Faster Distributed Shortest Path Approximations via Shortcuts

Bernhard Haeupler[1]
Carnegie Mellon University, USA
http://cs.cmu.edu/~haeupler

Jason Li
Carnegie Mellon University, USA
http://cs.cmu.edu/~jmli

Abstract

A long series of recent results and breakthroughs have led to faster and better distributed approximation algorithms for single source shortest paths (SSSP) and related problems in the CONGEST model. The runtime of all these algorithms, however, is $\tilde{\Omega}(\sqrt{n})$, regardless of the network topology[2], even on nice networks with a (poly)logarithmic network diameter D. While this is known to be necessary for some pathological networks, most topologies of interest are arguably not of this type.

We give the first distributed approximation algorithms for shortest paths problems that adjust to the topology they are run on, thus achieving significantly faster running times on many topologies of interest. The running time of our algorithms depends on and is close to Q, where Q is the quality of the best shortcut that *exists* for the given topology. While $Q = \tilde{\Theta}(\sqrt{n} + D)$ for pathological worst-case topologies, many topologies of interest[3] have $Q = \tilde{\Theta}(D)$, which results in near *instance optimal* running times for our algorithm, given the trivial $\Omega(D)$ lower bound.

The problems we consider are as follows:

- an approximate shortest path tree and SSSP distances,
- a polylogarithmic size distance label for every node such that from the labels of any two nodes alone one can determine their distance (approximately), and
- an (approximately) optimal flow for the transshipment problem.

Our algorithms have a tunable tradeoff between running time and approximation ratio. Our fastest algorithms have an arbitrarily good polynomial approximation guarantee and an essentially optimal $\tilde{O}(Q)$ running time. On the other end of the spectrum, we achieve polylogarithmic approximations in $\tilde{O}(Q \cdot n^{\epsilon})$ rounds for any $\epsilon > 0$. It seems likely that eventually, our non-trivial approximation algorithms for the SSSP tree and transshipment problem can be bootstrapped to give fast $Q \cdot 2^{O(\sqrt{\log n \log\log n})}$ round $(1+\epsilon)$-approximation algorithms using a recent result by Becker et al.

2012 ACM Subject Classification Theory of computation → Shortest paths, Theory of computation → Distributed algorithms, Theory of computation → Approximation algorithms analysis

Keywords and phrases Distributed Graph Algorithms, Shortest Path, Shortcuts

Digital Object Identifier 10.4230/LIPIcs.DISC.2018.33

Related Version A full version of the paper is available at https://arxiv.org/abs/1802.03671.

[1] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.
[2] We use $\tilde{\ }$-notation to hide polylogarithmic factors in n, e.g., $\tilde{O}(f(n)) = O(f(n)\log^{O(1)} n)$.
[3] For example, [8] and [10] show that large classes of interesting network topologies, including planar networks, bounded genus topologies, and networks with polylogarithmic treewidth have shortcuts of quality $Q = \tilde{O}(D)$. A similar statment is likely to hold for any minor closed graph family [11].

1 Introduction

This paper gives new distributed approximation algorithms for computing single source shortest path (SSSP) distances and various generalizations, such as computing a SSSP tree, distance labels, and a min-cost uncapacitated flow.

In the last few years, CONGEST algorithms for shortest path problems have seen a tremendous amount of interest and progress [5, 12, 17]. The main difference of the algorithms developed here, compared to those works, is that our algorithms achieve significantly faster running times for non-pathological network topologies by building on the recently developed [8, 9] low-congestion shortcut framework; for a detailed overview, see Appendix A of the full version on arXiv.

The low-congestion shortcut framework leads to faster algorithms for optimization problems with simple parallel divide and conquer style algorithms, such as the minimum spanning tree problem. However it initially seemed less applicable to shortest path problems, particularly because all previous approaches for CONGEST algorithms for these problems led to $\Omega(\sqrt{n})$ running times, for reasons that are independent of issues where shortcuts can help. Indeed, our approach for achieving non-trivial approximation ratios for shortest path problems deviates notably from these approaches, and uses different tools to obtain non-trivial approximation guarantees.

This paper is organized as follows: We briefly summarizes the key technical concepts of the shortcut framework in Section 1.1; a more detailed treatment of the framework is given in Appendix A in the full version. In Section 1.2, we define the different problems we treat in this paper, and explain the difficulties in beating the $\tilde{\Omega}(\sqrt{n} + D)$ barrier for approximating shortest path distances. We state our results in Section 1.3, compare it to related works in Section 1.4, and devote the remaining paper to describing our algorithms and proving them correct.

1.1 The Low-Congestion Shortcut Framework: A Brief Summary

This section provides the key technical definitions and facts about the low-congestion shortcut framework. However, it does not attempt to explain the reasons, generality or importance behind the definitions given here. Appendix A of the full version gives a more detailed treatment, and we highly recommend to readers not familiar with the low-congestion shortcut framework to read Appendix A first.

The shortcut framework is built around a simple and basic communication problem, given in the next two definitions:

▶ **Definition 1** (Valid Partitioning and Parts). For a graph $G = (V, E)$, we say that a collection of parts $S_1, S_2, \ldots \subset V$ is a **valid partition** if the parts are vertex disjoint and each induces a connected graph.

▶ **Definition 2** (The Part-wise Communication Problem). Let G be a network with a valid partitioning $S_1, S_2, \ldots$ and a value x_v for every node $v \in V$. Suppose $\oplus$ is an associative and commutative function. The **partwise communication problem** asks for every S_i and every $u \in S_i$ to compute the value $\bigoplus_{v \in S_i} x_v$.

We remark that for convenience, the parts of a valid partition do not necessarily need to contain every vertex in V. Alternatively, it can be convenient to think of each node in $V \setminus \bigcup_i S_i$ as forming its own single-vertex part, thus making any valid partitioning a partitioning in the usual sense.

The key findings of the shortcut framework can now be summarized as follows:

The shortcut framework allows us to characterize how hard it is to solve the part-wise communication problem described in Definition 2 in the CONGEST model for any given topology G. For any network with topology G this is captured by the quantity Q_G. In the worst-case, the value of Q_G is $\tilde{\Theta}(\sqrt{n}+D)$ for a network with n nodes and diameter D, such as the pathological network that shows a $\tilde{\Omega}(\sqrt{n}+D)$ lower bound for MST and related problems [19]. In many other networks of interest, including planar networks, networks which embed into a surface with bounded or polylogarithmic genus, networks with bounded or polylogarithmic tree-width or networks with small separators, the hardness Q_G is much lower and in fact only $\tilde{O}(D)$. Most importantly, whatever the hardness Q_G of a given topology is, there is a simple distributed algorithm which solves the part-wise communication problem in $\tilde{O}(Q_G)$ rounds for any valid partitioning in G. Thus, $\tilde{O}(Q_G)$ round shortcut-based algorithms necessarily have a worst-case running time of $\tilde{O}(\sqrt{n}+D)$ when expressed in terms of n and D; however, they are essentially running as fast as the given topology (and to some extent even the given input) allows it, which in many cases of interest is significantly faster, e.g., $\tilde{O}(D)$ rounds.

1.2 CONGEST model and Shortest Path Problems

1.2.1 CONGEST Model

We consider the classical CONGEST model of distributed computing where a network is given by a connected graph $G = (V, E)$ with n nodes and (hop-)diameter D. Communication proceeds in synchronous rounds. In each round, each node can send a different $O(\log n)$ bit message to each of its neighbors. Local computations are free and require no time. Nodes have no initial knowledge of the topology G, except that we assume that they know n and D up to constants (because these parameters can be computed in $O(D)$ time, which is negligible in our context). All of our algorithms are randomized and succeed with high probability[4]. In particular, we assume that each node has access to a private string of randomness, which it can also use to create an $O(\log n)$ bit ID that is unique w.h.p.

In all problems considered here, we assume that every edge e of the network G has a length or cost $w(e)$ associated with it. We assume that all lengths lie in the range $[1, n^C]$ for some constant C, and are initially only known to nodes adjacent to an edge. Interestingly, our algorithms also easily handle edges of length zero, but for sake of simplicity, we do not consider such edges in this paper. Any such length or cost function w produces a weighted graph which we call $G(w)$, and induces a distance between any two nodes $u, v \in V$, which we denote with $d_{G(w)}(u, v)$, or simply $d(u, v)$ when the weighted graph $G(w)$ is clear. We denote the weighted diameter of a network with $L = \max_{u,v} d_G(u, v)$.

1.2.2 Shortest Path Problems

The most important and most basic problem we are studying in this paper is the single source shortest path problem:

[4] Throughout this work, "with high probability" or w.h.p. means with probability at least $1 - n^{-C}$ for any desired constant C.

▶ **Definition 3.** The **α-approximate SSSP distance problem** assumes as input a weighted graph and a designated source node $s \in V$, and asks for every node $v \in V$ to compute an approximate distance d_v which satisfies $d(s,v) \le d_v \le \alpha \cdot d(s,v)$.

We furthermore consider the following generalizations of the SSSP distance problem:

▶ **Definition 4.** The **α-approximate SSSP tree problem** assumes that a weighted graph with a designated source node $s \in V$ is given and asks to compute a subtree $T \subseteq G$ such that for every node $v \in V$ distance $d_T(s,v) \le \alpha \cdot d(s,v)$. Each node should know which of its adjacent edges belong to T.

▶ **Definition 5** (Approximate distance labeling scheme). An **$(l(n),\alpha)$-approximate distance labeling scheme** is a function that labels the vertices of an input graph with distinct labels up to $l(n)$ bits, such that there exists a polynomial time algorithm that, given the labels of vertices x and y, provides an estimate $\tilde{d}(x,y)$ for the distance between these vertices such that

$$\tilde{d}(x,y) \le d(x,y) \le \alpha \cdot \tilde{d}(x,y).$$

▶ **Definition 6** (Transshipment Problem). The **transshipment problem** is the problem of uncapacitated min-cost flow. In it every node in a weighted graph G has some real demand d_v such that $\sum_v d_v = 0$. The cost of routing x amount of flow over an edge e of weight $w(e)$ is $xw(e)$. The problem is to compute a flow satisfying all demands of approximate minimum cost. Each node should know the flow an all edges incident to it.

1.3 Our Results

1.3.1 SSSP

Our first result is on computing an approximate, single source shortest path tree in a distributed setting. Note that due to communication limits in the CONGEST model, it is infeasible for each vertex to know the entire shortest path tree. However, it is sufficient that each vertex computes the *local* structure of the tree, which is made specific below.

▶ **Theorem 7.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, with a specified source vertex, and let $\beta := (\log n)^{-\Omega(1)}$. There is a distributed algorithm that, w.h.p., runs for $\tilde{O}(\frac{1}{\beta} Q_G)$ rounds and outputs a spanning tree that approximates distances to the source to factor $O(L^{O(\log\log n)/\log(1/\beta)})$.*[5] *By output, we mean that at the end of the algorithm, every vertex knows its set of incident edges in the spanning tree.*

By setting $\beta := n^{-\epsilon}$, $\beta := 2^{-\Theta(\sqrt{\log n})}$, and $\beta := \log^{-\Theta(1/\epsilon)} n$ for constant ϵ, respectively, we obtain the following three corollaries:

▶ **Corollary 8.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, with a specified source vertex. For any constant $\epsilon > 0$, there is a distributed algorithm that, w.h.p., runs for $\tilde{O}(Q_G n^{\epsilon})$ rounds and outputs a spanning tree that approximates distances to the source to factor* $\text{polylog}(n)$.

▶ **Corollary 9.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, with a specified source vertex. There is a distributed algorithm that, w.h.p., runs for $\tilde{O}(Q_G 2^{O(\sqrt{\log n})})$ rounds and outputs a spanning tree that approximates distances to the source to factor $2^{O(\sqrt{\log n})}$.*

[5] Recall that $L = \max_{u,v} d_G(u,v)$.

▶ **Corollary 10.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, with a specified source vertex. For any constant $\epsilon > 0$, there is a distributed algorithm that, w.h.p., runs for $\tilde{O}(Q_G)$ rounds and outputs a spanning tree that approximates distances to the source to factor $O(L^\epsilon)$.*

1.3.2 Distance labeling schemes

For distance labeling schemes, we have the following result.

▶ **Theorem 11.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$. There exists a $(\text{polylog}(n), n^{O(\log\log n)/\log(1/\beta)})$ approximate distance labeling scheme that runs in $\tilde{O}(\frac{1}{\beta}Q_G)$ rounds.*

Setting $\beta := n^\epsilon$ gives the following corollary:

▶ **Corollary 12.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, There exists a $(\text{polylog}(n), \text{polylog}(n))$ approximate distance labeling scheme that runs in $\tilde{O}(Q_G n^\epsilon)$ rounds.*

1.3.3 Transshipment problem

We also provide a distributed algorithm to compute an approximate flow for the transshipment problem.

▶ **Theorem 13.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$ and demands that sum to zero, and let $\beta := (\log n)^{-\Omega(1)}$. There is an algorithm that, w.h.p., runs for $\tilde{O}(\frac{1}{\beta}Q_G)$ rounds and computes a $\tilde{O}(\frac{1}{\beta}n^{O(\log\log n)/\log(1/\beta)})$-approximate flow.*

1.4 Related Work

The complexity theoretic issues in the design of distributed graph algorithms for the CONGEST model have received much attention in the last decade, and extensive progress has been made for many problems: Minimum-Spanning Tree [13], Minimum Cut [18], Diameter [14], Shortest Path [5], and so on. Most of those problems have $\tilde{\Theta}(\sqrt{n} + D)$-round upper and lower bounds for some sort of approximation guarantee [19]. The notion of low-congestion shortcuts was invented as a framework of circumventing these lower bounds [8]. Specifically, the ideas present in [8] can be turned into very short and clean $\tilde{O}(D + \sqrt{n})$ round algorithms for general graphs, and near-optimal $\tilde{O}(D)$ round algorithms for special classes of graphs, for problems such as MST and Min-Cut.

However, the shortcut framework cannot be applied directly to the SSSP problem, since, unlike MST and Min-Cut, shortest path problems are not inherently parallelizable. For SSSP, a new technique based on multiplicative weights results in a $(1 + \epsilon)$-approximation to SSSP in $\tilde{O}(D + \sqrt{n})$ time on general graphs [5]. However, until this paper, not much work has been done on circumventing the $\tilde{\Omega}(D + \sqrt{n})$ lower bound on restricted classes of graphs or otherwise.

As a subroutine to computing shortest paths, we will be running low-diameter graph decompositions. Low diameter decompositions have a long history in the centralized [4, 15] and parallel [3, 6, 16] settings, and have been applied in the distributed setting to compute a network decomposition with low "chromatic number" [7].

2 Distance-Preserving Tree

Let G be a weighted graph with Q_G-quality shortcuts. For a reader not familiar with shortcuts or the material in Appendix A of the full version, the parameter Q_G intuitively measures how easy it is for connected components of G to communicate within each other. As a general rule, the "nicer" the graph G is, the smaller the quantity Q_G and the closer it gets to the optimal D. For example, if G is a planar graph, then $Q_G = \tilde{O}(D)$.

We first consider the problem of finding a tree such that, for every pair of vertices $x, y \in V$, their distance is well-approximated with constant probability. Our algorithm is an adaptation of the algorithm of Section 5.4 from [2].

To motivate the ideas behind the algorithm, we describe it in a parallel framework with graph contraction support. In each iteration, the algorithm runs a low diameter decomposition (defined below; see Appendix B of the full version for details) on the graph and contracts each component into a single vertex. To compute the tree as described above, take the set of edges inside the BFS trees formed by each LDD, and map them back to the original graph. The resulting tree is simply the (disjoint) union of these edges over all iterations. Of course, in a distributed framework, we cannot maintain contracted graphs, so we substitute each contracted vertex with a part of the original graph with zero-weight edges inside. To communicate efficiently between the parts, we establish shortcuts within each part.

▶ **Definition 14.** For a weighted graph $G = (V, E)$, a low-diameter decomposition (LDD) of G is a probabilistic distribution over partitions of V into connected components $S_1, \ldots, S_k$, such that

1. W.h.p., every induced graph $G[S_i]$ has low weighted diameter.
2. For every two vertices $x, y \in V$, the probability that they belong to the same component is bounded from below by some function depending on $d_G(x, y)$.

We now describe the algorithm in detail. For a weight function $w : E \to \mathbb{R}$, denote $G(w)$ to be the graph G whose edges are reweighted according to w. The algorithm maintains a weight function $w : E \to \{0\} \cup [R, \mathrm{poly}(n)]$ on the set of edges, for a given value R. The zero-weight edges connect vertices within each component, while the threshold R increases geometrically over time. With a larger threshold R, we can compute the LDD on $G(\frac{1}{R}w)$, allowing the LDD to travel farther in the same amount of time. If R is large enough, this graph still has edge weights at least 1 in between components, so computing the LDD is feasible in a distributed manner.

In addition to w, the algorithm also maintains a forest T, which gets new edges every iteration until it results in the approximate shortest path tree. Consider the following `LDDSubroutine`, which we apply iteratively to w and T.

```
Algorithm $(w', T')$ = LDDSubroutine($w, T, \beta, R$)
  Algorithm:
1. Initially, set $w' := w$ and $T' := T$.
2. Consider $G_0(w)$, the subgraph of $G$ with only the edges $e$ with $w(e) = 0$.
3. Let $H$ be the (multi-)graph with every connected component of $G_0(w)$ contracted
   to a single vertex. Denote $w_H$ as the function $w$ restricted to the edges in $H$.
4. Simulate a LDD on $H(\frac{1}{R}w_H)$ with parameter $\frac{1}{\beta}$ (see Appendix C of the full version).
   The specifics are deferred to the next section.
5. For every edge in $H$ that is part of a BFS tree in the LDD, add that edge to $T'$.
6. For every edge $e$ in $H$ completely inside a LDD component, set $w'(e) := 0$.
7. For every other edge $e$ in $H$, set $w'(e) := w(e) + \frac{c_1}{\beta}\log n$ (for large enough constant
   $c_1$).
8. Output $(w', T')$.
```

2.1 Correctness

The following two lemmas bound the maximum weighted diameter of a component, and therefore also the running time of the subroutine, as well as the probability that two vertices close together belong to the same component. Their proofs are natural generalizations of those in [16] and appear in Appendix C of the full version.

▶ **Lemma 15.** *W.h.p., each component in* `LowDiameterDecomposition` *has weighted diameter* $O(\frac{1}{\beta}\log n)$.

▶ **Lemma 16.** *For vertices* $u, v \in V$ *of (weighted) distance* d*, the probability that* u *and* v *belong to the same component is* $e^{-O(d\beta)}$.

We now describe in more detail how to simulate the LDD in $H(\frac{1}{R}w_H)$ in the desired running time. Observe that we cannot directly compute the LDD on the contracted graph, since the contracted vertices are actually entire parts with limited communication between them. However, we can apply shortcuts to communicate quickly within the parts, up to the quality of the shortcut.

▶ **Lemma 17.** *The LDD on the contracted graph (step 4 of* `LDDSubroutine`*) can be simulated with a* $\tilde{O}(Q_G)$ *multiplicative overhead in running time. In other words, if the LDD takes* d *rounds, then it can be simulated in* $\tilde{O}(Q_G d)$ *rounds in the network* G.

Proof. Define the parts of V to be the connected components of G, and compute a set of $\tilde{O}(Q_G)$-quality shortcuts, one for each part. In every round of the LDD on $H(\frac{1}{R}w_H)$, we perform two steps sequentially: one to traverse nonzero weight edges between parts, and one to flood through the zero weight edges within each part. To take care of the edges between parts, note that every such edge has weight at least 1, so we can send them directly through the network G. To flood through the zero edges within each part, it suffices to compute the minimum time t that is received by any vertex, and then broadcast the message "t" to the entire part. By routing through shortcuts, this can be done in $\tilde{O}(Q_G)$ time per partition. Overall, every round of the LDD is replaced by $\tilde{O}(Q_G)$ rounds in the network G, hence the multiplicative overhead. ◀

Together with Lemma 15, we get a running time of $\tilde{O}(\frac{1}{\beta}Q_G)$.

▶ **Definition 18.** Let $w : E \to \mathbb{R}$ be a weight function, and $T \subseteq G$ a forest. Define $G_0(w)$ to be the subgraph of G with only the edges e with $w(e) = 0$. Let $C_1, C_2, \ldots$ of G be the connected components of $G_0(w)$. We say that (w, T) satisfies the **subroutine invariant with parameter R** if the following conditions hold:

1. The weighted diameter of each part C_i using edge weights in G is at most R.
2. Every edge within a part C_i has weight 0 in w.
3. Every edge between two parts C_i, C_j has weight at least R in w.
4. For all x, y belonging to the same part C_i, $d_T(x, y) \leq R$.
5. T has a spanning tree within each part C_i, and no edges in between parts.

▶ **Lemma 19.** *Fix parameter* β*. Suppose that the input* (w, T) *to* `LDDSubroutine` *satisfies the subroutine invariant with parameter* R*. Then, w.h.p., for large enough constants* c_1 *and* c_2*,*

- *The output* (w', T') *satisfies the subroutine invariant with parameter* $(\frac{c_1}{\beta}\log n)R$.
- *For all* $x, y \in V$, $\mathbb{E}[d_{G(w')}(x, y)] \leq (c_2 \log n) d_{G(w)}(x, y)$.

Proof. Note that the following properties of the invariant follow immediately:

2. Every edge within a part C'_i has weight 0 in w'.
3. Every edge between two parts C'_i, C'_j has weight at least $(\frac{c_1}{\beta}\log n)R$ in w'.
5. T' has a spanning tree within each part C'_i, and no edges in between parts.

To prove invariant (4), suppose that $x, y \in V$ are in the same C'_i. If they are also in the same C_i, then the property holds by the input guarantee. Otherwise, by Lemma 15, w.h.p. the parts containing x and y have distance $O(\frac{1}{\beta}\log n)$ in the BFS tree on $H(\frac{1}{R}w_H)$, which means that there is a path in the BFS tree that travels through $O(\frac{1}{\beta}\log n)$ vertices in $H(\frac{1}{R}w_H)$. We consider the distance through edges in $H(\frac{1}{R}w_H)$ and through vertices in $H(\frac{1}{R}w_H)$ (which are actually parts in G) separately. For the edges, the distance is at most $O(\frac{1}{\beta}\log n)R$ in H, and each of these edges has weight at least that in G, giving $O(\frac{1}{\beta}\log n)R$ total distance. For the vertices, traversing through T inside the $O(\frac{1}{\beta}\log n)$ parts takes $O(R)$ distance each, by the input guarantee, and $O(\frac{1}{\beta}\log n)R$ distance overall. Combining the two arguments proves (4) $d_{T'}(x,y) \le (\frac{c_1}{\beta}\log n)R$. Note that (4) immediately implies that (1) the weighted diameter of each part C'_i using edge weights in G is at most $(\frac{c_1}{\beta}\log n)R$.

Finally, we prove that $\mathbb{E}[d_{G(w')}(x,y)] \le (c_2\log n)d_{G(w)}(x,y)$. If $x, y \in V$ are in the same C'_i, then their distance in $G(w')$ is zero and the claim follows. Otherwise, consider the shortest path in H, which is also the shortest path in $H(\frac{1}{R}w_H)$. By Lemma 16, every edge e on this path has probability at most $1 - e^{-O(w_e\beta)} = O(w(e)\beta)$ of being cut between two components, so the expected length is at most $O(w(e)\beta)\cdot\frac{c_1}{\beta}\log n = O(w(e)\log n)$. By linearity of expectation, the expected multiplicative increase of the path in $H(\frac{1}{R}w_H)$, and also in $G(w')$, is $O(\log n)$. ◀

2.2 Algorithm Main Loop

In this section, we apply `LDDSubroutine` recursively with geometrically increasing values of R. We show that the resulting forest approximates distances in expectation.

Algorithm T = `ExpectedSPForest`(G, β, R_0)

Input:

- $G = (V, E)$, the network graph with edge weights in $[1, \text{poly}(n)]$.
- $\beta = (\log n)^{-\Omega(1)}$, freely chosen.
- $R_0 \in [\frac{c_2\beta}{c_1}, 1]$

Algorithm:

```
1. Initially, set $R^{(0)} := R_0$, $T^{(0)} := \emptyset$, and $w^{(0)}$ to have the same edge weights as $G$.
2. For $t = 1, 2, \ldots$, while $R < n^c$ for large enough $c$:
   a. $(w^{(t)}, T^{(t)}) :=$ LDDSubroutine$(w^{(t-1)}, T^{(t-1)}, \beta, R^{(t-1)})$.
   b. Set $R^{(t)} := (\frac{c_1}{\beta}\log n)R^{(t-1)}$.
3. Output the forest obtained on the last iteration.
```

Note that T is not guaranteed to be a tree at the end of the algorithm, so distances within T can be infinite. However, a simple induction with linearity of expectation shows that the expected increase in length behaves in a controlled way:

▶ **Lemma 20.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, and let $\beta := (\log n)^{-\Omega(1)}$. On the* t*th* *iteration of `ExpectedSPForest`, for any two vertices $x, y \in V$, $\mathbb{E}[d_{G(w^{(t)})}(x,y)] \le (c_2\log n)^t d_G(x,y)$.*

We now show that we get approximate shortest paths with constant probability.

▶ **Lemma 21.** *Let G be a network graph with edge weights in $[1, \mathrm{poly}(n)]$, and let $\beta := (\log n)^{-\Omega(1)}$. The algorithm* **ExpectedSPForest** *runs in $\tilde{O}(\frac{1}{\beta}Q_G)$ rounds. Consider the output forest T, and fix any two vertices $x, y \in V$. Then, $d_T(x,y) \geq d_G(x,y)$ always*[6]*, and with constant probability, $d_T(x,y) \leq O(\frac{1}{\beta}\log n \cdot d_G(x,y)^{O(\log\log n)/\log(1/\beta)}) \cdot d_G(x,y)$.*

Proof. For the running time, there are $O(\frac{\log n}{\log(1/\beta)})$ iterations of the LDD, each of which takes $\tilde{O}(Q_G)$ time.

For simpler notation, define $L := d_G(x,y)$. Since every edge added to T has weight at least the weight of that same edge in G, we clearly have $d_T(x,y) \geq L$. We now prove the other bound on $d_T(x,y)$.

Consider any iteration t such that $R^{(t)} \geq 2(c_2\log n)^t L$. (We later argue that such an iteration t must exist.) By Lemma 20 and Markov's inequality, $d_{\tilde{G}^{(t)}}(x,y) < R^{(t)}$ with probability at least $\frac{1}{2}$. If this occurs, then x and y cannot belong to different parts at iteration t, since the distance between parts is at least $R^{(t)}$. By the subroutine guarantee, $d_{T^{(t)}}(x,y) = O(\frac{1}{\beta}\log n)R^{(t-1)} = O(R^{(t)})$, and since the edges of $T^{(t)}$ are preserved for the rest of the algorithm, $d_T(x,y) = O(R^{(t)})$ as well. Therefore, for this value of t, the approximation factor is $2(c_2\log n)^t$ with probability at least $\frac{1}{2}$.

It remains to find the smallest satisfying t. The condition on t is equivalent to $(\frac{c_1}{\beta}\log n)^t R_0 \geq 2(c_2\log n)^t L$, or $t \geq \lceil \frac{\log(2L)-\log(R_0)}{\log(c_1/(c_2\beta))}\rceil$. For t achieving equality, we get

$$R^{(t)} = O\left(\frac{c_1}{\beta}\log n\right)^t R_0 \leq \left(\frac{c_1}{\beta}\log n\right)^{\frac{\log(2L)-\log(R_0)}{\log(c_1/(c_2\beta))}+1} = \frac{c_1}{\beta}\log n \cdot \left(\frac{c_1}{\beta}\log n\right)^{\frac{\log(2L)-\log(R_0)}{\log(c_1/(c_2\beta))}}.$$

First, consider the case when $L \leq c_1/(c_2\beta)$. Since $R_0 \geq (c_2\beta)/c_1$, we have $\log(2L) - \log(R_0) \leq \log 2$, so

$$R^{(t)} \leq \frac{c_1}{\beta}\log n \cdot \left(\frac{c_1}{\beta}\log n\right)^{\frac{\log 2}{\log(c_1/(c_2\beta))}} \leq \frac{c_1}{\beta}\log n \cdot O(1) \leq O\left(\frac{c_1}{\beta}\log n\right) \cdot L,$$

where the second-to-last inequality uses that $\beta = (\log n)^{\Omega(1)}$.

Now consider the case when $L \geq c_1/(c_2\beta)$. Since $R_0 \geq (c_2\beta)/c_1$, we have $\log(2L) - \log(R_0) \leq \log(2L^2)$, so

$$R^{(t)} \leq \left(\frac{c_1}{\beta}\log n\right)^{\frac{\log(2L^2)}{\log(c_1/(c_2\beta))}+1} = \frac{c_1}{\beta}\log n \cdot \left(\frac{c_1}{\beta}\log n\right)^{\frac{\log(2L^2)}{\log(c_1/(c_2\beta))}}$$

$$= \frac{c_1}{\beta}\log n \cdot (2L^2)^{\frac{\log(c_1/\beta\cdot\log n)}{\log(c_1/(c_2\beta))}} = O\left(\frac{1}{\beta}\log n \cdot (2L^2)^{\frac{O(\log\log n)}{\log(1/\beta)}}\right),$$

as desired.

Lastly, we show that such an iteration t must exist. In particular, we show that the value of t chosen above satisfies $R^{(t)} \leq n^c$ for some large enough constant c in the algorithm. Since $L = \mathrm{poly}(n)$ and $R = 1/\mathrm{poly}(n)$, we have

$$t = \left\lceil \frac{\log(2L) - \log(R_0)}{\log(c_1/(c_2\beta))}\right\rceil = O\left(\frac{\log n}{\log(1/\beta)}\right).$$

[6] In particular, $d_T(x,y) = \infty$ if x and y are not in the same connected component in T

Therefore,

$$R^{(t)} = \left(\frac{c_1}{\beta}\log n\right)^t R_0 = \left(\frac{\log n}{\beta}\right)^{O\left(\frac{\log n}{\log(1/\beta)}\right)} = \left(\frac{1}{\beta}\right)^{O\left(\frac{\log n}{\log(1/\beta)}\right)} \cdot (\log n)^{O\left(\frac{\log n}{\log(1/\beta)}\right)}$$
$$= n^{O(1)} \cdot n^{O(1)},$$

where the last equality uses the fact that $\beta = (\log n)^{-\Omega(1)} \implies \log(1/\beta) = \Omega(\log\log n)$. Therefore, $R^{(t)} \le n^c$ for large enough c. ◀

From the shortest path forest, we can also derive the distances to each vertex v from a specified source s. Below is the algorithm, which runs in $\tilde{O}(\frac{1}{\beta}Q_G)$ rounds.

Algorithm `ExpectedSPDistance`(G, β, s)

```
1. Run ExpectedSPForest$(G, \beta)$ to obtain forest $T$. Set $\tilde{T}$ to be the connected component of $T$ that contains the source $s$.
2. For all vertices $v \notin \tilde{T}$, set $d(s, v) := \infty$.
3. Run AggregatePathToRoot (see Appendix B of the full version) with $x_v = 1$ for all $v \in \tilde{T}$ to determine the depth of each vertex in the tree $\tilde{T}$ rooted at $s$.
4. Every vertex $v \in \tilde{T}\backslash\{s\}$ computes its parent in the rooted tree, which it can determine by finding the one neighbor with smaller depth.
5. For each $v \in \tilde{T}\backslash\{s\}$, set $x_v$ to be the weight of the edge to its parent, and set $x_s := 0$. Run AggregatePathToRoot on these values to determine $d(s, v)$ for $v \in T$.
```

3 Solving SSSP and Related Problems

3.1 SSSP Trees

In this section, we describe an algorithm that outputs an approximate single source shortest path tree with source s. At a high level, to boost the probability that distances are well-approximated, we construct many randomized trees and take a collective "best" tree.

Algorithm `SSSPTree`(G, β, s)

```
1. Repeat ExpectedSPDistance$(G, \beta, s)$ $\Theta(\log n)$ times to obtain distances $d_{T_i}(v) := d_{T_i}(s, v)$.
2. For each vertex $v$, set $d_{\min}(v) := \min_i d_{T_i}(v)$.
3. For each vertex $v$ except the source, connect an edge to some neighbor $u$ that satisfies $d_{\min}(u) + w_{(u,v)} \le d_{\min}(v)$. Return the tree $T^*$ of all such edges.
```

▶ **Lemma 22.** *Let G be a network graph with edge weights in $[1, \text{poly}(n)]$, and let $\beta := (\log n)^{-\Omega(1)}$. W.h.p., `SSSPTree` runs for $\tilde{O}(\frac{1}{\beta}Q_G)$ rounds and outputs a shortest path tree that $O(\frac{1}{\beta}d_G(v))^{\frac{O(\log\log n)}{\log(1/\beta)}}\log n)$-approximates distances from the source to each v.*

Proof. Observe that in step 3 of `SSSPTree`, such a neighbor always exists, since in the tree T_i that achieves distance $d_{\min}(v)$ to v, the parent u of v in T_i satisfies $d_{\min}(u)+w_{(u,v)} = d_{\min}(v)$. To show that $d_{T^*}(v) \le d(v)$ for each v, consider the path $s = v_0, v_1, v_2, \ldots, v_\ell = v$ in T^*. We have $w(v_i, v_{i-1}) \le d_{\min}(v_i) - d_{\min}(v_{i-1})$ for each i, and summing up the inequalities gives the result.

From Lemma 21, each vertex v achieves the desired approximation with constant probability. By taking the minimum $d_{T_i}(v)$ over $\Theta(\log n)$ trees, this approximation is satisfied w.h.p. for every v, giving $d_{T^*}(v) \le d_{\min}(v) = O(\frac{1}{\beta} d_G(v)^{1+\frac{O(\log\log n)}{\log(1/\beta)}} \log n) \cdot d_G(v)$. ◀

By repeating `SSSPTree` multiple times with differing R_0, we can shave off the $\frac{1}{\beta}\log n$ in the approximation as follows, giving our main result for SSSP.

▶ **Theorem 7.** *Let G be a network graph with edge weights in $[1, \mathrm{poly}(n)]$, with a specified source vertex, and let $\beta := (\log n)^{-\Omega(1)}$. There is a distributed algorithm that, w.h.p., runs for $\tilde{O}(\frac{1}{\beta} Q_G)$ rounds and outputs a spanning tree that approximates distances to the source to factor $O(L^{O(\log\log n)/\log(1/\beta)})$.*[7] *By output, we mean that at the end of the algorithm, every vertex knows its set of incident edges in the spanning tree.*

Proof. We first handle the pairs $u, v \in V$ with $d(u,v) \ge c_1/(c_2\beta)$.

Run `SSSPTree` $\lceil \log(c_1/(c_2\beta)) \rceil = O(\log n)$ many times, setting $R_0 := 2^{-t}$ on the t^{th} iteration. The total number of rounds is $\tilde{O}(\frac{1}{\beta} Q_G)$. Consider the case $L \ge c_1/(c_2\beta)$ in the proof of 21. Observe that the factor $\frac{1}{\beta}$ comes from the $+1$ in the ceiling computation in the expression $\lceil \frac{\log(2L) - \log(R_0)}{\log(c_1/(c_2\beta))} \rceil$. However, with the differing values of R_0, there exists one such R_0 such that taking the ceiling increases the value by at most $\frac{1}{\log(c_1/(c_2\beta))}$. This factor gets absorbed in the exponent $\frac{O(\log\log n)}{\log(1/\beta)}$. Therefore, for each v, there exists a tree with this approximation factor, and running steps 2 and 3 from `SSSPTree` on these trees gives the result.

Now we handle the pairs $u, v \in V$ with $d(u,v) \le c_1/(c_2\beta)$. Intuitively, this should not be a problem: if we run an LDD with $\beta \in [1/L, 2/L]$, then by Lemma 16, with constant probability, u and v are in a common component of diameter $O(L \log n)$, stretching the distance by a factor $O(\log n)$.

Let us define w_G to be the weights of the input graph G. Then the algorithm runs `LDDSubroutine`$(w_G, \emptyset, \beta', 1)$ times for each $\beta' \in [1, c_1/(c_2\beta)]$ satisfying $\beta' = 2^{-i}$ for some positive integer i, and repeats this loop $O(\log n)$ times. In total, this takes $\tilde{O}(\frac{1}{\beta} Q_G)$. W.h.p., for each pair $u, v \in V$ with $d(u,v) \le c_1/(c_2\beta)$, there is a forest T returned by one of the `LDDSubroutine`s for which $d_T(u,v) \le O(\log n) d_G(u,v)$. Finally, running steps 2 and 3 from `SSSPTree` on these forests, along with the tree obtained from the case $L \ge c_1/(c_2\beta)$, gives the desired SSSP tree. ◀

3.2 Distance Labeling Schemes

We restate our main result on approximate distance labeling schemes.

▶ **Theorem 11.** *Let G be a network graph with edge weights in $[1, \mathrm{poly}(n)]$. There exists a $(\mathrm{polylog}(n), n^{O(\log\log n)/\log(1/\beta)})$ approximate distance labeling scheme that runs in $\tilde{O}(\frac{1}{\beta} Q_G)$ rounds.*

Proof. For each t from 1 to $\lceil \log(c_1/(c_2\beta)) \rceil$, run `ExpectedSPForest` $\Theta(\log n)$ times with $R_0 := 2^{-t}$. By analysis from Lemma 21 and Theorem 7, w.h.p., for every $x, y \in V$, there is an iteration of `ExpectedSPForest` with $R = O(d_G(v)^{1+\frac{O(\log\log n)}{\log(1/\beta)}})$ that outputs a cluster containing both x and y. The total number of rounds is $\tilde{O}(\frac{1}{\beta} Q_G)$.

[7] Recall that $L = \max_{u,v} d_G(u,v)$.

In each of the $O(\log^2 n)$ iterations of `ExpectedSPForest`, consider all of the clusters formed throughout the algorithm, and give each one a unique ID. For every iteration with parameter R and a cluster formed in that iteration, assign to every vertex within the cluster the label (ID, R). Each vertex is assigned to $O(\frac{\log n}{\log(1/\beta)})$ clusters per `ExpectedSPForest`, so the label size is $\mathrm{polylog}(n)$.

To compute distances given two vertices $x, y \in V$, simply output the minimum possible R over all clusters that contain both x and y, which is easily computed with the labels of x and y. By the analysis above, the minimum possible R gives the desired approximation factor $O(d_G(v)^{O(\log\log n)/\log(1/\beta)}) = O(n^{O(\log\log n)/\log(1/\beta)})$. ◀

3.3 Transshipment Problem

Let G be a transshipment network with demand d_v at each node v. The following algorithm computes an approximate transshipment flow in expectation.

```
Algorithm ExpectedTS
1. Run ExpectedSPForest and root the tree $T$ arbitrarily.
2. Using AggregateSubtree (see Appendix B) compute $F(v) := \sum_{u \in S_v} d_v$ for all $v$, where
   $S_v$ is the subtree rooted at $v$.
3. For each edge $(v, p) \in T$ with $p$ the parent of $v$ in the rooted tree, direct $F(v)$ flow
   from $v$ to $p$. (If $F(v)$ is negative, then direct the flow the other way.)
```

▶ **Lemma 23.** *Let G be a network graph with edge weights in $[1, \mathrm{poly}(n)]$ and demands that sum to zero, and let $\beta := (\log n)^{-\Omega(1)}$. The expected total cost of `ExpectedTS` is within $\tilde{O}(\frac{1}{\beta} n^{O(\log\log n)/\log(1/\beta)})$ of optimum.*

Proof. Decompose the optimal solution into a set of (shortest) paths. For a path from s to t, we have $\mathbb{E}[d_T(s,t)] = \tilde{O}(\frac{1}{\beta} n^{O(\log\log n)/\log(1/\beta)}) \cdot d_G(s,t)$ by Lemma 21, and by linearity of expectation, the cost C of routing each of these paths through T gives an expected $\tilde{O}(\frac{1}{\beta} n^{O(\log\log n)/\log(1/\beta)})$ approximation. It remains to show that the total cost of `ExpectedTS` is at most C. If `ExpectedTS` places $F(e)$ flow along an edge e, then the total demand difference between the two halves of the tree split at e is $|2F|$. Therefore, any sequence of paths along T that satisfies all demands must route at least $|F|$ flow along edge e. It follows that C must be at least the cost of `ExpectedTS`. ◀

By running `ExpectedTS` repeatedly and taking the overall best flow, we obtain our main result for transshipment.

▶ **Theorem 13.** *Let G be a network graph with edge weights in $[1, \mathrm{poly}(n)]$ and demands that sum to zero, and let $\beta := (\log n)^{-\Omega(1)}$. There is an algorithm that, w.h.p., runs for $\tilde{O}(\frac{1}{\beta} Q_G)$ rounds and computes a $\tilde{O}(\frac{1}{\beta} n^{O(\log\log n)/\log(1/\beta)})$-approximate flow.*

Proof. Run `ExpectedTS` $\Theta(\log n)$ many times and output the minimum total cost. By Markov's inequality and Lemma 23, w.h.p., some iteration achieves within twice the expected approximation of $\tilde{O}(\frac{1}{\beta} n^{O(\log\log n)/\log(1/\beta)})$. ◀

4 Conclusion and Future Work

Using the shortcuts framework from [8, 9], we give the first nontrivial approximation algorithms for shortest path problems which run in $o(\sqrt{n} + D)$ time on non-pathological network topologies. Our algorithms feature a tuneable parameter β that represents the balance between approximation ratio and running time. For certain values of β, we obtain polylogarithmic-approximate solutions in $\tilde{O}(n^{\epsilon} \cdot Q_G)$ rounds for the shortest path and distance labeling problems. While sublogarithmic approximation ratios are known to be impossible (even existentially) for labeling schemes with polylogarithmic labels we believe that our approximation guarantees can likely be improved for nice family of graphs, and, in the case of the SSSP-tree and transshipment problems, even generally.

In particular, for the quite general set of minor closed families of graphs one might be able to use more sophisticated low-diameter decompositions, such as [1], which would directly lead to $O(1)$-approximation guarantees for such networks in our framework. However, [1] is written for the sequential setting and making the algorithms in [1] distributed and compatible with the shortcut framework is a nontrivial extension, which we plan to explore for the journal version of this work.

More importantly, it seems possible that our non-trivial approximation ratios for the SSSP-tree and transshipment problem can be improved all the way to $(1+\epsilon)$-approximations using tools from continuous optimization, such as, gradient descent or the multiplicative weights method. As one example, the recent and brilliant work of Becker et al. [5] shows how to obtain a $(1+\epsilon)$-approximation for the SSSP-tree problem and the transshipment problem by computing $\tilde{O}(\alpha^2)$ many α-approximations to the transshipment problem. This work also demonstrates that the required updates to weight and demand vectors can be performed in various non-centralized models, including CONGEST. If this method could be applied to our transshipment algorithm, we could choose $\beta = 2^{-O(\sqrt{\log n \log\log n})}$ to get a $2^{O(\sqrt{\log n \log\log n})}$-approximate solution to the transshipment problem in $Q_G \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds, which could then be transformed into a $(1+\epsilon)$ approximation with the exact same running time (up to the constant hidden by the O-notation). This extension is highly nontrivial as well and left for future work.

References

1 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 79–88. ACM, 2014.

2 Noga Alon, Richard M Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

3 Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Low-diameter graph decomposition is in nc. In *Scandinavian Workshop on Algorithm Theory*, pages 83–93. Springer, 1992.

4 Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 184–193. IEEE, 1996.

5 Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *International Symposium on Distributed Computing*, 2017.

6 Guy E Blelloch, Anupam Gupta, Ioannis Koutis, Gary L Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory of Computing Systems*, 55(3):521–554, 2014.

7 Michael Elkin and Ofer Neiman. Distributed strong diameter network decomposition. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 211–216. ACM, 2016.

8 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 202–219. Society for Industrial and Applied Mathematics, 2016.

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

10 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In *International Symposium on Distributed Computing*, pages 158–172. Springer, 2016.

11 Bernhard Haeupler, Goran Zuzic, and Jason Li. Low-congestion shortcuts for any minor closed family. In *personal communications*, 2017.

12 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. An almost-tight distributed algorithm for computing single-source shortest paths. In *Proceedings of the ACM Symposium on Theory of Computing*, 2016.

13 Shay Kutten and David Peleg. Fast distributed construction of k-dominating sets and applications. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 238–251. ACM, 1995.

14 Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 153–162. ACM, 2015.

15 Nathan Linial and Michael E Saks. Decomposing graphs into regions of small diameter. In *SODA*, volume 91, pages 320–330, 1991.

16 Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 196–203. ACM, 2013.

17 Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 565–573, 2014.

18 Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *International Symposium on Distributed Computing*, pages 439–453. Springer, 2014.

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