Round- and Message-Optimal Distributed Graph Algorithms

Bernhard Haeupler*, D. Ellis Hershkowitz*, David Wajc*

Carnegie Mellon University, {haeupler,dhershko,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]

*Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

[1]Throughout 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.

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, 22, 23, 28, 29, 32], MST [13, 25, 26, 37], Min-Cut [16, 33], and Max Flow [17]. 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, 16, 37].

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].[1] However, for several decades the only MST algorithms known to match this message lower bound had sub-optimal round complexity [1, 2, 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 Rubinovich [37] asked whether it might be achievable for MST. In a recent breakthrough work Pandurangan et al. [35] answered this question in the affirmative, providing a randomized MST algorithm with simulataneously 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, as well as Appendix A. 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 $G = (V, E)$;*
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, 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 $f(P_i) := f(val(v_1), f(val(v_2), \dots))$, 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

[1] Strictly 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].

block parameter, b, and *congestion*, c – are formally defined in Section 2.2. 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.*[2] *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, 13, 14, 15, 16, 17, 18, 26, 33]. 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 give proofs of these corollaries and also discuss further applications of our PA algorithm and our subroutines in Appendix A. For a flavor of these corollaries' proofs, we note that Borůvka's MST algorithm [34] 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 Ghaffari and Haeupler [15] and Haeupler and Li [18], respectively.

The input to all three problems consists of 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. Since every graph admits a shortcut with $b = 1$ and $c = \sqrt{n}$, our algorithms simultaneously achieve message complexities of $\tilde{O}(m)$ and runtimes 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.*

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

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 result [21] implies that our algorithms run in $\tilde{O}(D^2)$ time on minor-free graphs. We elaborate on our results for all the above graphs in Appendix C. 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.

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 [36]. 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 Shortcuts and 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. $\mathcal{H} = H_1, \ldots, H_N$ 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 edge $e \in E$ 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.*[3]

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. [27] 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, termed *block parameter*.

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

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.

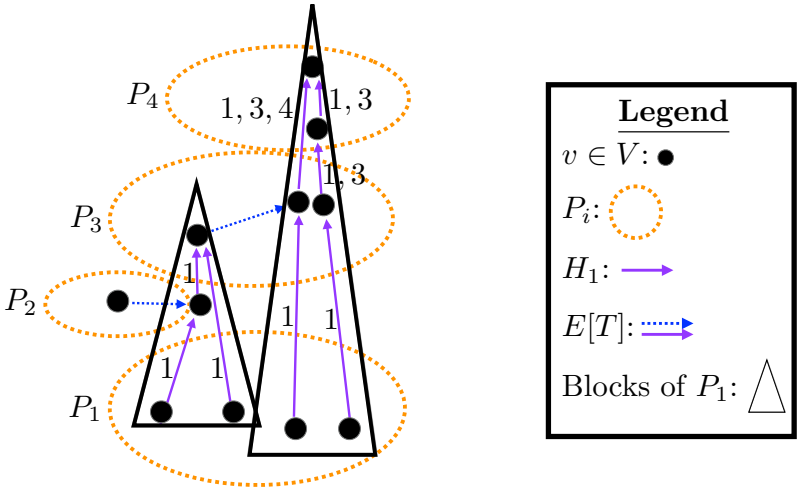


Figure 1: An example of a T-restricted shortcut on 4 parts. $e \in E[T]$ labeled with $\{i : e \in H_i\}$. Edges directed towards root of T. Here $c = 3$ and $b = 2$.

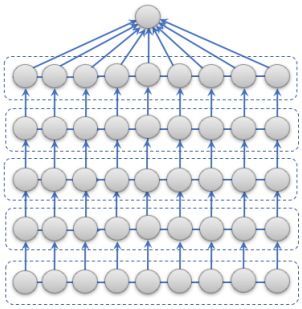
3 Techniques

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.

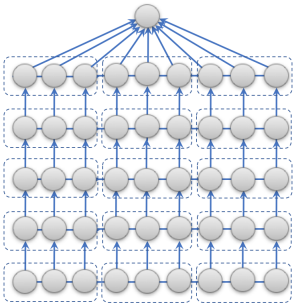
3.1 Bad Example for Previous Shortcut-Based Algorithms

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



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



(b) A workaround. Here sub-parts are indicated by (smaller) dotted rectangles.

Figure 2: A bad example for prior shortcut algorithms, and a workaround.

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 2a. 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.

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 2b. 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 procedure, 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 $O(n/D) \cdot O(D) = O(n)$ 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 2a. 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 [39]. 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 Appendix B, 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 *$S_1, \ldots, S_{k_i}$. 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*.*

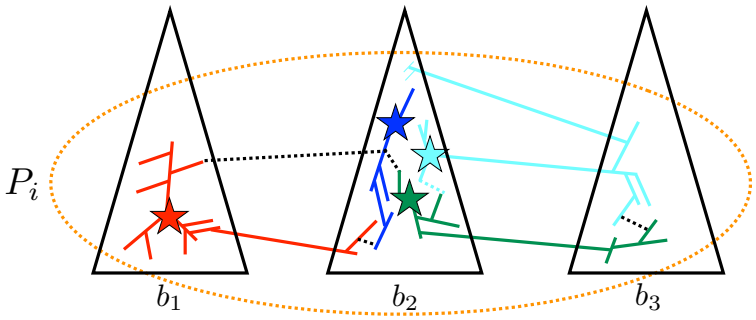


Figure 3: A division of a part, P_i, incident to blocks b_1, b_2 and b_3, into 4 sub-parts. Sub-part representatives: stars. Solid colored lines: edges in the tree of each sub-part according to the color of the representative. Dashed lines: edges in E between sub-parts.

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. See Figure 3 for an illustration of how sub-parts and blocks might interact.

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 KLPA and Verifying the 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 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. 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 (ignoring congestion issues). 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. Moreover, once a sub-part becomes inactive, it never again

Algorithm 1 PA given shortcut and sub-part division.

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

Notation: for any $v \in V$, $S(v) \subseteq V$ and $r(v) \in V$ denote v's sub-part and its representative;
for any $U \subseteq V$ we let $S(U) = \bigcup_{u \in U} S(u)$ and $R(U) = \{r(U) \mid u \in U\}$;
for a part P_i we denote by $R_i = \{r(U) \mid u \in U\}$ the set of representatives in P_i;
for part P_i and node $v \in V$, we denote by $B_i(v) \subseteq V$ the block of P_i incident to v;

Output: solves PA

1: **for** Part P_i **do**
2: **if** $|P_i| < D$ **then**
3: Broadcast m_i from l_i to all of P_i along 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 subsequent calls to BlockRoute by $O(\log n)$.
8: Route m_i from l_i to $r(l_i)$ using BlockRoute.
9: $\mathcal{A} \leftarrow \{r(l_i)\}$, $\mathcal{I} \leftarrow \{\}$. ▷ Initialize sets of "active"/"inactive" representatives.
10: **for** b iterations **do**
11: Run BlockRoute on $\mathcal{A}$ to send m_i to all nodes in $\bigcup_{r \in \mathcal{A}} B_i(r) \cap R_i$.
12: $\mathcal{A} \leftarrow \mathcal{A} \bigcup_{r \in \mathcal{A}} B_i(r) \cap R_i$.
13: **for all** $r \in \mathcal{A}$ **do**
14: Broadcast m_i from r to $S(r)$ along $S(r)$'s representing tree.
15: Broadcast m_i over edges in E that exit sub-parts in $S(\mathcal{A})$.
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: $\mathcal{A} \leftarrow$ representatives 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, broadcast result of $f(P_i)$ from l_i to all nodes in P_i.

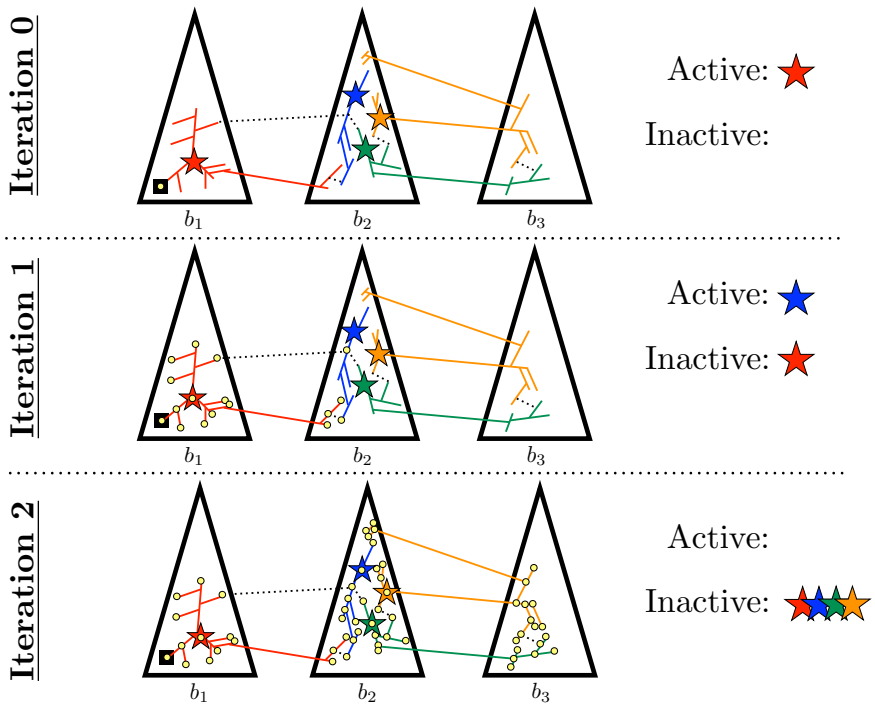


Figure 4: Nodes with m_j (yellow circles) and (in)active representatives at the end of each broadcast iteration for a part with 3 blocks b_1, b_2 and b_3. The leader l_i is indicated by a black square, l_i, while sub-part representatives are indicated by stars; solid lines and dotted black lines correspond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

becomes active. Consequently, routing m_i to l_i requires $O(n)$ messages across all sub-parts in all parts. Moreover, each of the $\tilde{O}\left(\frac{|P_i|}{D}\right)$ sub-parts in part P_i use BlockRoute at most once, using $O(D)$ messages per sub-part by Lemma 4.2; 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 $|P_i| > D$, 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. □

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. 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;
desired block parameter b

Output: for every P_i, $v \in P_i$ learns if P_i has block parameter b in the input shortcut

1: **for** part P_i **do**
2: Run Algorithm 1 to broadcast arbitrary message 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 Algorithm 1 to broadcast if a node that received m_i also received $\bar{m}_i$.
6: **for** every i and $v \in P_i$ **do**
7: **if** P_i did not receive m_i or received $\bar{m}_i$ **then**
8: v decides block parameter of P_i exceeds b.
9: **else** Run Algorithm 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. □

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 Randomly

We first show how a sub-part division can be computed with randomization, by randomly sampling sub-part representatives. 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. This is Algorithm 3, whose 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.*

Algorithm 3 Randomized sub-part division.

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

for part P_i **do**
  if $|P_i| \leq D$ **then**
    Let P_i have one sub-part with representative l_i.
    Compute sub-part spanning tree by an $O(D)$ round BFS restricted to P_i starting at l_i.
  else
    for $v \in P_i$ **do**
      With prob. $\min\{1, \frac{\log n}{D}\}$, node v is a representative and sends its ID to P_i neighbors.
      for $O(D)$ rounds **do**
        v broadcasts the first representative ID it hears to neighbors in P_i once.
        v's sub-part parent is the neighbor from which it first heard a representative ID.
        $v \in P_i$ determines for which of its neighbors it is a parent.

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 $\tilde{O}\left(\frac{|P_i|}{D}\right)$ 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 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 $\tilde{O}\left(\frac{|P_i|}{D}\right)$ sub-parts in P_i. A union bound shows this holds for all parts w.h.p. □

5.2 Computing Shortcuts with Randomization

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 $< 3b$

Set all P_i *active*.
for $O(\log n)$ iterations **do**
  Run CoreFast [19] shortcut construction algorithm on representatives in active parts.
  Set every P_i with block parameter $< 3b$ on CoreFast result *inactive* (see Lemma 4.5).
  Let every newly inactive P_i use the shortcut edges assigned to it by CoreFast.

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. We first argue runtime and message complexity. Haeupler, Izumi, and Zuzic [19, Lemma 4] show that CoreFast takes $O(D \log n + c)$ rounds. However, in this algorithm every node potentially sends a message up T once leading to super-linear message complexity. By amending CoreFast so only the $\tilde{O}\left(\frac{n}{D}\right)$ sub-part representatives send a message up T once as we do, it is easy to see that the algorithm uses only $\tilde{O}(n)$ messages total. Lastly, Lemma 4.5 shows that block parameter with randomization uses only $\tilde{O}(bD + c))$ and $\tilde{O}(m)$ messages.

We now argue correctness. Haeupler, Izumi, and Zuzic [19, Lemma 4] show that each time CoreFast is run, it computes a T-restricted shortcut with block parameter at most $3b$ for at least half of the nodes and congestion at most $8c$. It is easy to see that only having sub-part representatives participate in CoreFast does not affect correctness and so we conclude that after $O(\log n)$ iterations every P_i has been rendered inactive. By construction every P_i has block parameter $< 3b$ and since the congestion of any edge increases by at most $8c$ in any iteration of Algorithm 4, the total congestion of our returned shortcut is $\tilde{O}(c)$. □

6 Deterministic Subroutines

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

6.1 Computing Star Joinings Deterministically

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)

Algorithm 5 Deterministic star joining.

Input: $(P_i)_{i=1}^N$ s.t. $v \in P_i$ knows edge e_i 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: $G' \leftarrow \left(((P_i)_{i=1}^N), \{e_i\}_{i=1}^N\right)$
3: $\mathcal{R} \leftarrow \mathcal{R} \cup \{P_i : \delta^-_{G'}(P_i) \geq 2\}$ by running $\mathcal{A}$.
4: $\mathcal{J} \leftarrow \mathcal{J} \cup \{P_i : P_i \notin \mathcal{R} \wedge e_i = (P_{i'}, P_i)$ s.t. $P_{i'} \in \mathcal{R}\}$ by running $\mathcal{A}$.
5: $G' \leftarrow G' \setminus (\mathcal{R} \cup \mathcal{J})$.
6: Run the 3-coloring algorithm of Cole and Vishkin [4] on G' .
7: **for** color $k = 1, 2, 3$ **do**
8: $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{P_i : P_i$ colored $k\}$ by running $\mathcal{A}$.
9: $\quad \mathcal{J} \leftarrow \mathcal{J} \cup \{P_i : P_i \notin \mathcal{R} \wedge e_i = (P_{i'}, P_i)$ s.t. $P_{i'} \in \mathcal{R}\}$ by running $\mathcal{A}$.

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.

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. We begin by proving correctness. Line 4 yields stars of joiners centered around receivers. Moreover, the union of all nodes designated in Line 4 from a forest with trees of internal degree at least 2. Therefore, the number of internal (marked) super-nodes (and therefore the number of stars) in Line 4 is at most one half of the super-nodes of the tree.

Now consider the result of Line 8. As no super-node in G' has in-degree at least two at this point in the algorithm, the super-graph considered in Line 8 consists of directed cycles and paths. Thus, each time we remove a P_i from the super-graph we remove at most three super-nodes from the graph and P_i gets to merge with its neighbor. It follows that at least $\frac{1}{3}$ of these super-nodes are merged. Combining the first and second stage, we find that the super-nodes are combined into stars, where the number of obtained nodes is less than 2/3 of the original nodes. Therefore the above algorithm computes a star joining.

We now argue that our algorithm requires $O(\log^* n)$ runs of $\mathcal{A}$. This clearly holds for all subroutines of our algorithm except for Line 6. In particular, we must argue how the Cole-Vishkin algorithm can be efficiently simulated on our super-graph using $O(\log^* n)$ runs of $\mathcal{A}$. We repeat the following $O(\log^* n)$ times. Let l_i be the known leader of P_i. Each P_i begins with the color of l_i's ID. Next, the node in P_i incident to the edge chosen by P_i routes the color it received to l_i using $\mathcal{A}$. Then, l_i performs the Cole-Vishkin computation and then broadcasts V_i's new color to all nodes in P_i using $\mathcal{A}$. □

6.2 Computing Sub-Part Divisions Deterministically

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

for part P_i **do**
  $\mathcal{I}_i \leftarrow \{\{v\} : v \in P_i\}$ ▷ Initialize incomplete sub-parts
  $\mathcal{C}_i \leftarrow \{\}$ ▷ Initialize complete sub-parts
  for $O(\log n)$ iterations **do**
    for $F_j \in \mathcal{I}_i$ **do**
      if $\exists$ edge $(u, v) \in F_j \times (\bigcup_{F_{j'} \in \mathcal{I}_i} F_{j'} \setminus F_j)$ **then**
        $e_j \leftarrow e$ for such edge $e = (u, v)$ using PA.
      else
        $e_j \leftarrow e$ for some edge $e = (u, v) \in F_j \times (\bigcup_{F_{j'} \in \mathcal{C}_i} F_{j'})$ using PA.
    Run Algorithm 5 on $\mathcal{I}_i$ sub-parts with edges $\{e_j\}$ to compute a star-joining.
    for Joiner F_j with edge $e_j = (u, v)$ with $v \in F_{j'}$ **do**
      F_j merges with $F_{j'}$ using PA. ▷ if $F_{j'} \in \mathcal{C}_i$, update $\mathcal{C}_i$ accordingly.
      u remembers v as its parent.
      F_j orients its tree edges to v using PA.
    $\mathcal{C}'_i \leftarrow \{F_j \in \mathcal{I}_i : |F_j| \geq D\}$ using PA.
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \mathcal{C}'_i$.
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.*

Proof. Round and message complexities are trivial apart from the fact that we must show that PA can be solved within our bounds on incomplete sub-parts. However, notice that an incomplete sub-part has fewer than D nodes by definition along with a spanning tree in which every node knows its parent; as such aggregating within each incomplete sub-part is trivially achievable with $O(D)$ rounds and $O(m)$ messages.

We now argue correctness. Correctness for parts of fewer than D nodes is trivial. Consider parts of more than D nodes. Sub-parts continue to merge until they are complete and have at least D nodes and so our division clearly has $\tilde{O}\left(\frac{P_i}{D}\right)$ sub-parts. It remains to show that every complete sub-part's spanning tree has diameter $\tilde{O}(D)$. When a complete sub-part results from two incomplete sub-parts joining, its spanning tree has diameter at most $2D$. Call these nodes the *core* of the complete sub-part. When an incomplete sub-part F_j – which has spanning tree with diameter at most D since it has fewer than D nodes by definition – joins a complete sub-part, it necessarily joins by way of nodes in the core. Thus, any node in F_j is within $3D$ of any node in the core by

way of the resulting sub-part's spanning tree. Similarly, any other subsequent incomplete sub-part that joins the complete sub-part will be within $4D$ of any nodes in F_j by way of the associated spanning tree. Thus, every complete sub-part has spanning tree with diameter at most $4D$. □

6.3 Computing Shortcuts Deterministically

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 [39].

Definition 6.5 (Heavy Path Decomposition [39])**.** *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 $\lfloor \log_2 n \rfloor$ 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, $v = 1, 2, \ldots$ (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 $S : V \to 2^{(P_j)_{j=1}^N}$;
Desired congestion c
Output: Mapping $S_f : V \to 2^{(P_j)_{j=1}^N}$

```
for v ∈ V do
    Set S_0(v) ← S(v).
for i = 0, 1, 2, ..., log_2 D − 1 do
    for every node v ≡ 2^i mod 2^{i+1} do
        if |S_i(v)| ≥ 2c then
            Break (v, v + 1) and set S_i(v) ← ∅.
        else
            u ← v + 2^i
            if no broken edges between v and u then
                Transmit S_i(v) from v to u along P.
                Set S_{i+1}(u) ← S_i(u) ∪ S_i(v).
return S_f = S_{log_2 D}.
```

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

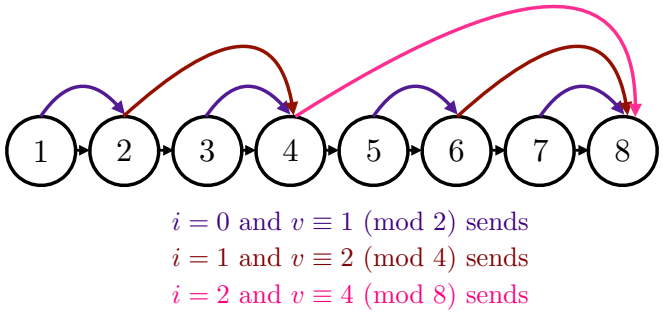


Figure 5: Illustration of Algorithm 7. Source of colored edge gives v that transmits $S_i(v)$ and sink gives u that updates $S_{i+1}(u)$ (assuming no edges broken). Black edges give path edges.

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 $i = 0, 1, \ldots, \log_2 D - 1$, 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 $u \equiv 0 \mod 2^{i+1}$. The congestion on this edge increases by $|S_i(v)|$ where $v \equiv 2^i \mod 2^{i+1}$. 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. □

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. We first prove the runtime and message complexity. As mentioned above, a heavy path decomposition of T can be computed in $O(D)$ rounds using $O(n)$ messages. We now bound the message and round complexity of each iteration. First note that we can inform every path if it has a light edge in $O(D)$ rounds with $O(n)$ messages. Next, running Algorithm 7 $O(\log n)$ times – once on each heavy path – requires $O(c \log D \log n + D \log n)$ rounds and $O(n \log n)$ messages. Lastly, notice that informing every node in a path that the path is now active requires $O(D)$ rounds using $O(n)$ messages. Lastly, by Lemma 4.5, running Algorithm 7 is within our stated bounds. Summing over iterations, we conclude that the overall round and message complexities are $\tilde{O}(b(c + D))$ and $\tilde{O}(m)$ respectively.

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** $j = 1, 2, \ldots, O(\log n)$ **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: Let S_f be the output of *Algorithm* 7 run on all active heavy paths.
12: For active path sink node v and light edge (v, u) let $S_j(u) = S_{j-1}(u) \cup \bigcup_v S_f(v)$.
13: Set all active paths inactive and all heavy paths with source u as in Line 12 active.
14: Set parts with block parameter $< 3b$ *inactive* (see Lemma 4.5).
15: **return** $\cup_j S_j(v)$ as v's shortcut edges

We now prove correctness. Notice that by Lemma 6.6 the number of parts assigned to an edge in any particular iteration is at most $O(c \log D)$ and so the overall congestion on any edge is at most $O(c \log D \log n) = \tilde{O}(c)$.

We now analyze the block parameter. In particular, we argue that the number of active parts is at least halved in each iteration. Let A_j be the set of active parts in iteration j. Let U_j be the set of heavy edges used by $\mathcal{H}$ but broken in iteration j and therefore not assigned to any parts by S_j. Each edge in U_j received at least $2c - c = c$ more requests by parts to use it than in $\mathcal{H}$. Thus each edge in U_j receives at least $2c - c = c$ requests from parts in A_j. However, each part in A_j can contribute at most b such additional requests to a broken edge, as each block can only send one additional request towards the tree's root. Consequently, we have $|U_j| \leq A_j \frac{b}{2c}$. Next, we say an active part is *bad* in iteration j if more than $2b$ of its edges of $\mathcal{H}$ are broken in iteration j, and *good* in iteration j otherwise. Note that for a good part the number of blocks in the output shortcut is at most $3b = O(b)$. On the other hand, every broken heavy edge used in $\mathcal{H}$ is used at most c times in $\mathcal{H}$. We conclude that the number of bad active parts is at most $|U_j| \frac{c}{2b}$.

Combining both upper and lower bounds on $|U_j|$, the number of bad parts active parts is at most $A_j/2$ in iteration j. Thus, after $O(\log n)$ iterations all parts will be marked inactive, meaning the block parameter in the returned shortcut is at most $3b$. □

References

[1] Baruch Awerbuch. 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)*, pages 230–240, 1987.

[2] Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2): 238–256, 1990.

[3] F Chin and HF Ting. 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)*, pages 257–266, 1985.

[4] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[5] 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 (SICOMP)*, 41(5):1235–1265, 2012.

[6] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *Journal of Computer and System Sciences*, 72(8):1282–1308, 2006.

[7] Michael Elkin. An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. *SIAM Journal on Computing (SICOMP)*, 36(2): 433–456, 2006.

[8] Michael Elkin. 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. A linear-time optimal-message distributed algorithm for minimum spanning trees. *Distributed Computing*, 17(2):151–170, 2004.

[10] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1162, 2012.

[11] Eli Gafni. Improvements in the time complexity of two message-optimal election algorithms. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 175–185, 1985.

[12] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.

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

[14] Mohsen Ghaffari. Near-optimal distributed approximation of minimum-weight connected dominating set. In *Proceedings of the 41nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 483–494, 2014.

[15] Mohsen Ghaffari and Bernhard Haeupler. 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)*, pages 202–219, 2016.

[16] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, pages 1–15, 2013.

[17] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 81–90, 2015.

[18] Bernhard Haeupler and Jason Li. Beating $\mathcal{O}(\sqrt{n} + D)$ for distributed shortest path approximations via shortcuts. *arXiv preprint arXiv:1802.03671*, 2018.

[19] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 451–460, 2016.

[20] Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 158–172, 2016.

[21] Bernhard Haeupler, Jason Li, and Goran Zuzic. Minor excluded network families admit fast distributed algorithms. *arXiv preprint arXiv:1801.06237*, 2018.

[22] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

[23] Taisuke Izumi and Roger Wattenhofer. Time lower bounds for distributed distance oracles. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 60–75, 2014.

[24] David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 648–657, 1994.

[25] Maleq Khan and Gopal Pandurangan. A fast distributed approximation algorithm for minimum spanning trees. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, pages 355–369, 2006.

[26] Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 238–251, 1995.

[27] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *Journal of the ACM (JACM)*, 62(1):7, 2015.

[28] Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 153–162, 2015.

[29] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 375–382, 2013.

[30] Ali Mashreghi and Valerie King. Time-communication trade-offs for minimum spanning tree construction. In *Proceedings of the 18th International Conference on Distributed Computing and Networking (ICDCN)*, 2017.

[31] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–203, 2013.

[32] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 565–573, 2014.

[33] Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 439–453, 2014.

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

[35] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. A time- and message-optimal distributed algorithm for minimum spanning trees. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 743–756, 2017.

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

[37] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing (SICOMP)*, 30(5):1427–1442, 2000.

[38] Lucia D Penso and Valmir C Barbosa. A distributed algorithm to find k-dominating sets. *Discrete Applied Mathematics*, 141(1-3):243–253, 2004.

[39] Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[40] Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.

[41] Ramakrishna Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. *Journal of Algorithms*, 23(1):160–179, 1997.

Appendix

A Applications of Our PA Algorithms

Here we outline the applications of our round- and message-optimal PA algorithms for multiple distributed graph problems. We start with a discussion of the problems referred to in Section 1.2, and then discuss additional applications of our PA algorithm to optimization as well as verification problems, in Appendix A.2.

A.1 Deferred Proofs of Section 1.2

Here we address how to apply our PA algorithm to solve MST, Approximate Min-Cut and Approximate SSSP, starting with a formal definition of these problems.

We now restate the round and message complexities we obtain for these problems using our new PA algorithm and discuss the algorithms used to achieve these bounds. As before, recall that since every graph admits a shortcut with $b = 1$ and $c = \sqrt{n}$, our algorithms simultaneously achieve worst-case optimal $\tilde{O}(m)$ message complexity and worst-case optimal $\tilde{O}(D+\sqrt{n})$ round complexity.

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.*

Proof. Our algorithm uses Theorem 1.2 to simulate Borůvka's classic MST algorithm [34]. In Borůvka's algorithm every node initially belongs to its own part. Then, for $O(\log n)$ rounds each part merges with the part it is connected to by its minimum outbound edge. The problem of determining the minimum-weight outbound edge of a part is an example of Part-Wise Aggregation, which we solve with the algorithm of Theorem 1.2. Whenever a node v is incident to this edge it remembers that the neighbor along that edge is now in the same part as v. Round and message complexities are trivial and correctness follows from that of Borůvka's algorithm. □

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.*

Proof. Section 5.2 of Ghaffari and Haeupler [15] provides an algorithm for approximate min-cut based on shortcuts. The algorithm works roughly as follows: use sampling ideas from Karger [24] to downsample edge weights so that the min-cut is of size $O(\log n/\epsilon^2)$; by Thorup [40], we can now solve MST $O(\log n) \cdot \text{poly}(1/\epsilon)$ times (using certain different weights each time) such that there is one edge e^* in one of our MSTs T^* such that the two connected components of $T^* \setminus e^*$ define an approximately optimal min-cut; lastly, using a sketching approach this edge can be found by solving PA $\text{poly}\log(n) \cdot \text{poly}(1/\epsilon)$ times with high probability. See [15] for a proof of correctness. Our claimed round and message complexities are trivial given Corollary 1.3 and Theorem 1.2, as the above algorithm requires downsampling edge weights (only requiring $O(1)$ rounds and $O(m)$ messages) and by Corollary 1.3 and Theorem 1.2, the $\text{poly}\log n \cdot \text{poly}(1/\epsilon)$ instances of MST and PA can be solved with $\tilde{O}(bD+c) \cdot \text{poly}(1/\epsilon)$ rounds and $\tilde{O}(m) \cdot \text{poly}(1/\epsilon)$ messages. □

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.*

Proof. Haeupler and Li [18] provide a distributed algorithm with the stated round complexity and approximation factor based on a solution to PA. Roughly the algorithm works as follows. We compute $O(\frac{\log n}{\beta})$ low diameter decomposition; in particular, as in Miller et al. [31] every node starts a weighted BFS at a randomly-chosen time and runs this weighted BFS for $O(\frac{\log n}{\beta})$ rounds. During this weighted BFS, nodes claim as part of their ball any nodes they reach that have not yet been claimed or started their weighted BFS. The weights used for the weighted BFS change in each round: any weight strictly inside a claimed ball is updated to have weight 0 and any edge incident to two claimed balls is additively increased. Moreover, the increments used by the weighted BFS geometrically increase in each iteration so that despite increasing edge weights, the weighted BFS can still efficiently proceed. Lastly, the union of all BFS trees returned by every weighted BFS is returned as a tree, T^*, that approximates distance in the graph; to inform nodes of their approximate distance from the source, the source can simply broadcast on T^*.

What makes this algorithm difficult to implement is that running a weighted BFS with edge weights set to 0 requires that a weighted BFS traverse components connected by weight-zero edges of potentially large diameter "in a single round". To overcome this issue, Haeupler and Li [18] use PA to efficiently traverse these connected components. For more see Haeupler and Li [18]. Relying on our algorithm for PA and observing that these dominate the round and message complexity of the weighted BFS calls, our claimed round and message complexities follow. □

A.2 Further Applications of Our PA Algorithms

Here we mention some further applications of PA, all of which prior work has show to have PA as their round and communication bottleneck. For all of these problems our new PA algorithm of Theorem 1.2 therefore yields $\tilde{O}(D+\sqrt{n})$-round and $\tilde{O}(m)$-message algorithms.

Graph Verification Problems. In [5], Das Sarma et al. provided an extensive list of lower bounds for optimization problems (many of which we referred to throughout this paper, as their lower bounds prove our algorithms' round complexity to be optimal). Das Sarma et al. further showed that their lower bounds carry over to *verification* problems. For these problems, the input is a graph G and a subgraph H of G and an algorithm must verify whether this subgraph satisfies some property, such as whether H is a spanning tree or H is a cut (see [5] for more).

Das Sarma et al. also provided algorithms for this long list of verification problems, relying heavily on an optimal MST algorithm and the following connected component algorithm of Thurimella [41, Algorithm 5]. Thurimella's algorithm, given a graph G and subgraph H as above, outputs a label $\ell(v)$ for each vertex $v \in V(G)$ such that $\ell(u) = \ell(v)$ if and only if u and v are in the same connected component of H. The problem solved by Thurimella is easily cast as an instance of PA, by having each part elect a leader in a connected component in H – say, a node of minimum ID – and use the leader's ID as a label.[4] Without repeating the arguments of Das Sarma et al. [5],

[4]We note that for bipartiteness verification, Das Sarma et al. relied on the algorithm of [41] also outputting a rooted spanning tree of each connected component of H with each vertex knowing its level in the tree. As our PA algorithm maintains such rooted spanning trees, it can also be used to solve this verification problem within the same bounds.

we note that as MST and Thurimella's algorithm require $\tilde{O}(D + \sqrt{n})$ rounds (based on [26]), so do the algorithms of Das Sarma et al., and this is tight for these verification problems by their lower bounds. Our MST and PA algorithms show that for the long list of verification problems Das Sarma et al. considered, optimal round complexity does not preclude optimal message complexity, as we can attain both simultaneously.

Corollary A.1. *All the graph verification problems considered in [5, Section 8] can be solved in an optimal $\tilde{O}(D + \sqrt{n})$ rounds and $\tilde{O}(m)$ messages.*

Approximation of Minimum-Weight Connected Dominating Set Another application of our PA algorithms follows from the work of Ghaffari [14]. In that work, Ghaffari shows that the algorithm of Thurimella [41], discussed above, can be used to to compute an $O(\log n)$-approximate minimum weight connected dominating set (a set of nodes S such that all vertices in G are at distance at most one from a node in S). In particular, Ghaffari relied on the ability to extend Thurimella's algorithm to the case where nodes also have some value $x(v)$ assigned to them so that the label of nodes in a connected component can be equal to: (A) the list of the $k = O(1)$ largest values $x(v)$ in the component, or (B) the sum of values $x(v)$ in the component. Both these applications can be cast as instances of PA. Plugging in our PA algorithm into Ghaffari's algorithm [14], we obtain the following.

Corollary A.2. *There exists an $\tilde{O}(D + \sqrt{n})$-round, $\tilde{O}(m)$-message algorithm which computes an $O(\log n)$-approximate minimum weight connected dominating set.*

k-dominating sets. Another distributed primitive used in multiple distributed graph algorithms is the problem of computing an $O(n/k)$-node k-dominating set. That is, a set of nodes S such that every node in G is at distance at most k from some node in S. This problem has found applications in distributed algorithm for MST [26] and $(1+\epsilon)$-approximate eccentricity computation [22]. For this problem $\tilde{O}(k)$-round algorithms are known, [26]), including some with linear message complexity [38]. However, for large k, i.e. $k \gg \max\{D, \sqrt{n}\}$, no $\tilde{O}(D + \sqrt{n})$-round, $\tilde{O}(m)$-message algorithm was known. Such an algorithm follows immediately from a simple generalization of our sub-part division algorithm, as follows. As in Algorithm 6, we repeatedly merge sub-parts, marking a sub-part as complete when it attains some size. Unlike the above algorithm, this threshold is chosen to be $k/6$ rather than D. By arguments which are a simple generalization of Algorithm 6's analysis, this implies that the obtained sub-parts have diameter at most k, and that each sub-part contains at least $k/6$ nodes, one of which is its representative. This set of representatives therefore has cardinality at most $6n/k$ and it forms a k-dominating set. The one delicate point to notice is that now we can solve Part-Wise Aggregation using our round- and message-optimal algorithm for PA – unlike in Algorithm 6, which is a sub-routine in our PA algorithm. In particular, even if $k \gg \max\{D, \sqrt{n}\}$, this can be done in $\tilde{O}(D+\sqrt{n})$ rounds (and $\tilde{O}(m)$ messages). Therefore, as each of the $O(\log n)$ iterations of this algorithm can be implemented using some $O(\log^* n)$ many calls to PA (by Lemma 6.3), together with some local computation, we obtain the following corollary of Theorem 1.2.

Corollary A.3. *For any integer k, there exists an $\tilde{O}(D + \sqrt{n})$-round, $\tilde{O}(m)$-message algorithm which computes a k-dominating set of size $O(n/k)$.*

This last corollary bolsters our confidence that the ubiquitous $\tilde{\Theta}(D+\sqrt{n})$ bounds for distributed graph algorithms can be matched with $\tilde{O}(m)$ messages for an even wider range of problems not discussed in this paper.

B Dispensing with Known Leader Assumption

Throughout this paper we have assumed that parts always know a leader. That is for every part P_i every node $v \in P_i$ knows the ID of some leader $l_i \in P_i$. We solved PA assuming that this holds. We now show that this assumption can dispensed with. In particular, we demonstrate that an algorithm that solves PA with the assumption of a known leader for each part can be converted into one that makes no such assumption with only logarithmic overhead in round and message complexities. The conversion is deterministic and so it demonstrates that a known leader is not required for either or deterministic or our randomized results.

Our algorithm is Algorithm 9 and works as follows: start with the singleton partition where every node is its own leader; repeatedly coarsen this partition $O(\log n)$ times until it matches the input PA partition by applying our PA solution that assumes that a leader is known to merge the stars given by a star joining. At each step in the coarsening we maintain the invariant that every part knows a leader and so in the end we need only solve PA with a known leader which we can do by assumption.

Algorithm 9 PA without leaders.

Input: PA instance with parts $(P_i)_{i=1}^N$;
PA algorithm, $\mathcal{A}$, that assumes every part knows a leader.
Output: a solution to the input PA problem.

for all $i \in |V|$ **do**
  Set $P_i' \leftarrow \{i\}$ and $l_i \leftarrow i$. ▷ Each P_i' maintains a leader l_i, initially set to i
for $O(\log n)$ rounds **do**
  for all part P_i' **do**
    Pick some $e_i = (u, v) \in P_i' \times (V \setminus P_i')$ by running $\mathcal{A}$ where u, v are in the same P_i.
  Compute a star joining with Algorithm 5 over the P_i's using edges $\{e_i\}$.
  for all part P_i' which joined P_j' in the star joining **do**
    Inform each $v \in P_i'$ that their leader is now l_j by running $\mathcal{A}$.
    Merge P_i' into P_j'.
Run $\mathcal{A}$ on the PA instance consisting of the P_i's, each with a known leader l_i.

Lemma B.1. *Given a PA instance where no leaders are known and a PA algorithm, $\mathcal{A}$, that assumes leaders are known using R rounds and M messages, Algorithm 9 solves the PA instance with no leaders in $\tilde{O}(R)$ rounds and $\tilde{O}(M)$ messages.*

Proof. We first prove round and message complexities. Our algorithm runs $\mathcal{A}$ to solve PA with a known leader and Algorithm 5 to compute a star-joining logarithmically many times. The latter consists of $O(\log^* n)$ calls to $\mathcal{A}$. Thus, the stated round and message complexities follow trivially.

We now argue correctness. Each round a constant fraction of the P_j's participating in the algorithm get to merge by definition of a star joining and so $O(\log n)$ repetitions are sufficient to

coarsen every P'_j to a P_i. Moreover, $P'_1, \ldots, P'_{N'}$ is valid input to $\mathcal{A}$ since we maintain the invariant that every node in a P'_j has an elected leader. At the end of this coarsening our PA instance now has elected leaders. By the correctness of $\mathcal{A}$ our algorithm is correct. □

C Our Results In Tabular Form

Throughout the paper we state our results in utmost generality by giving our algorithms' running time in terms of the optimal block parameter b and congestion c. As stated in Theorem 1.2 and its Corollaries 1.3, 1.4 and 1.5 and Appendix A, for PA and the wide range of application problems we consider, our deterministic algorithms terminate in $\tilde{O}(b(D+c))$ rounds and our randomized algorithms terminate in $\tilde{O}(bD+c)$ rounds. To make these bounds more concrete, we review some known bounds on the parameters b and c in Table 1, and then state the implied running times of all our algorithms for the above problems in Table 2.[5]

	General [15]	Planar [15]	Genus g [19]	Treewidth t [20]	Pathwidth p [20]	Minor Free [21]
b	1	$O(\log D)$	$O(\sqrt{g})$	$O(t)$	p	$\tilde{O}(D)$
c	$\sqrt{n}$	$\tilde{O}(D)$	$\tilde{O}(\sqrt{g}D)$	$\tilde{O}(t)$	p	$\tilde{O}(D)$

Table 1: Known bounds on block parameter, b, and congestion, c.

	General	Planar	Genus g	Treewidth t	Pathwidth p	Minor Free
Deterministic	$\tilde{O}(D+\sqrt{n})$	$\tilde{O}(D)$	$\tilde{O}(gD)$	$\tilde{O}(tD+t^2)$	$\tilde{O}(pD+p^2)$	$\tilde{O}(D^2)$
Randomized	$\tilde{O}(D+\sqrt{n})$	$\tilde{O}(D)$	$\tilde{O}(\sqrt{g}D)$	$\tilde{O}(tD)$	$\tilde{O}(pD)$	$\tilde{O}(D^2)$

Table 2: Summary of running times of our algorithms.

Reviewing Table 2, we note that for all problems considered, a matching worst case round lower bound of $\tilde{\Omega}(D+\sqrt{n})$ is given by Das Sarma et al. [5], while a trivial lower bound of $\Omega(D)$ holds for these problems for all graphs. Our algorithms match the worst case bounds and the $\Omega(D)$ lower bound (up to polylog terms) for any constant genus, treewidth and pathwidth, all while requiring only $\tilde{O}(m)$ messages. The exact optimal dependence on the parameters g, t and p remains an open question.

[5]The two exceptions to this rule are our L^ϵ-approximate SSSP and $(1+\epsilon)$-approximate Min-Cut, for which the (randomized) bounds hold as stated in the table only for fixed (or polylogarithmic) ϵ.