. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1150–1162.
- [11] Eli Gafni. 1985. Improvements in the time complexity of two message-optimal election algorithms. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing (PODC)*. 175–185.
- [12] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. 1983. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 5, 1 (1983), 66–77.
- [13] Juan A Garay, Shay Kutten, and David Peleg. 1998. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing (SICOMP)* 27, 1 (1998), 302–316.
- [14] Mohsen Ghaffari. 2014. Near-optimal distributed approximation of minimum-weight connected dominating set. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*. 483–494.
- [15] Mohsen Ghaffari and Bernhard Haeupler. 2016. Distributed Algorithms for Planar Networks II: Low-congestion Shortcuts, MST, and Min-Cut. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 202–219.
- [16] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. 2015. Near-optimal distributed maximum flow. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*. 81–90.
- [17] Mohsen Ghaffari and Fabian Kuhn. 2013. Distributed minimum cut approximation. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*. 1–15.
- [18] Bernhard Haeupler, D Ellis Hershkowitz, and David Wajc. 2018. Round- and Message-Optimal Distributed Graph Algorithms. *arXiv preprint arXiv:1801.05127* (2018).
- [19] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. 2016. Low-congestion shortcuts without embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*. 451–460.
- [20] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. 2016. Near-Optimal Low-Congestion Shortcuts on Bounded Parameter Graphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*. 158–172.
- [21] Bernhard Haeupler and Jason Li. 2018. Beating $O(\sqrt{n} + D)$ for Distributed Shortest Path Approximations via Shortcuts. *arXiv preprint arXiv:1802.03671* (2018).
- [22] Bernhard Haeupler, Jason Li, and Goran Zuzic. 2018. Minor Excluded Network Families Admit Fast Distributed Algorithms. *arXiv preprint arXiv:1801.06237* (2018).
- [23] Stephan Holzer and Roger Wattenhofer. 2012. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*. 355–364.
- [24] Taisuke Izumi and Roger Wattenhofer. 2014. Time Lower Bounds for Distributed Distance Oracles. In *International Conference on Principles of Distributed Systems (OPODIS)*. 60–75.
- [25] Maleq Khan and Gopal Pandurangan. 2006. A fast distributed approximation algorithm for minimum spanning trees. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*. 355–369.
- [26] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. 2015. On the complexity of universal leader election. *Journal of the ACM (JACM)* 62, 1 (2015), 7.
- [27] Shay Kutten and David Peleg. 1995. Fast distributed construction of k -dominating sets and applications. In *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing (PODC)*. 238–251.
- [28] Christoph Lenzen and Boaz Patt-Shamir. 2015. Fast partial distance estimation and applications. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*. 153–162.
- [29] Christoph Lenzen and David Peleg. 2013. Efficient distributed source detection with limited bandwidth. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*. 375–382.
- [30] Ali Mashreghi and Valerie King. 2017. Time-communication Trade-offs for Minimum Spanning Tree Construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN)*.
- [31] Danupon Nanongkai. 2014. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*. 565–573.
- [32] Danupon Nanongkai and Hsin-Hao Su. 2014. Almost-tight distributed minimum cut algorithms. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*. 439–453.
- [33] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. 2001. Otakar Borůvka on Minimum Spanning Tree Problem Translation of Both the 1926 Papers, Comments, History. *Discrete Mathematics* 233, 1 (2001), 3–36.
- [34] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. 2017. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*. 743–756.
- [35] David Peleg. 2000. *Distributed computing: a locality-sensitive approach*. SIAM.
- [36] David Peleg and Vitaly Rubinfeld. 2000. A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction. *SIAM Journal on Computing (SICOMP)* 30, 5 (2000), 1427–1442.
- [37] Daniel D Sleator and Robert Endre Tarjan. 1983. A data structure for dynamic trees. *J. Comput. System Sci.* 26, 3 (1983), 362–391.