Distributed Exact Weighted All-Pairs Shortest Paths in Õ(n 5/4) Rounds

Chien-Chung Huang, Danupon Nanongkai, Thatchaphol Saranurak

Distributed Exact Weighted All-Pairs Shortest Paths in $\tilde{O}(n^{5/4})$ Rounds

Chien-Chung Huang[1], Danupon Nanongkai[2], and Thatchaphol Saranurak[2]

[1]CNRS, École Normale Supérieure, France
[2]KTH Royal Institute of Technology, Sweden

Abstract

We study computing *all-pairs shortest paths* (APSP) on distributed networks (the CONGEST model). The goal is for every node in the (weighted) network to know the distance from every other node using communication. The problem admits $(1+o(1))$-approximation $\tilde{O}(n)$-time algorithms [LP15, Nan14], which are matched with $\tilde{\Omega}(n)$-time lower bounds [Nan14, LPS13, FHW12][1]. No $\omega(n)$ lower bound or $o(m)$ upper bound were known for exact computation.

In this paper, we present an $\tilde{O}(n^{5/4})$-time randomized (Las Vegas) algorithm for exact weighted APSP; this provides the first improvement over the naive $O(m)$-time algorithm when the network is not so sparse. Our result also holds for the case where edge weights are *asymmetric* (a.k.a. the directed case where communication is bidirectional). Our techniques also yield an $\tilde{O}(n^{3/4}k^{1/2}+n)$-time algorithm for the *k-source shortest paths* problem where we want every node to know distances from k sources; this improves Elkin's recent bound [Elk17b] when $k=\tilde{\omega}(n^{1/4})$.

We achieve the above results by developing distributed algorithms on top of the classic *scaling technique*, which we believe is used for the first time for distributed shortest paths computation. One new algorithm which might be of an independent interest is for the *reversed r-sink shortest paths* problem, where we want every of r sinks to know its distances from all other nodes, given that every node already knows its distance to every sink. We show an $\tilde{O}(n\sqrt{r})$-time algorithm for this problem. Another new algorithm is called *short range extension*, where we show that in $\tilde{O}(n\sqrt{h})$ time the knowledge about distances can be "extended" for additional h hops. For this, we use weight rounding to introduce small *additive* errors which can be later fixed.

Remark: Independently from our result, Elkin recently observed in [Elk17b] that the same techniques from an earlier version of the same paper (`https://arxiv.org/abs/1703.01939v1`) led to an $O(n^{5/3}\log^{2/3} n)$-time algorithm.

[1]$\tilde{\Theta}$, $\tilde{O}$ and $\tilde{\Omega}$ hide polylogarithmic factors. Note that the lower bounds also hold even in the unweighted case and in the weighted case with polynomial approximation ratios.

Contents

1 Introduction

Distributed Graph Algorithms. Among fundamental questions in distributed computing is how fast a network can compute its own topological properties, such as minimum spanning tree, shortest paths, minimum cut and maximum flow. This question has been extensively studied in the so-called *CONGEST model* [Pel00] (e.g. [Elk17b, PRS17, HKN16, Nan14, LPS13, DHK+12, Elk06, PR00, GKP98]). In this model (see Section 2 for details), a network is modeled by a weighted n-node m-edge graph G. Each node represents a processor with unique ID and infinite computational power that initially only knows its adjacent edges and their weights. Nodes must communicate with each other in *rounds* to discover network properties, where in each round each node can send a message of size $O(\log n)$ to each neighbor. The *time complexity* is measured as the number of rounds needed to finish the task. It is usually expressed in terms of n, m, and D, where D is the diameter of the network when edge weights are omitted. Throughout we use $\tilde{\Theta}$, $\tilde{O}$ and $\tilde{\Omega}$ to hide polylogarithmic factors in n.

Note that the whole network can be aggregated to a single node in $O(m)$ time. Thus any graph problem can be trivially solved within $O(m)$ time. A fundamental question is whether this bound can be beaten, and if so, what is the best possible time complexity for solving a particular graph problem. This question has been studied for several decades, marked by a celebrated $O(n \log n)$-time algorithm for the minimum spanning tree (MST) problem by Gallager et al. [GHS83]. This result was gradually improved and settled with $\tilde{\Theta}(\sqrt{n} + D)$ upper and lower bounds [PR00, GKP98, KP98, Awe87, CT85, Gaf85].[2]

Approximation vs. Exact Algorithms. Besides MST, almost no other problems were known to admit an $o(m)$-time distributed algorithm when we require the solution to be *exact*. More than a decade ago, a lot of attention has turned to *distributed approximation*, where we allow algorithms to return approximate solutions (e.g. [Elk04]). This relaxation has led to a rapid progress in recent years. For example, SSSP, minimum cut, and maximum flow can be $(1 + o(1))$-approximated in $\tilde{O}(\sqrt{n} + D)$ time [HKN16, BKK+16, Nan14, NS14, GK13, GKK+15][3], and all-pairs shortest paths can be $(1 + o(1))$-approximated in $\tilde{O}(n)$ time [LP15, Nan14]; moreover, these bounds are essentially tight up to polylogarithmic factors [DHK+12, Elk06, PR00, KKP13]. Given that approximating many graph properties are essentially solved, it is natural to turn back to exact algorithms. A fundamental question is:

Are approximation distributed algorithms more powerful than the exact ones?

So far, we only have an answer to the MST problem: due to the lower bound of Das Sarma et al. [DHK+12] (building on [Elk06, PR00, KKP13]), any poly(n)-approximation algorithm requires $\tilde{\Omega}(\sqrt{n} + D)$ rounds; thus, approximation does *not* help. For most other problems, however, answering the above question still seems to be beyond current techniques: On the one hand, we are not aware of any lower bound technique that can distinguish distributed $(1 + o(1))$-approximation from exact algorithms for the above problems. On the other hand, for most of these problems we do not even know any non-trivial (e.g. $o(m)$-time) exact algorithm. (One exception that we are aware of is SSSP where the classic Bellman-Ford

[2] See also [PRS17, Elk17a] for recent results.

[3] For the maximum flow algorithm, there is an extra $n^{o(1)}$ term in the time complexity.

algorithm [Bel58, For56] takes $O(n)$ time. This bound was recently (in STOC'17) improved by Elkin [Elk17b].)

All-Pairs Shortest Paths (APSP). Motivated by the above question, in this paper we attempt to reduce the gap between the upper and lower bounds for solving APSP exactly. The goal of the APSP problem is for every node to know the distances from every other node.[4] Besides being a fundamental problem on its own, this problem is a key component in, e.g., routing tables constructions [LPS13, LP15].

Nanongkai [Nan14] and Lenzen and Patt-Shamir [LPS13, LP15] presented $(1 + o(1))$-approximation $\tilde{O}(n)$-time algorithms, as well as an $\tilde{\Omega}(n)$ lower bound which holds even when $D = O(1)$, when the network is unweighted, and against randomized algorithms.[5] Very recently, Censor-Hillel et al. [CKP17] improved the lower bound to $\Omega(n)$. The same lower bound obviously holds for the exact case. Neither an $\omega(n)$ lower bound nor an $o(m)$-time algorithm was known, except for some special cases; a notable one is the unweighted case where there are $O(n)$-time algorithms [LP13, HW12, PRT12].

Results. Our main result is an $\tilde{O}(n^{5/4})$-time exact APSP algorithm. Our algorithm is randomized Las Vegas: the output is always correct, and the time guarantee holds both in expectation and with high probability[6]. This result provides the first improvement over the naive $O(m)$-time algorithm when the network is not so sparse, and significantly reducing the gap between upper and lower bounds.

Our algorithm also works with the same guarantees when edge weights are *asymmetric*, a.k.a. the *directed case*. In this case, an edge between nodes u and v can be viewed as two *directed* edges, one from u to v and another from v to u. These two edges might have different weights, and the weight can be set to infinity. (Note, however, that the infinite weight is not really necessary as it can be replaced by a large $\text{poly}(n)$ weight.) We emphasize that the underlying network is undirected so neither edge direction or weight affect the communication. While the previous $(1+o(1))$-approximation $\tilde{O}(n)$-time algorithms for APSP also work for this case [Nan14, LP15], in general it is less understood than the undirected case. For example, while there are tight $(1+o(1))$-approximation $\tilde{O}(\sqrt{n}+D)$-time algorithms for SSSP [HKN16, BKK+16] in the undirected case, the best known algorithms for the directed case are the $(1 + o(1))$-approximation $\tilde{O}(\sqrt{nD} + D)$-time one [Nan14] and the $\tilde{O}(\sqrt{n}D^{1/4} + D)$-time algorithm for the special case called single-source reachability [GU15].

Our techniques also yield an improved algorithm for the *k-source shortest paths* (k-SSP) problem. In this problem, some nodes are marked as source nodes initially (each node knows whether it is marked or not). The goal is for every node to know its distance from every source node. We let k denote the number of source nodes. We show a randomized Las-Vegas $\tilde{O}(n^{3/4}k^{1/2} + n)$-time algorithm. (Observe that our APSP algorithm is simply a

[4] This problem is sometimes referred to as *name-independent routing schemes*. See, e.g. [LPS13, LP15] for discussions and results on another variant called *name-dependent routing schemes* which is not considered in this paper.

[5] In fact, the lower bound holds even for $\text{poly}(n)$-approximation algorithms when the network is weighted and for $(\text{polylog}(n))$-approximation algorithms when the network is unweighted. The same lower bound also holds even for the easier problem of approximating the network diameter [FHW12]. In particular, for the weighted case, the lower bound holds even for $\text{poly}(n)$-approximation algorithms. For the unweighted case, the lower bound holds even for $(3/2 - \epsilon)$-approximating the diameter, and even for sparse networks [ACK16].

[6] We say that an event holds with high probability (w.h.p.) if it holds with probability at least $1 - 1/n^c$, where c is an arbitrarily large constant.

special case when $k = n$.) Prior to our work, approximation algorithms and the unweighted case were often considered for this problem (e.g. [EN16, HW12, KKM+12, Elk05]). The only non-trivial exact algorithm known earlier was the algorithm of Elkin [Elk17b]. The performance of such algorithm compared to ours is as follows. (We ignore the polylogarithmic terms below for simplicity. For a more precise time guarantee, see [Elk17b].)

1. When $D = O(k\sqrt{n})$ and $k = O(\sqrt{n})$, Elkin's algorithm takes $\tilde{O}(n^{5/6}k^{2/3})$ time. In this case our algorithm is faster when $k = \tilde{\omega}(n^{1/4})$.
2. When $k = \Omega(\sqrt{n})$, Elkin's algorithm takes $\tilde{O}(n^{2/3}k)$ time. In this case our algorithm is faster when $k = \tilde{\omega}(n^{1/3})$.

To conclude, our k-SSP algorithm is faster whenever $k = \tilde{\omega}(n^{1/4})$.

Remarks.

1. The time guarantees of our algorithms depend on the number of bits needed to represent edge weights. This is typically polylog(n) since edge weights are usually assumed to be positive integers bounded from above by poly(n); see Section 2.1. This is the main drawback of the scaling technique that our algorithm heavily relies on (see below). The guarantees of other distributed algorithms we discussed, including Elkin's algorithm [Elk17b], do not have this dependency.

2. Throughout the paper we only show that the output is correct with high probability, but in $O(n)$ time we can check the correctness as follows. First, every node lets its neighbors know about its distances from other nodes (this takes $O(n)$ time). Then, every node checks if it can improve its distance from any node using the distance knowledge from neighbors. If the answer is "no" for every node, then the computed distance is correct. If some node answers "yes", it can broadcast its answer to all other nodes in $O(n)$ time.

3. Independently from our result, Elkin recently observed in [Elk17b] that the same techniques from an earlier version of the same paper (`https://arxiv.org/abs/1703.01939v1`) led to an $O(n^{5/3}\log^{2/3} n)$-time algorithm for APSP on undirected networks. Also very recently, Censor-Hillel et al. [CKP17] showed that the standard Alice-Bob framework is incapable of providing a super-linear lower bound for exact weighted APSP, and raised the complexity of APSP as "an intriguing open question". They also showed an $\tilde{\Omega}(n^2)$ lower bound for exactly solving some NP-hard problems, such as minimum vertex cover, maximum independent set and graph coloring. This implies a huge separation between approximation and exact algorithms, since some of these problems can be solved approximately in $O(\text{polylog}(n))$ time.

1.1 Overview of Algorithms and Techniques.

Our algorithms are built on the *scaling technique*. This is a classic technique heavily studied in the sequential setting (e.g. [Gol95, GT89, Gab85]). As far as we know this is the first time it is used for shortest paths computation in the distributed setting. This technique (see Section 3 for details) allows us to assume that the distance between any two nodes is $O(n)$ (i.e., the so-called *weighted diameter* is $O(n)$). The main challenge here is that edge weight can be *zero* (but cannot be negative); without the zero weight, there are already many $\tilde{O}(n)$-time exact algorithms available (e.g. [LP13, Nan14]). Our algorithms consist of two main subroutines developed for this case, which might be of independent interest. We discuss these subroutines below. To avoid the discussion being too complicated, readers may

assume throughout the discussion that the input network has symmetric edge weight. Note however that in reality we have to deal with the asymmetric weights even if the original weight is symmetric. Additionally, we assume for simplicity that every pair of nodes has a unique shortest path.

1. Short-range extension. The first subroutine is called *short-range-extension*. For simplicity, let us first consider a special case called *short-range* problem. In this problem we are given a parameter h. The goal is for every node v to know the distance from every node u such that the shortest uv-path has at most h edges. Previously, this task can be achieved in $\tilde{O}(nh)$ time by running the Bellman-Ford algorithm for h rounds from every node. By exploiting special properties obtained from the scaling technique, we develop an $\tilde{O}(n\sqrt{h})$-time algorithm for this problem. The main idea is as follows. First we increase the zero weight to a small positive weight $\Delta = 1/\sqrt{h}$. By a breadth-first-search (BFS) algorithm, we can solve APSP in the new network (with positive weights) in $\tilde{O}(n/\Delta)$ time. This solution gives an *upper bound* to the APSP problem on the original network (with zero weights). Since we are interested in only shortest paths with at most h edges, it can be argued that the upper bound obtained has an *additive error* of $h\Delta$; i.e. it is only $h\Delta$ higher than the actual distance. We fix this additive error by running the Bellman-Ford algorithm for $h\Delta$ rounds from every node.

The short-range algorithm above can be generalized to the following *short-range-extension* problem. We are given an integer h, and initially some nodes in the network already know distances to some other nodes. For any nodes u and v, let $(u = x_0, x_1, x_2, \ldots, x_k = v)$ be the shortest uv-path. We say that (u, v) is *h-nearly realized* if at least one node among $x_k, x_{k-1}, \ldots, x_{k-h}$ knows its distance from u. (Note that the fact that (u, v) is h-nearly realized does not necessarily imply that (v, u) is also h-nearly realized.) At the end of our algorithm we want to make node v know the distance from u, for every nodes u and v such that (u, v) is h-nearly realized initially. Observe that the short-range problem is the special case where initially node u knows distances from no other nodes. By modifying the short-range algorithm, we can show that this problem can be solved in $\tilde{O}(n\sqrt{h})$ time as well.

2. Reversed r-sink shortest paths. The second subroutine is called *reversed r-sink shortest paths*. Initially, we assume that every node v knows the distance from v to r sink nodes. The goal is for every sink to know its distance from every node. A naive solution is for every node v to broadcast to the whole network the distance from v to every sink. This takes $O(nr)$ time since there are $O(nr)$ distance information to broadcast. In this paper, we develop an $\tilde{O}(n\sqrt{r})$-time algorithm for this task.

The main idea is for every node v to route the distance from v to every sink t through the shortest vt-path. If there is a node x that is contained in more than $n\sqrt{r}$ shortest paths (thus there will be too much information going through x), we will call x a *bottleneck node*. We can bound the number of bottleneck nodes to $O(\sqrt{r})$ by a standard argument – we charge each bottleneck node to $n\sqrt{r}$ distinct shortest paths among nr of them. Now, for every shortest vt-path that does not contain a bottleneck node, we route the distance from node v and sink t as originally planned. This takes $\tilde{O}(n\sqrt{r})$ time since there is $\tilde{O}(n\sqrt{r})$ bits of information going through each node. For shortest vt-paths that contain bottleneck nodes, we do the following. For every bottleneck node c, we make every node know their distances from and to c by running the Bellman-Ford algorithm starting at c. Then every node broadcasts to the whole network its distance to and from every bottleneck node. Since there are $\sqrt{r}$ bottleneck nodes, this takes $O(n\sqrt{r})$ time in total. It is not hard to show that

every sink t knows the distance from every node v after this step.

Putting things together. Finally, we sketch how all tools are put together. First we run the short-range algorithm with parameter $h = \sqrt{n}$. Then we sample $\tilde{O}(\sqrt{n})$ nodes uniformly at random called *centers* so that every h-hop path contains a center with high probability. Each center c broadcasts to the whole network its distances to some centers that it learns from the short-range algorithms. At this point, every node knows its distance to every center. We invoke the reversed r-sink shortest paths algorithm with centers as sink nodes (so $r = \tilde{O}(\sqrt{n})$), so that every center knows its distance from every node. At this point, it is not hard to prove that every pair of nodes is h-nearly realized. So, we finish by invoking the short-range-extension algorithm with parameter $h = \sqrt{n}$. The total time is $\tilde{O}(n\sqrt{r} + n\sqrt{h}) = \tilde{O}(n^{5/4})$.

To extend the above idea to the k-source shortest paths problem, we need slight modifications here and there; in particular, (i) we modify the short-range extension and reversed r-sink shortest paths algorithms to deal with k source nodes, and (ii) we treat the sampled centers as source nodes since we need to know the distances from and to them.

2 Preliminaries

2.1 The Model

In a nutshell, we consider the standard CONGEST model, except that instead of an undirected graph the underlying graph is modeled by a *bidirected* graph, i.e. a directed graph in which the reverse of every edge is also an edge. This is because we have to deal with asymmetric edge weight (even when the initial network has symmetric weights). Additionally, for simplicity we assume that nodes IDs are in the range of $\{0, 1, \ldots, n-1\}$. (This assumption can be achieved in $O(n)$ time.)

More precisely, we model a network by a *bidirected* unweighted n-node m-edge graph G, where nodes model the processors and edges model the *bounded-bandwidth* links between the processors. Let $V(G)$ and $E(G)$ denote the set of nodes and (directed) edges of G, respectively. The processors (henceforth, nodes) are assumed to have unique IDs in the range of $\{0, 1, \ldots, n-1\}$ and infinite computational power. (Note again that typically nodes' IDs are assumed to be in the range of $\{1, \ldots, \text{poly}(n)\}$. But in $O(n)$ time the range can be reduced to $\{0, 1, \ldots, n-1\}$.) Each node has limited topological knowledge; in particular, it only knows the IDs of its neighbors and knows *no* other topological information (e.g., whether its neighbors are linked by an edge or not). Nodes may also accept some additional inputs as specified by the problem at hand.

For the case of graph problems, the additional input is *edge weights*. Let $w : E(G) \to \{1, 2, \ldots, \text{poly}(n)\}$ be the edge weight assignment.[7] We refer to network G with weight assignment w as the *weighted network*, denoted by $G(w)$. The weight $w(u, v)$ of each edge (u, v) is known only to u and v. As commonly done in the literature, we will assume that the maximum weight is $\text{poly}(n)$; so, each edge weight can be sent through an edge (link) in one round. We refer to the weight function as *symmetric*, or sometimes *undirected*, if for every (directed) edge (u, v), $w(u, v) = w(v, u)$. Otherwise, it is called *asymmetric*, or

[7] Note that it might be natural to include ∞ as a possible edge weight. But this is not necessary since it can be replaced by a large weight of value $\text{poly}(n)$.

sometimes *directed*. We note again that the symmetric case is the typical case considered in the literature, but we have to deal with the asymmetric case in our algorithm.

We measure the performance of algorithms by its running time, defined as the worst-case number of *rounds* of distributed communication. At the beginning of each round, all nodes wake up simultaneously. Each node u then sends an arbitrary message of $O(\log n)$ bits through each edge (u, v), and the message will arrive at node v at the end of the round. We assume that nodes always know the number of the current round. In this paper, the running time is analyzed in terms of the number of nodes (n). Since n can be computed in $O(D)$ time, where D is the diameter of G, we will assume that every node knows n.

2.2 Problems and Notations

For every nodes s and t in a weighted network $G(w)$, let $\mathsf{dist}_w(s, t)$ be the distance from s to t in $G(w)$. Note that if w is asymmetric then it might be the case that $\mathsf{dist}_w(s, t) \neq \mathsf{dist}_w(t, s)$. Let $P^*_w(s, t)$ be the shortest path from s to t in $G(w)$; if there are more than one such path, we let $P^*_w(s, t)$ be the one with the least number of edges (if there are still more than one, break tie arbitrarily). We refer to $P^*_w(s, t)$ as *the* shortest st-path.

The goal of the *all-pairs shortest paths* (APSP) problem is for every node t to know $\mathsf{dist}_w(s, t)$ for every node s. In the case of *k-source shortest paths* (k-SSP) problem, there is a set S of k *source nodes* (every node knows whether it is in S or not). The goal is for every node t to know $\mathsf{dist}_w(s, t)$ for every source $s \in S$. When $k = 1$, the problem is called *single-source shortest paths* (SSSP).

We say that an event holds *with high probability* (w.h.p.) if it holds with probability at least $1 - 1/n^c$, where c is an arbitrarily large constant.

2.3 Basic Distributed Algorithms

The Bellman-Ford Algorithm. We note the following algorithm for SSSP on network $G(w)$, known as *Bellman-Ford* [Bel58, For56]. Let s be the source node. For any node t, let $d^t_w(s, t)$ denote the knowledge of t about $\mathsf{dist}_w(s, t)$. Initially, $d^t_w(s, t) = \infty$ for every node t, except that $d^s_w(s, s) = 0$. The algorithm proceeds as follows.

(i) In round 0, every node t sends $d^t_w(s, t)$ to all its neighbors.
(ii) When a node t receives the message about $d^x_w(s, x)$ from its neighbors x, it uses the new information to decrease the value of $d^t_w(s, t)$.
(iii) If $d^t_w(s, t)$ decreases, then node t sends the new value of $d^t_w(s, t)$ to all its neighbors.
(iv) Repeat (ii) and (iii) for n rounds.

Clearly, the above algorithm takes $O(n)$ rounds. Moreover, it can be proved that when the algorithm terminates $d^t_w(s, t) = \mathsf{dist}_w(s, t)$; i.e. t knows $\mathsf{dist}_w(s, t)$.

Scheduling of Distributed Algorithms. Consider k distributed algorithms $A_1, A_2 \ldots, A_k$. Let $\mathsf{dilation}$ be such that each algorithm A_i finishes in $\mathsf{dilation}$ rounds if it runs individually. Let $\mathsf{congestion}$ be such that there are at most $\mathsf{congestion}$ messages, each of size $O(\log n)$, sent through each edge (counted over all rounds), when we run all algorithms together. We note the following result of Ghaffari [Gha15]:

Theorem 2.1 ([Gha15]). *There is a distributed algorithm that can execute $A_1, A_2 \ldots, A_k$ altogether in $O(\mathsf{dilation} + \mathsf{congestion} \cdot \log n)$ time.*

Broadcasting. We need to follow fact following from basic upcasting and downcasting techniques [Pel00]. (The statement is from [LP13].)

Lemma 2.2. *Suppose each $v \in V$ holds $k_v \geq 0$ messages of $O(\log n)$ bits each, for a total of $K = \sum_{v \in V} k_v$ messages. Then all nodes in the network can receive these K messages within $O(K + D)$ rounds.*

2.4 Sampling the Centers

In the beginning of each iteration, a special node (with ID 0) chooses a subset of *centers* uniformly random and broadcasts this information (their IDs) to all other nodes. Here we use a lemma of Ullman and Yannakakis [UY91, Lemma 2.2].

Lemma 2.3 ([UY91]). *If we choose z distinct nodes uniformly at random from an n-node graph, then the probability that a given (acyclic) path has a sequence of more than $(cn \log n)/z$ nodes, none of which is distinguished, is, for sufficiently large n, bounded above by $2^{-\alpha c}$ for some positive α.*

The special node chooses $\sqrt{n}\,\mathrm{polylog}(n)$ centers at random and broadcasts this information (the broadcasting can be done in $O(\sqrt{n}\,\mathrm{polylog}(n) + D) = O(n)$ rounds). Then the following lemma is a direct consequence of the previous one.

Lemma 2.4. *Let w be any non-negative weight function. For any nodes s and t, let $P^*_w(s,t)$ be the shortest st-path in $G(w)$ as defined in Section 2.2. Then, with high probability, every $P^*_w(s,t)$ can be decomposed into a set of subpaths $P_0 = (s = u_0, \ldots, u_1)$, $P_1 = (u_1, \ldots, u_2)$, $\ldots$, $P_{k-1} = (u_{k-1}, \ldots, u_k = t)$, where*

- *the u_i are centers for $1 \leq i \leq k-1$.*
- *each subpath has at most $\sqrt{n} - 1$ edges.*

3 The Scaling Framework

Let $\bar{w}$ denote the given (possibly asymmetric) weight function of the input graph G. We want every node t to know the distances from other nodes s to itself with respect to $\bar{w}$. We emphasize that every edge (u, v) is directed, i.e., (u, v) is an ordered pair. We need the following definitions:

Definition 3.1. *Let β be the integer such that $2^{\beta-1} \leq \max_{(u,v)\in E(G)} \bar{w}(u,v) < 2^\beta$. For any $0 \leq i \leq \beta$ and edge (u,v), let $w_i(u,v) = \left\lfloor \bar{w}(u,v)/2^{\beta-i} \right\rfloor$. That is, $w_i(u,v)$ is the number represented by the first i most significant bits of $\bar{w}(u,v)$ (when we treat the β-th bit as the most significant one). Let $b_i(u,v) \in \{0,1\}$ be the i-th bit in the binary representation of $\bar{w}(u,v)$, i.e., $\bar{w}(u,v) = \sum_{i=0}^{\beta-1} b_i(u,v) 2^i$.*

Note that $\beta = O(\log n)$ because the weights of edges in G are polynomial. For any edge (u,v), $w_0(u,v) = 0$, $w_\beta(u,v) = \bar{w}(u,v)$, and $w_{i+1}(u,v) = 2w_i(u,v) + b_{\beta-i}(u,v)$ for $0 < i < \beta$. For each i, we can treat w_i and b_i as a weight function.

Definition 3.2. *For any (asymmetric) weight function $\hat{w}$, we denote by $d^u_{\hat{w}}(s,t)$ the knowledge of the node u about $\mathsf{dist}_{\hat{w}}(s,t)$, i.e., the distance from s to t with respective the weight $\hat{w}$.*

The algorithm will runs in β iterations. At the i-th iteration, we assume that for every node t knows the distances from all other nodes s to itself with respect to the weight w_{i-1}, i.e. $d^t_{w_{i-1}}(s,t) = \mathsf{dist}_{w_{i-1}}(s,t)$ for all s and t. The goal is to use this information to so that at the end of the iteration the knowledge of the distances with respect to w_i, i.e. we have $d^t_{w_i}(s,t) = \mathsf{dist}_{w_i}(s,t)$ for all s and t. Note that the assumption about the knowledge holds in the very beginning when $i = 1$, because $d^t_{w_0}(s,t) = \mathsf{dist}_{w_0}(s,t) = 0$ for all s and t by Definition 3.1.

For convenience, throughout the paper, we fix the iteration i. We denote the weight functions $w := w_i$, $w' := w_{i+1}$ and $b := b_{\beta-i}$. That is, we have $w'(u,v) = 2w(u,v) + b(u,v)$ for every edge (u,v). In the beginning, we have $d^t_w(s,t) = \mathsf{dist}_w(s,t)$ and we want to have $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$ at the end.

3.1 Upper Bounding the Distances

As $\mathsf{dist}_{w'}(s,t)$ can be a large polynomial for some s,t, we can avoid this by working with a *set* of reduced weights r_s defined as follows.

Definition 3.3. *For any node s and edge $e = (u,v)$, let*

$$r_s(u,v) = 2\mathsf{dist}_w(s,u) + w'(u,v) - 2\mathsf{dist}_w(s,v). \tag{1}$$

We note that r_w is an asymmetric weight function even if w and w' are symmetric. The next lemma states some useful properties of r_s:

Lemma 3.4. *Let r_s be defined as in Definition 3.3. Then the following holds.*

(i) For any edge $e = (u,v)$, $r_s(u,v) \geq 0$.

(ii) For any nodes s and t, $\mathsf{dist}_{r_s}(s,t) \leq n-1$.

(iii) For any nodes s and t, $\mathsf{dist}_{w'}(s,t) = 2\mathsf{dist}_w(s,t) + \mathsf{dist}_{r_s}(s,t)$. In fact, any path is a shortest st-path in $G(w')$ if and only if it is a shortest st-path in $G(r_s)$.

Proof. For (i), observe that $r_s(u,v) = 2\mathsf{dist}_w(s,u) + w'(u,v) - 2\mathsf{dist}_w(s,v) \geq 2\mathsf{dist}_w(s,u) + 2w(u,v) - 2\mathsf{dist}_w(s,v) \geq 0$, where the last inequality follows from the triangle inequality.

For (ii), first notice that

$$r_s(P) = w'(P) - 2\mathsf{dist}_w(s,t), \text{for any } st\text{-path } P. \tag{2}$$

The above inequality follows easily from definition. Let $P = (s = v_0, v_1, \ldots, t = v_k)$, for

some $k \leq n-1$. Then,

$$\begin{aligned} r_s(P) &= \sum_{j=0}^{k-1} r_s(v_j, v_{j+1}) \\ &= \sum_{j=0}^{k-1} 2\mathsf{dist}_w(s, v_j) + w'(v_j, v_{j+1}) - 2\mathsf{dist}_w(s, v_{j+1}) \\ &= (\sum_{j=0}^{k-1} w'(v_j, v_{j+1})) - 2\mathsf{dist}_w(s, v_k) \\ &= w'(P) - 2\mathsf{dist}_w(s,t)\,. \end{aligned}$$

Now assume that P is a shortest st-path in $G(w)$. Then

$$\begin{aligned} \mathsf{dist}_{r_s}(s,t) \leq r_s(P) \leq w'(P) - 2\mathsf{dist}_w(s,t) \\ &= (\sum_{j=0}^{k-1} 2w(v_j, v_{j+1}) + b(v_j, v_{j+1})) - 2\mathsf{dist}_w(s,t) \\ &= (\sum_{j=0}^{k-1} b(v_j, v_{j+1})) + 2w(P) - 2\mathsf{dist}_w(s,t) \\ &= \sum_{j=0}^{k-1} b(v_j, v_{j+1}) \leq n-1. \end{aligned}$$

Here the second inequality follows from (2), the fifth equality from the assumption that P is a shortest path in $G(w)$ and the last inequality from the fact that $k \leq n-1$ and $b(v_j, v_{j+1}) \in \{0,1\}$. This proves (ii).

Finally for (iii), let $P = (s = v_0, v_1, \ldots, t = v_k)$ be a shortest st-path in $G(w')$. Then

$$\mathsf{dist}_{r_s}(s,t) \leq r_s(P) = w'(P) - 2\mathsf{dist}_w(s,t) = \mathsf{dist}_{w'}(s,t) - 2\mathsf{dist}_w(s,t),$$

where the last equality holds as P is a shortest st-path in $G(w')$. On the other hand, let $P' = (s = v_0, v_1, \ldots, t = v_{k'})$ be a shortest path in $G(r_s)$. Then

$$\mathsf{dist}_{w'}(s,t) \leq w'(P') = r_s(P') + 2\mathsf{dist}_w(s,t) = dist_{r_s}(s,t) + 2\mathsf{dist}_w(s,t),$$

where the last equality holds because P' is a shortest path in $G(r_s)$. The above two inequalities establish the first part of (iii), while the second part follows from the first part. □

Lemma 3.4(ii) implies that shortest path tree with a source s, based on r_s, has depth at most n. However, we *cannot* construct such a tree using the standard BFS starting from s in just $O(n)$ rounds, the difficulty being that it can happen that $r_s(u,v) = 0$ for some edge (u,v). We also note that Lemma 3.4(ii) does *not* imply that every edge in the shortest paths has 0/1-weight.

4 Main Algorithm

In this section, we show the main algorithm described in Algorithm 1 which is the algorithm for one iteration in the scaling framework from Section 3. The setting is that there are three weight functions w, w' and b such that, for every edge (u, v) of the input graph G, $b(u, v) \in \{0, 1\}$ and

$$w'(u, v) = 2w(u, v) + b(u, v). \tag{3}$$

In the beginning, we have $d^t_w(s, t) = \mathsf{dist}_w(s, t)$ and we want that every node t knows $\mathsf{dist}_{w'}(s, t)$ for every node s, i.e., $d^t_{w'}(s, t) = \mathsf{dist}_{w'}(s, t)$ at the end of the algorithm.

For every pair of nodes s and t, recall that $P^*_{w'}(s, t)$ is the shortest st-path in $G(w')$; if there are more than one shortest st-paths in $G(w')$, pick the one with the least number of edges (if there are still more than one, break tie arbitrarily). Let C be the set of centers decided in Step 1 of Algorithm 1. Next, we define an important definition for our algorithm. Let $P^*_{w'}(s, t)|C$ denote the subpath of $P^*_{w'}(s, t)$ from the last center in $C \cap P$ to t. If there is no center in P, let $P^*_{w'}(s, t)|C = P^*_{w'}(s, t)$. Let $|P^*_{w'}(s, t)|$ be the number of edges in $P^*_{w'}(s, t)$; similarly, $|P^*_{w'}(s, t)|C|$ is the number of edges in $P^*_{w'}(s, t)|C$.

Recall that by Lemma 3.4(iii), $\mathsf{dist}_{w'}(s, t)$ differs from $\mathsf{dist}_{r_s}(s, t)$ by $2\mathsf{dist}_w(s, t)$, which is known to t. So if every node t knows that the distances w.r.t. r_s from each node s, i.e., $d^t_{r_s}(s, t) = \mathsf{dist}_{r_s}(s, t)$, then each node t can deduce the the distances w.r.t. to w' as well, i.e., $d^t_{w'}(s, t) = \mathsf{dist}_{w'}(s, t)$ for all s.

We first explain the high-level ideas behind our algorithm. In Algorithm 1, Step 1 is for sampling the centers. Step 2 is needed for the execution of Steps 3 and 6. Note that the implementation details of Steps 3, 5 and 6 will be elaborated in the subsequent sections.

Correctness: Let $h = \sqrt{n}$. For any nodes s and t, we will argue that, after executing Steps 3 to 6, every node t knows the distance w.r.t. w' from s to t, i.e., $d^t_{w'}(s, t) = \mathsf{dist}_{w'}(s, t)$. Let c_s be the first node in the path $P^*_{w'}(s, t)|C$, i.e. $P^*_{w'}(c_s, t) = P^*_{w'}(s, t)|C$. From the definition, if there is no centers in $P^*_{w'}(s, t)$ then $c_s = s$ and otherwise c_s is the last center appeared in the path $P^*_{w'}(s, t)$ from s to t.

We claim that after Step 5, the node c_s will know the distance w.r.t. w' from s to c_s, i.e., $d^{c_s}_{w'}(s, c_s) = \mathsf{dist}_{w'}(s, c_s)$. If $c_s = s$, this is trivial. Suppose $c_s \neq s$. Consider the shortest path $P^*_{w'}(s, c_s)$ from s to c_s. By Lemma 2.4, we can partition $P^*_{w'}(s, c_s)$ into subpaths, say $P_0 = (u_0 := s, \ldots, u_1)$, $P_1 = (u_1, \ldots, u_2)$, $\ldots$, $P_{k-1} = (u_{k-1}, \ldots, u_k := c_s)$ so that each subpath P_j has at most $h - 1$ edges for $0 \leq j \leq k - 1$, and the u_j's are centers for $1 \leq j \leq k - 1$. As subpath P_j has at most $h - 1$ edges, the **short-range** algorithm guarantees in Lemma 4.5 that u_j knows $d^{u_j}_{w'}(u_j, u_{j+1}) = \mathsf{dist}_{w'}(u_j, u_{j+1})$ for $0 \leq j \leq k - 1$ after Step 3 in Algorithm 1. In Step 4, $d^{u_j}_{w'}(u_j, u_{j+1})$, for $1 \leq j \leq k - 1$, will broadcast and be known to s. Therefore, after Step 4, the node s would be able to calculate $\mathsf{dist}_{w'}(s, c_s)$ and so $d^s_{w'}(s, c_s) = \mathsf{dist}_{w'}(s, c_s)$. Then, by the guarantee from Lemma 4.12 of the **reversed r-sink shortest paths** algorithm in Step 5, the knowledge is "exchanged" and so c_s knows $\mathsf{dist}_{w'}(s, c_s)$, i.e. $d^{c_s}_{w'}(s, c_s) = \mathsf{dist}_{w'}(s, c_s)$.

By Lemma 2.4, we also have that $P^*_{w'}(c_s, t) = P^*_{w'}(s, t)|C$ has at most $h - 1$ edges. As $d^{c_s}_{w'}(s, c_s) = \mathsf{dist}_{w'}(s, c_s)$, by the guarantee of the **short-range-extension** algorithm by Lemma 4.8, we have after Step 6 the node t knows the distance $\mathsf{dist}_{w'}(s, t)$, i.e. $d^t_{w'}(s, t) = \mathsf{dist}_{w'}(s, t)$ and we are done.

Algorithm 1: Main APSP Algorithm (for one iteration in the scaling framework)

Input: A graph G and the weight functions w, w', and b satisfying Equation (3). Every node t knows $\mathsf{dist}_w(s,t)$ for every node s, i.e., $d^t_w(s,t) = \mathsf{dist}_w(s,t)$. Let $h = \sqrt{n}$.

Output: Every node t knows $\mathsf{dist}_{w'}(s,t)$ for every node s, i.e. $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$.

1 Node 0 randomly samples $\sqrt{n}\text{polylog}(n)$ centers (collectively denoted as C) and broadcast their IDs to all other nodes. // This steps takes $O(n)$ rounds.

2 Node t sends $\mathsf{dist}_w(s,t)$, for all nodes s, to its neighbors x in G. The neighbor x internally uses this knowledge to compute $r_s(x,t)$, for all nodes s, as defined in Definition 3.3. // This steps takes $O(n)$ rounds.

3 Apply the **short-range** algorithm (in Section 4.1) so that every node s knows $d^s_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes t, and if $|P^*_{w'}(s,t)| \leq h$, $d^s_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$. // This step takes $\tilde{O}(n^{1.25})$ rounds.

4 All centers $c \in C$ broadcast their knowledge of $d^c_{w'}(c,c')$, for all centers $c' \in C$, to all other nodes in the network. Every node s internally uses this knowledge to calculate $d^s_{w'}(s,c) = \mathsf{dist}_{w'}(s,c)$ for all centers $c \in C$. // This step takes $\tilde{O}(n)$ rounds

5 Apply the **reversed r-sink shortest paths** algorithm (in Section 4.3) with nodes in C as sinks so that every center $c \in C$ knows $d^c_{w'}(s,c) = \mathsf{dist}_{w'}(s,c)$ for all nodes s. // This step takes $\tilde{O}(n^{1.25})$ rounds.

6 Apply the **short-range-extension** algorithm (in Section 4.2) so that every node t knows $d^t_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes s, and if $|P^*_{w'}(s,t)|C| \leq h$, $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$. // This step takes $\tilde{O}(n^{1.25})$ rounds.

Running Time: There are $O(|C|)$ messages to be broadcasted in Step 1, and $O(|C|^2)$ messages in Step 4. By Lemma 2.2, this takes $O(|C|^2 + D) = \tilde{O}(n)$ in total. Step 2 easily takes $O(n)$ rounds (by Theorem 2.1 we have $\mathsf{congestion} = n$ and $\mathsf{dilation} = 1$). In the following three subsections, we will show that Steps 3, 5 and 6 take $\tilde{O}(n^{1.25})$ rounds each. In particular, Lemmas 4.5 and 4.8 state that the short-range algorithm in Step 3 and the short-range-extension algorithm in Step 6 both take $\tilde{O}(n\sqrt{h})$. Lemma 4.12 states that the reversed r-sink shortest paths algorithm in Step 5 takes $\tilde{O}(n\sqrt{|C|})$. In total, the running time in each iteration is $\tilde{O}(n^{1.25})$ rounds.

Theorem 4.1. *At the end of Algorithm 1, with high probability, for every node t, $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$ for all nodes s. Furthermore, the algorithm takes $\tilde{O}(n^{1.25})$ rounds.*

4.1 Short-Range Algorithm

In this section we show how to implement Step 3 of Algorithm 1 so that every node s knows $d^s_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes t, and if $|P^*_{w'}(s,t)| \leq h$, $d^s_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$.

The main algorithm in this section is precisely described in Algorithm 2. However, it yields a slightly different output: after finishing, every node t knows $d^t_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes s, and if $|P^*_{w'}(s,t)| \leq h$, $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$. As they are completely symmetric, we can use Algorithm 2 as an algorithm for Step 3 of Algorithm 1 just by switching the direction of every edge in the graph. The reason for presenting Algorithm 2 that does not give exactly what we want for Step 3 of Algorithm 1 is that, later in Section 4.2, we will extend Algorithm 2 and obtain the short-range-extension algorithm. This formulation of Algorithm 2 simplifies the modification a lot. From now on, we will call Algorithm 2 the short-range algorithm as well.

Recall that we mentioned earlier that some edges (u,v) may have $r_s(u,v) = 0$ and this poses difficulty. Our main idea is to deal with a strictly positive weight function r'_s, defined as r_s rounded up to the next multiple of $\Delta = \sqrt{1/h}$. More precisely,

Definition 4.2. *Let $\Delta = \sqrt{1/h}$. For every node s and every edge (u,v), let*

$$r'_s(u,v) = \begin{cases} \Delta & \text{if } r_s(u,v) = 0, \text{ and} \\ \Delta\lceil r_s(u,v)/\Delta \rceil & \text{otherwise.} \end{cases}$$

Running Time: In Algorithm 2, Steps 1, 2 and 5 takes no time. For a single source s, the BFS in Step 3 has $\mathsf{dilation} = O((n + h\Delta)/\Delta) = O(n/\Delta + h)$ rounds. As in BFS each node sends messages only once and we run the BFS in parallel from all nodes s, we have $\mathsf{congestion} = O(n)$. By Theorem 2.1, we have that Step 3 takes $\tilde{O}(\mathsf{dilation} + \mathsf{congestion}) = \tilde{O}(n/\Delta + h + n) = \tilde{O}(n/\Delta)$. Step 4 is essentially the Bellman-Ford algorithm except the following modifications:

1. we start with $d^t_{r_s}(s,t) = \lfloor d^t_{r'_s}(s,t) \rfloor$ instead of $d^t_{r_s}(s,t) = \infty$, and
2. a node t sends its updated value of $d^t_{r_s}(s,t)$ only when $d^t_{r_s}(s,t) \geq d^t_{r'_s}(s,t) - h\Delta$ (instead of sending it every time $d^t_{r_s}(s,t)$ is decreased); see Step 4.(iii).

We run the modified Bellman-Ford algorithm for every node s in parallel. This algorithm for a single source node s has $\mathsf{dilation} = O(h)$ and $\mathsf{congestion} = O(h\Delta) = O(\sqrt{h})$ since every node sends a message to its neighbors at most $O(h\Delta)$ times (due to the second modification).

Algorithm 2: Short-Range Algorithm

Input: Every node t knows $\mathsf{dist}_w(s,t)$ and $r_s(t,x)$ for all nodes s and all t's neighbors x in G.

Output: For every pair of nodes s and t, node t knows $d^t_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ and if $|P^*_{w'}(s,t)| \leq h$, $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$.

1 For every edge (u,v) and all nodes s, both u and v internally compute $r'_s(u,v)$ according to Definition 4.2.

2 For every node t, initially set $d^t_{r'_s}(s,t) = \infty$ for all nodes $s \neq t$ and $d^t_{r'_t}(t,t) = 0$.

3 For every node s, compute SSSP tree from s up to depth $n + h\Delta$ in terms of r'_s by implementing the following BFS: each node $t(\neq s)$ updates $d^t_{r'_s}(s,t)$ according to the message $d^x_{r'_s}(s,x)$ it receives from its neighbor x. If $d^t_{r'_s}(s,t) \leq n + h\Delta$, then in round $d^t_{r'_s}(s,t)/\Delta$, the node t sends $d^t_{r'_s}(s,t)$ to all its neighbors in G, if t did not send any message in this step yet. `// Note that we count the number of rounds from 0.`

4 Every node t sets $d^t_{r_s}(s,t) = \lfloor d^t_{r'_s}(s,t) \rfloor$ for all nodes s. (Note that $d^t_{r_t}(t,t) = 0$.) Run the following algorithm (which is a modification of the Bellman-Ford algorithm) for every node s, in parallel:

(i) In round 0, every node t sends $d^t_{r_s}(s,t)$ to all its neighbors.

(ii) When a node t receives the message about $d^x_{r_s}(s,x)$ from its neighbors x, it uses the new information to decrease the value of $d^t_{r_s}(s,t)$ (as an upper estimate of $\mathsf{dist}_{r_s}(s,t)$). Note that $d^t_{r_s}(s,t)$ is always an integer.

(iii) If $d^t_{r_s}(s,t)$ decreases and $d^t_{r_s}(s,t) \geq d^t_{r'_s}(s,t) - h\Delta$, then the node t sends the new value of $d^t_{r_s}(s,t)$ to all its neighbors.

(iv) Repeat (ii) and (iii) for h rounds.

5 Every node t calculates $d^t_{w'}(s,t) = 2\mathsf{dist}_w(s,t) + d^t_{r_s}(s,t)$ for all nodes s.

By Theorem 2.1, parallelizing n such algorithms takes $\tilde{O}(h + n \cdot h\Delta) = \tilde{O}(nh\Delta)$ rounds. Now it can be concluded that Algorithm 2 takes $\tilde{O}(n/\Delta + nh\Delta) = \tilde{O}(n\sqrt{h})$ rounds.

Correctness: Next, we show the correctness of Algorithm 2 using the following lemmas.

Lemma 4.3. *After Step 3 of Algorithm 2, every node t knows $d^t_{r'_s}(s,t) \geq \mathsf{dist}_{r'_s}(s,t)$ for all nodes s and in particular $d^t_{r'_s}(s,t) = \mathsf{dist}_{r'_s}(s,t)$ if $\mathsf{dist}_{r'_s}(s,t) \leq n + h\Delta$.*

Proof. The first part follows from the property of the BFS. For the second part, first notice that Δ divides $\mathsf{dist}_{r'_s}(s,t)$ for all nodes s and t. By a straightforward induction, it can be shown that by round $\frac{\mathsf{dist}_{r'_s}(s,t)}{\Delta}$, $d^t_{r'_s}(s,t) = \mathsf{dist}_{r'_s}(s,t)$, if $0 \leq \mathsf{dist}_{r'_s}(s,t) \leq n + h\Delta$. □

Lemma 4.4. *After Step 4 of Algorithm 2, every node t knows $d^t_{r_s}(s,t) \geq \mathsf{dist}_{r_s}(s,t)$ for all nodes s, furthermore, if $|P^*_{w'}(s,t)| \leq h$, then $d^t_{r_s}(s,t) = \mathsf{dist}_{r_s}(s,t)$; in particular, $d^t_{r_s}(s,t)$ is decreased to $\mathsf{dist}_{r_s}(s,t)$ in round $|P^*_{w'}(s,t)|$ or before.*

Observe that the correctness of output of the algorithm follows from this lemma, since in Step 5, every note t can correctly compute $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$ if $|P^*_{w'}(s,t)| \leq h$ and otherwise $d^t_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$.

The intuition behind the proof is to show that $\mathsf{dist}_{r'_s}(s,t)$ (stored as $d^t_{r'_s}(s,t)$) computed in Step 3 is not very far from $\mathsf{dist}_{r_s}(s,t)$; i.e $\mathsf{dist}_{r'_s}(s,t) - \mathsf{dist}_{r_s}(s,t) \leq h\Delta$. Intuitively, this is because $|P^*_{w'}(s,t)| \leq h$, and for each edge (u,v), $0 \leq r'_s(u,v) - r_s(u,v) \leq \Delta$. This allows us to modify the Bellman-Ford algorithm in Step 4 to allow a node to speak only when $d^t_{r'_s}(s) - d^t_{r_s}(s,t) \leq h\Delta$.

Proof of Lemma 4.4. The fact that after Step 4, $d^t_{r_s}(s,t) \geq \mathsf{dist}_{r_s}(s,t)$ follows easily from induction on the number of rounds. We prove the rest by induction on $|P^*_{w'}(s,t)|$. For the base case where $|P^*_{w'}(s,t)| = 0$, i.e. $s = t$, the claim trivially holds as we set $d^t_{r_t}(t,t) = 0$ in the beginning of Step 4. Now consider any pair of s and t, and assume that the lemma holds for any t' such that $|P^*_{w'}(s,t')| < |P^*_{w'}(s,t)| \leq h$. Let x be the neighbor of t in $P^*_{w'}(s,t)$, i.e. $\mathsf{dist}_{w'}(s,t) = \mathsf{dist}_{w'}(s,x) + w'(x,t)$. Note that

$$\begin{aligned} \mathsf{dist}_{r_s}(s,t) &= \mathsf{dist}_{r_s}(s,x) + r_s(x,t) \\ &= d^x_{r_s}(s,x) + r_s(x,t), \end{aligned} \tag{4}$$

where the first equality holds because $P^*_{w'}(s,t) = P^*_{r_s}(s,t)$ is a shortest path in $G(r_s)$ by Lemma 3.4(iii). The second inequality then holds by the induction hypothesis. We will be done if the following claim holds.

Claim: x sends the message "$d^x_{r_s}(s,x) = \mathsf{dist}_{r_s}(s,x)$" to t in round $|P^*_{w'}(s,x)| + 1$ or before that (equivalently, $d^x_{r_s}(s,x)$ is decreased to $\mathsf{dist}_{r_s}(s,x)$ by round $|P^*_{w'}(s,x)| \leq h - 1$ or before that).

To see why we will be done, observe that the claim implies that t can update $d^t_{r_s}(s,t)$ to $\mathsf{dist}_{r_s}(s,t)$ using Equation (4) in round $|P^*_{w'}(s,t)|$ or before that. Note that t knows $r_s(x,t)$ from the initial knowledge. To prove the claim, we just need to show that $d^x_{r'_s}(s,x) -$

$\mathsf{dist}_{r_s}(s,x) \le (h-1)\Delta$. We have

$$\begin{aligned}
\mathsf{dist}_{r'_s}(s,x) &\le \mathsf{dist}_{r_s}(s,x) + |P^*_{r_s}(s,x)|\Delta && \text{by the definition of } r'_s \\
&\le \mathsf{dist}_{r_s}(s,x) + |P^*_{w'}(s,x)|\Delta && \text{by Lemma 3.4(iii)} \\
&\le \mathsf{dist}_{r_s}(s,x) + (h-1)\Delta \\
&\le n + (h-1)\Delta && \text{by Lemma 3.4(ii)}
\end{aligned}$$

By Lemma 4.3, we have $d^x_{r'_s}(s,x) = \mathsf{dist}_{r'_s}(s,x)$. By the second last inequality, we conclude that $d^x_{r'_s}(s,x) - \mathsf{dist}_{r_s}(s,x) \le (h-1)\Delta$. This proves the claim and the entire lemma. □

By flipping the direction of edges in the graph, we can conclude the result that is used in the main algorithm:

Lemma 4.5. *After running Algorithm 2 on a graph where the direction of each edge is flipped, every node s knows $d^s_{r_s}(s,t) \ge \mathsf{dist}_{r_s}(s,t)$ for all nodes t, furthermore, if $|P^*_{w'}(s,t)| \le h$, then $d^s_{r_s}(s,t) = \mathsf{dist}_{r_s}(s,t)$. Moreover the algorithm takes $\tilde{O}(n\sqrt{h})$ rounds.*

4.2 Short-Range-Extension Algorithm

In this section we show how to implement Step 6 of Algorithm 1 with the algorithm called short-range-extension algorithm. We are in the setting such that in the beginning, every center c already knows $d^c_{w'}(s,c) = \mathsf{dist}_{w'}(s,c)$ for all nodes s. By Lemma 2.4, this implies with high probability that for every pair s and t, (s,t) is *h-nearly realized*. Indeed, let $P^*_{w'}(s,t) = (s = x_0, x_1, x_2, \ldots, x_k = t)$ be the shortest path from s to t with respect to w'. We have that there is a center $c_s \in \{x_k, x_{k-1}, \ldots, x_{k-h}\}$ who knows its distance from s to itself with high probability by Lemma 2.4. The goal is that, at the end, every node t knows the distance $\mathsf{dist}_{w'}(s,t)$ for all nodes s. Moreover, it suffices to show that, at the end, every node t knows $d^t_{w'}(s,t) \ge \mathsf{dist}_{w'}(s,t)$ for all nodes s, and if $|P^*_{w'}(s,t)|C| \le h$, $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$.

The short-range-extension algorithm is a minor modification of the short-range algorithm in Algorithm 2, with the same running time and almost identical implementation. But, in this setting, the centers have additional initial knowledge: every center t already knows $\mathsf{dist}_{w'}(s,t)$ and hence $\mathsf{dist}_{r_s}(s,t)$ for all nodes s, i.e., $d^t_{r_s}(s,t) = \mathsf{dist}_{r_s}(s,t)$. The following changes exploit this knowledge:

- For any node s, let G_s be the graph obtained from G by adding imaginary edges into G: for every center t, there is an additional edge (s,t) with weight $\mathsf{dist}_{r_s}(s,t)$. We call G_s the *s-augmented* graph. We define the weight function r''_s for G_s in the same way as how we define the weight function r'_s for G. That is, for each original edge (u,v) in G_s, we set $r''_s(u,v) = r'_s(u,v)$, and, for each imaginary edge (s,t) where t is a center, we set
$$r''_s(s,t) = \begin{cases} \Delta & \text{if } \mathsf{dist}_{r_s}(s,t) = 0, \text{ and} \\ \Delta\lceil \mathsf{dist}_{r_s}(s,t)/\Delta \rceil & \text{otherwise.} \end{cases}$$
Let $\mathsf{dist}_{r''_s}(u,v)$ denote the distance from u to v with respect to r''_s *in the s-augmented graph G_s*.

- In Step 2, every pair of nodes s and t, initially set $d^t_{r''_s}(s,t) = \infty$ and $d^t_{r''_s}(t,t) = 0$, unless t itself is a center. In this case, let

$$d^t_{r''_s}(s,t) = \begin{cases} \Delta & \text{if } \mathsf{dist}_{r_s}(s,t) = 0, \text{ and} \\ \Delta\lceil \mathsf{dist}_{r_s}(s,t)/\Delta \rceil & \text{otherwise.} \end{cases}$$

  This is possible because each center t already knows $\mathsf{dist}_{r_s}(s,t)$ for all nodes s.

- In Step 3, for every node s, we compute the same SSSP tree w.r.t. r''_s instead of r'_s. Observe that, for every node s, running the BFS with respect to r''_s is the same as simulating Step 3 of the original short-range algorithm in the s-augmented graph G_s.

- In the beginning of Step 4, every node t sets $\lfloor d^t_{r_s}(s,t) = d^t_{r''_s}(s,t) \rfloor$ for all nodes s, unless t itself is a center. In this case, $d^t_{r_s}(s,t) = \mathsf{dist}_{r_s}(s,t)$. Moreover, we run this step for $h+1$ rounds instead of h rounds.

The running time clearly does not asymptotically change, and so this algorithm takes $\tilde{O}(n\sqrt{h})$ rounds. The next two lemmas establish the correctness of the algorithm and they are close parallels of Lemmas 4.3 and 4.4.

Lemma 4.6. *After Step 3 of the modified Algorithm 2, every node t knows $d^t_{r''_s}(s,t) \geq \mathsf{dist}_{r''_s}(s,t)$ for all nodes s, and in particular, $d^t_{r''_s}(s,t) = \mathsf{dist}_{r''_s}(s,t)$ if $\mathsf{dist}_{r''_s}(s,t) \leq n + h\Delta$.*

Proof. The proof is identical to Lemma 4.3 except that r'_s is replaced by r''_s. □

Lemma 4.7. *After Step 4 of the modified Algorithm 2, every node t knows $d^t_{r_s}(s,t) \geq \mathsf{dist}_{r_s}(s,t)$ for all nodes s, furthermore, if $|P^*_{w'}(s,t)|C| \leq h$, then $d^t_{r_s}(s,t) = \mathsf{dist}_{r_s}(s,t)$; in particular $d^t_{r_s}(s,t)$ decreases to $\mathsf{dist}_{r_s}(s,t)$ in round $|P^*_{w'}(s,t)|C| + 1$.*

Proof. The proof is almost identical to the proof of Lemma 4.4, with the difference that we consider the case that $|P^*_{w'}(s,t)|C| \leq h$ and not $|P^*_{w'}(s,t)| \leq h$. Similarly, we prove by induction on the length of $|P^*_{w'}(s,t)|C|$. For the base case where $|P^*_{w'}(s,t)|C| = 0$, we have that t itself is a center. Hence, the node t already knows the distance $\mathsf{dist}_{r_s}(s,t)$, i.e., $d^t_{r_s}(s,t) = \mathsf{dist}_{r_s}(s,t)$. For the inductive step, we only need to show that x, who is the previous node of t in $P^*_{w'}(s,t)|C$, has decreased $d^x_{r_s}(s,x)$ down to $\mathsf{dist}_{r_s}(s,x)$ in round $|P^*_{w'}(s,x)|C| + 1 \leq h$ or before that. This follows if we can show $d^x_{r''_s}(s,x) - \mathsf{dist}_{r_s}(s,x) \leq h\Delta$. Suppose that c_s is the first node in $P^*_{w'}(s,t)|C$ which is the first node in $P^*_{w'}(s,x)|C$ as well. We have that

$$\begin{aligned}
\mathsf{dist}_{r''_s}(s,x) &\leq \mathsf{dist}_{r''_s}(s,c_s) + \mathsf{dist}_{r''_s}(c_s,x) \\
&\leq (\mathsf{dist}_{r_s}(s,c_s) + \Delta) + (\mathsf{dist}_{r_s}(c_s,x) + |P^*_{r_s}(c_s,x)|\Delta) && \text{by the definition of } r''_s \\
&= \mathsf{dist}_{r_s}(s,x) + (|P^*_{r_s}(c_s,x)| + 1)\Delta \\
&= \mathsf{dist}_{r_s}(s,x) + (|P^*_{w'}(c_s,x)| + 1)\Delta && \text{by Lemma 3.4(iii)} \\
&= \mathsf{dist}_{r_s}(s,x) + (|P^*_{w'}(s,x)|C| + 1)\Delta \\
&\leq \mathsf{dist}_{r_s}(s,x) + h\Delta \\
&\leq n + h\Delta && \text{by Lemma 3.4(ii)}
\end{aligned}$$

By Lemma 4.6, we have $d^x_{r''_s}(s,x) = \mathsf{dist}_{r''_s}(s,x)$. By the second last inequality, we conclude that $d^x_{r''_s}(s,x) - \mathsf{dist}_{r_s}(s,x) \leq h\Delta$. And this completes the induction step and the entire proof. □

Note that the knowledge about $\mathsf{dist}_{r_s}(s,t)$ implies the knowledge about $\mathsf{dist}_{w'}(s,t)$. So now we can conclude the lemma that is used in the main algorithm:

Lemma 4.8. *Suppose that every center c already knows $d^c_{w'}(s,c) = \mathsf{dist}_{w'}(s,c)$ for all nodes s. After running the modified Algorithm 2, every node t knows $d^t_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes s, furthermore, if $|P^*_{w'}(s,t)| \leq h$ or $|P^*_{w'}(s,t)|C| \leq h$, then $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$. Furthermore, the algorithm runs in $\tilde{O}(n\sqrt{h})$ rounds.*

4.3 Reversed r-Sink Shortest Paths Algorithm

In this section, we assume that r special sink nodes $\mathbf{v}_1, \ldots, \mathbf{v}_r$ are given and every node s knows $d^t_{w'}(s,\mathbf{v}_i) = \mathsf{dist}_{w'}(s,\mathbf{v}_i)$ for all sink nodes $\mathbf{v}_i$. (Note that these r special sinks correspond to the centers C in Algorithm 1.) We present an $\tilde{O}(n\sqrt{r})$-time algorithm so that each sink $\mathbf{v}_i$, $1 \leq i \leq r$, acquires the knowledge $d^{\mathbf{v}_i}_{w'}(s,\mathbf{v}_i) = \mathsf{dist}_{w'}(s,\mathbf{v}_i)$ for all nodes s in the end. The algorithm is described in Algorithm 3. Here, we write the t-sink shortest path tree to mean the shortest path tree (w.r.t. w') that has t as the sink.

Now, we explain the idea of Algorithm 3. By Steps 1 and 2, for every sink $\mathbf{v}_i$, each node s can decide which neighbor x^* is its parent in the $\mathbf{v}_i$-sink shortest path tree: if $\mathsf{dist}'_w(s,\mathbf{v}_i) = w'(s,x^*) + \mathsf{dist}'_w(x^*,\mathbf{v}_i)$, then x^* is the parent of s. Also, every node s knows which neighbors are its children because the children informed s in Step 2.

The basic idea is to propagate $\mathsf{dist}_{w'}(\mathbf{v}_i,t)$ for all node t upwards to $\mathbf{v}_i$ in the $\mathbf{v}_i$-sink shortest path tree (as done in Step 5) until $\mathbf{v}_i$ receives all the informations. However, a brute-force implementation of this idea leads to $O(nr)$ time complexity, since some nodes may need to send out $O(nr)$ messages.

We overcome this issue by creating a set B of *bottleneck nodes* (or just *bottlenecks* for short), which is empty initially. Intuitively, these nodes are the bottlenecks of the above propagation process. We will let them become a sort of "ad-hoc" sinks, namely, if $b \in B$, we will let all nodes s know $d^s_{w'}(s,b) = \mathsf{dist}_{w'}(s,b)$. Furthermore, for all $1 \leq i \leq r$, the $\mathbf{v}_i$-shortest path trees will be "pruned" from these bottlenecks downwards in the following sense. In Step 4, a node s, if not a bottleneck in B, aggregates the number of its descendants (including s itself) in the $\mathbf{v}_i$-sink shortest path tree, for each $1 \leq i \leq r$, and then informs its parent in the same tree. On the other hand, if t is a bottleneck, it informs its parent in the $\mathbf{v}_i$-sink shortest path trees, for all $1 \leq i \leq r$, that it has no descendants, i.e., it is a leaf. (this can be regarded as our pruning the $\mathbf{v}_i$-sink shortest path trees from the bottlenecks downwards).

In Step 5, if some nodes t, which are neither bottlenecks nor the original sinks, have more than $n\sqrt{r}$ descendants, it declares itself as a potential candidate to become a new bottleneck. The special node with ID 0 will then decide on a unique node b to be the new bottleneck (so $B = B \cup \{b\}$) and broadcasts this decision. Then we build the b-sink shortest path tree and b-source shortest path tree using the Bellman-Ford algorithm so that all nodes s knows $d^s_{w'}(s,b) = \mathsf{dist}_{w'}(s,b)$ and $d^s_{w'}(b,s) = \mathsf{dist}_{w'}(b,s)$. Then, all nodes s forward $d^s_{w'}(s,b)$ to the sinks (by broadcasting to the whole network) so that every sink $\mathbf{v}_i$ knows $d^{\mathbf{v}_i}_{w'}(s,b) = \mathsf{dist}_{w'}(s,b)$. This will be useful information for sinks. The same process (Steps 4 and 5) continues until no more bottleneck is created.

Lemma 4.9. *The number of bottlenecks is $|B| = O(\sqrt{r})$ and so Steps 4 and 5 repeat $O(\sqrt{r})$ times.*

Algorithm 3: Reversed r-Sink Shortest Paths Algorithm

Input: r sink nodes $\mathbf{v}_1, \cdots, \mathbf{v}_r$. Every node s knows $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$ for all $1 \le i \le r$, i.e., $d^s_{w'}(s, \mathbf{v}_i) = \mathsf{dist}_{w'}(s, \mathbf{v}_i)$

Output: Each sink node $\mathbf{v}_i$ knows $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$ for all nodes s, i.e., $d^{\mathbf{v}_i}_{w'}(s, \mathbf{v}_i) = \mathsf{dist}_{w'}(s, \mathbf{v}_i)$

1 Every node s sends $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$, for each $1 \le i \le r$, to all its neighbors.

2 For each $1 \le i \le r$ and every node s, s uses the information $\mathsf{dist}_{w'}(x, \mathbf{v}_i)$ from all its neighbors x to decide which neighbor x^* is its parent in the $\mathbf{v}_i$-sink shortest path tree. The node s then informs x^* that it is a child of x^* in the $\mathbf{v}_i$-sink shortest path tree.

3 Set $B = \emptyset$. // B is the set of bottleneck nodes.

4 For each $1 \le i \le r$ and every node s, s waits until it receives the message $\#(i, x_j)$ from all its children x_j in the $\mathbf{v}_i$-sink shortest path tree. If the node $s \notin B$, let $\#(i, s) = 1 + \sum_j \#(i, x_j)$; otherwise $\#(i, s) = 0$. The node s sends $\#(i, s)$ to its parent in the $\mathbf{v}_i$-sink shortest path tree.

5 If any node $s \notin B \cup \{\mathbf{v}_i\}_{i=1}^r$ has $\sum_{i=1}^r \#(i, s) > \sqrt{k}n$:

(i) s broadcasts its intent of becoming a new bottleneck.

(ii) Node 0 chooses one of the candidates (say the one with the smallest ID) as the new bottleneck b and broadcasts its ID to all nodes. Set $B = B \cup \{b\}$.

(iii) Apply the Bellman-Ford algorithm to build the b-sink shortest path tree and the b-source shortest path tree, so that every node s knows $d^s_{w'}(s, b) = \mathsf{dist}_{w'}(s, b)$ and $d^s_{w'}(b, s) = \mathsf{dist}_{w'}(b, s)$.

(iv) Every node s broadcasts $\mathsf{dist}_{w'}(s, b)$ to all nodes (in particular, to all sinks), so that every sink $\mathbf{v}_i$ knows $d^{\mathbf{v}_i}_{w'}(s, b) = \mathsf{dist}_{w'}(s, b)$ for all nodes s.

(v) Go back to Step 4.

6 For each $1 \le i \le r$ and each node s, $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$ is relayed to sink $\mathbf{v}_i$ through the path $P^*_{w'}(s, \mathbf{v}_i)$ in the $\mathbf{v}_i$-sink shortest path tree if $P^*_{w'}(s, \mathbf{v}_i) \cap B = \emptyset$. That is, every node $x \in V \setminus B$ sends $\mathsf{dist}_{w'}(x, \mathbf{v}_i)$ to its parent in the $\mathbf{v}_i$-sink shortest path tree. When a node $v \in V \setminus B$ receives a message $\mathsf{dist}_{w'}(x, \mathbf{v}_i)$, it sends such message to its parent in the $\mathbf{v}_i$-sink shortest path tree.

7 Each sink $\mathbf{v}_i$, for $1 \le i \le r$, computes $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$ for all nodes s.

Proof. Observe that originally the total number of nodes in all $\mathbf{v}_i$-sink shortest path trees, for $1 \leq i \leq r$, is nr. Each time a new node becomes a bottleneck, all its descendants (at least $\Omega(n\sqrt{r})$ of them) are pruned from these trees. Thus, we can create up to at most $O(\frac{nr}{n\sqrt{r}}) = O(\sqrt{r})$ bottlenecks and accordingly Steps 4 and 5 repeat the same number of times. □

When there is no more bottleneck to be created, Step 6 simply relays the information $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$ to sink $\mathbf{v}_i$ through the $\mathbf{v}_i$-sink shortest path tree, for each $1 \leq i \leq r$, as long as 1) $s \in B$ is a bottleneck, or 2) s is not a bottleneck and the path from s to $\mathbf{v}_i$ in the $\mathbf{v}_i$-sink shortest path tree does not contain a bottleneck, i.e. $P^*_{w'}(s, \mathbf{v}_i) \cap B = \emptyset$. The last step finishes the algorithm.

Lemma 4.10. *In Step 7, each sink $\boldsymbol{v}_i$, for $1 \leq i \leq r$, correctly computes $\mathsf{dist}_{w'}(s, \boldsymbol{v}_i)$ for all nodes s, i.e., $d^{\boldsymbol{v}_i}_{w'}(s, \boldsymbol{v}_i) = \mathsf{dist}_{w'}(s, \boldsymbol{v}_i)$.*

Proof. Consider the path $P^*_{w'}(s, \mathbf{v}_i)$ from s to $\mathbf{v}_i$ in the $\mathbf{v}_i$-sink shortest path tree. There are two cases. First, if $s \in B$ or $P^*_{w'}(s, \mathbf{v}_i) \cap B = \emptyset$, then, by Step 6, $\mathsf{dist}_{w'}(s, \mathbf{v}_i)$ is relayed to $\mathbf{v}_i$ and we are done. Second, if s is not a bottleneck and there is a bottleneck b in $P^*_{w'}(s, \mathbf{v}_i)$, then, by Step 5(iv), $\mathsf{dist}_{w'}(s, b)$ is known to $\mathbf{v}_i$; i.e. $d^{\mathbf{v}_i}_{w'}(s, b) = \mathsf{dist}_{w'}(s, b)$. Also, by Step 5(iii), $d^{\mathbf{v}_i}_{w'}(b, \mathbf{v}_i) = \mathsf{dist}_{w'}(b, \mathbf{v}_i)$ is known to $\mathbf{v}_i$. Therefore, $\mathbf{v}_i$ can use these pieces of information to correctly compute $\mathsf{dist}_{w'}(s, \mathbf{v}_i) = d^{\mathbf{v}_i}_{w'}(s, b) + d^{\mathbf{v}_i}_{w'}(b, \mathbf{v}_i)$. □

The lemma above concludes the correctness of Algorithm 3. Now we analyze the running time.

Lemma 4.11. *Algorithm 3 takes $\tilde{O}(n\sqrt{r})$ rounds.*

Proof. We will use extensively Theorem 2.1 by analyzing $\mathsf{dilation}$ and $\mathsf{congestion}$ in each step. In Steps 1 and 2, each node only sends r messages to its neighbors. So $\mathsf{dilation} = 1$ and $\mathsf{congestion} = r$, and so this takes $\tilde{O}(r)$ rounds. In Step 4, for every sink $\mathbf{v}_i$, every node s sends a message once along the $\mathbf{v}_i$-sink shortest path tree. As there can be a path of n hops in the tree, $\mathsf{dilation} = n$. Parallelizing the processes for all sinks $\mathbf{v}_i$ yields $\mathsf{congestion} = r$. So this step takes $\tilde{O}(n + r) = \tilde{O}(n)$.

Now, we analyze Step 5. In Step 5(i), at most n nodes need to broadcast one message. By Lemma 2.2, this takes $O(n + D) = O(n)$ rounds. In Step 5(ii), only one node broadcast a message and this takes $O(D)$ rounds. In Step 5(iii), running Bellman-Ford algorithm for finding the b-sink shortest path tree, for one node b, takes $O(n)$. In Step 5(iv), every node broadcasts one messages and this takes $O(n + D) = O(n)$ rounds by Lemma 2.2.

By Lemma 4.9, Steps 4 and 5 repeat $O(\sqrt{r})$ times. In total, this takes $\tilde{O}(n\sqrt{r})$ rounds. Next, in Step 6, the messages are relayed in the shortest path trees, and so $\mathsf{dilation} = n$. Moreover, $\mathsf{congestion} = O(n\sqrt{r})$ because all the nodes s which are descendants of bottlenecks in any tree do not send messages. So this step also takes $O(n\sqrt{r})$ rounds. Therefore, the total number of rounds of the algorithm is $O(n\sqrt{r})$. □

Finally, we conclude with the lemma that is used in the main algorithms:

Lemma 4.12. *Every node s knows $\mathsf{dist}_{w'}(s, \boldsymbol{v}_i)$ for all sinks $\boldsymbol{v}_i$ where $1 \leq i \leq r$, i.e., $d^s_{w'}(s, \boldsymbol{v}_i) = \mathsf{dist}_{w'}(s, \boldsymbol{v}_i)$. Then, running Algorithm 3, each sink node $\boldsymbol{v}_i$ knows $\mathsf{dist}_{w'}(s, \boldsymbol{v}_i)$ for all nodes s, i.e., $d^{\boldsymbol{v}_i}_{w'}(s, \boldsymbol{v}_i) = \mathsf{dist}_{w'}(s, \boldsymbol{v}_i)$. Furthermore, Algorithm 3 takes $\tilde{O}(n\sqrt{r})$ rounds.*

5 k-Source Shortest Paths

In this section, we show how to extend the algorithms presented in Section 4 to solve the k-source shortest paths (k-SSP) problem. Recall that in this problem we want every node v to know its distance from every of k sources. We let S be the set of sources. Initially, every node knows whether it is a source or not.

We modify the APSP algorithm as follows. First, we pick β sets of random centers[8], where each set has size

$$\zeta = \min(k, \sqrt{n}) \operatorname{polylog}(n).$$

Denoted these sets by $C_1, C_2, \ldots, C_\beta$. (Observe that this step can be done in $\tilde{O}(\sqrt{n} + D)$ time since there are only $\tilde{O}(\sqrt{n})$ centers in total.) Now we run each iteration of the scaling framework as in Section 4, except that in each iteration we only compute shortest paths from only some sources (instead of all nodes). In particular, the set of sources at iteration i is $S_i = C_{i+1} \cup C_{i+2} \cup \ldots \cup C_\beta \cup S$. Thus, we can assume that every node knows its distance from all nodes in $S_{i-1} = C_i \cup C_{i+1} \cup \ldots \cup C_\beta \cup S$. We will use C_i as a set of random centers in iteration i, in the same way we use C in Algorithm 1. Algorithm 4 describes the new algorithm in details. In this algorithm, we also need to modify the short-range, reversed r-sink shortest paths, and short-range-extension so that they can run faster when there are only q sources, where $q = |S_i|$. This is done as in Algorithm 4.

q-Source Short-Range(-Extension) Algorithms. We round up edge weights to multiples of Δ as done previously. However, we only run the BFS algorithm from q sources (the depth is still $n + h\Delta$). We also run the modification of the Bellman-Ford algorithm with q-sources. By the same analysis as in Sections 4.1 and 4.2, the running time of the q-source short-range and short-range-extension algorithms becomes $\tilde{O}(n/\Delta + h + qh\Delta)$ which is

$$\tilde{O}(\sqrt{nqh})$$

when we set $\Delta = \sqrt{n/qh}$.

Reversed q-Source r-Sink Shortest Paths. The algorithm proceeds as in Section 4.3 except that:

- Bottleneck nodes are defined to be those that have $g = \sqrt{nqr}$ messages sent through them.
- In Step 6 the messages are relayed in the shortest path trees only from each source node (and not each node).

Since there are qr source-sink pairs, there are $|B| \leq \lceil qr/g \rceil = O(1 + \sqrt{qr/n})$ bottleneck nodes. By following the proof of Lemma 4.11, the running time of this algorithm becomes

$$\tilde{O}(r + |B|n + \sqrt{nqr}) = \tilde{O}(n + \sqrt{nqr}).$$

Total Time of Algorithm 4. The q-source short-range and short-range-extension algorithms take $\tilde{O}(\sqrt{nqh}) = \tilde{O}(n \cdot \sqrt{(k+\zeta)/\zeta}) = \tilde{O}(n + n^{3/4}k^{1/2})$ rounds. The reversed q-source r-sink shortest paths algorithm takes $\tilde{O}(n + \sqrt{nqr}) = \tilde{O}(n + \sqrt{n(k+\zeta)\zeta}) = \tilde{O}(n + n^{3/4}k^{1/2})$ rounds. Other steps can be easily seen to take $\tilde{O}(n)$ rounds. Thus, Algorithm 4 takes $\tilde{O}(n + n^{3/4}k^{1/2})$ rounds in total.

[8] Recall the β is the number of bits needed to represent edge weight (see Section 3).

Algorithm 4: k-SSP Algorithm (for iteration i in the scaling framework)

Input: A graph G, weight functions w, w', and b, and set of k sources S. Every node knows whether it is a source or not. Every node t knows $\mathsf{dist}_w(s,t)$, i.e. $d^t_w(s,t) = \mathsf{dist}_w(s,t)$, for every node $s \in S_{i-1} = C_i \cup \ldots \cup C_\beta \cup S$. Let $h = n/\zeta = \max(n/k, \sqrt{n})$.

Output: Every node t knows $\mathsf{dist}_{w'}(s,t)$ for every node $s \in S_i = C_{i+1} \cup \ldots \cup C_\beta \cup S$, i.e. $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$.

1 Let $C = C_i$.

2 Node t sends $\mathsf{dist}_w(s,t)$, for all nodes $s \in S_{i-1}$, to its neighbors x in G. The neighbor x internally uses this knowledge to compute $r_s(x,t)$, for all nodes $s \in S_{i-1}$, as defined in Definition 3.3. // This steps takes $O(q)$ rounds.

3 Apply the ***q*-source short-range** algorithm with nodes in S_i as sources and h as above so that every node $s \in S_i$ knows $d^s_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes t, and if $|P^*_{w'}(s,t)| \leq h$, $d^s_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$. // This step takes $\tilde{O}(\sqrt{nqh})$ $= \tilde{O}(n \cdot \sqrt{(k+\zeta)/\zeta}) = \tilde{O}(n + n^{3/4}k^{1/2})$ rounds.

4 All centers $c \in C$ broadcast their knowledge of $d^c_{w'}(c,c')$, for all centers $c' \in C$, to all other nodes in the network. Every node $s \in S_i$ internally uses this knowledge to calculate $d^s_{w'}(s,c) = \mathsf{dist}_{w'}(s,c)$ for all centers $c \in C$. // This step takes $\tilde{O}(\zeta^2) = \tilde{O}(n)$ rounds

5 Apply the **reversed *q*-source *r*-sink shortest paths** algorithm with nodes in S_i as sources and nodes in C as sinks, so that every center $c \in C$ knows $d^c_{w'}(s,c) = \mathsf{dist}_{w'}(s,c)$ for all nodes $s \in S_i$. // This step takes $\tilde{O}(n + \sqrt{nqr})$ $= \tilde{O}(n + \sqrt{n(k+\zeta)\zeta}) = \tilde{O}(n + n^{3/4}k^{1/2})$ rounds.

6 Apply the ***q*-source short-range-extension** algorithm so that every node t knows $d^t_{w'}(s,t) \geq \mathsf{dist}_{w'}(s,t)$ for all nodes $s \in S_i$, and if $|P^*_{w'}(s,t)|C| \leq h$, $d^t_{w'}(s,t) = \mathsf{dist}_{w'}(s,t)$. // This step takes $\tilde{O}(\sqrt{nqh}) = \tilde{O}(n + n^{3/4}k^{1/2})$ rounds.

6 Open Problems

The main question is whether distributed APSP can be solved in $\tilde{O}(n)$ time. Both super-linear lower bound or near-linear upper bound will be a major result. Another related problem is SSSP, where there is still a gap between the lower bound of [DHK+12] and upper bound of [Elk17b]. In general, it is very interesting to close the gap between approximation and exact distributed algorithms. We found this question particular interesting for exact maximum matching and minimum cut; these problem admit an $\tilde{\Omega}(\sqrt{n})$ lower bound while no non-trivial upper bound is known (even an $O(n)$ one). Note that the existing $\tilde{\Omega}(\sqrt{n})$ lower bound for minimum cut does not hold for a natural special case of checking whether the network has small, e.g. $O(1)$, *edge connectivity*. Given that small edge connectivity may indicate the network's likeliness to fail, it is interesting to determine their time complexity exactly. Currently there is a big jump from $O(D)$ time for checking edge connectivity of at most two [Thu97, PT11] to $\tilde{O}(\sqrt{n})$ for higher values [NS14].

7 Acknowledgement

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. Nanongkai and Saranurak were also partially supported by the Swedish Research Council (Reg. No. 2015-04659.) Nanongkai and Saranurak would like to thank Rotem Oshman for comments on the preliminary version of the result.

References

[ACK16] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. "Near-Linear Lower Bounds for Distributed Distance Computations, Even in Sparse Networks". In: *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*. 2016, pp. 29–42 (cit. on p. 2).

[Awe87] Baruch Awerbuch. "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and Related Problems (Detailed Summary)". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. 1987, pp. 230–240 (cit. on p. 1).

[BKK+16] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. "Approximate Undirected Transshipment and Shortest Paths via Gradient Descent". In: *CoRR* abs/1607.05127 (2016) (cit. on pp. 1, 2).

[Bel58] Richard Bellman. "On a Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90 (cit. on pp. 2, 6).

[CKP17] Keren Censor-Hillel, Seri Khoury, and Ami Paz. "Quadratic and Near-Quadratic Lower Bounds for the CONGEST Model". In: *DISC*. 2017 (cit. on pp. 2, 3).

[CT85] Francis Y. L. Chin and H. F. Ting. "An Almost Linear Time and O(n log n + e) Messages Distributed Algorithm for Minimum-Weight Spanning Trees". In: *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*. 1985, pp. 257–266 (cit. on p. 1).

[DHK+12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. "Distributed Verification and Hardness of Distributed Approximation". In: *SIAM Journal on Computing* 41.5 (2012). Announced at STOC'11, pp. 1235–1265 (cit. on pp. 1, 22).

[EN16] Michael Elkin and Ofer Neiman. "Hopsets with Constant Hopbound, and Applications to Approximate Shortest Paths". In: *FOCS* (2016) (cit. on p. 3).

[Elk04] Michael Elkin. "Distributed approximation: a survey". In: *SIGACT News* 35.4 (2004), pp. 40–57 (cit. on p. 1).

[Elk05] Michael Elkin. "Computing almost shortest paths". In: *ACM Transactions on Algorithms* 1.2 (2005). Announced at PODC'01, pp. 283–323 (cit. on p. 3).

[Elk06] Michael Elkin. "An Unconditional Lower Bound on the Time-Approximation Trade-off for the Distributed Minimum Spanning Tree Problem". In: *SIAM Journal on Computing* 36.2 (2006). Announced at STOC'04, pp. 433–456 (cit. on p. 1).

[Elk17a] Michael Elkin. "A Simple Deterministic Distributed MST Algorithm, with Near-Optimal Time and Message Complexities". In: *CoRR* abs/1703.02411 (2017) (cit. on p. 1).

[Elk17b] Michael Elkin. "Distributed Exact Shortest Paths in Sublinear Time". In: *Symposium on Theory of Computing, STOC*. 2017 (cit. on pp. i, 1–3, 22).

[FHW12] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. "Networks cannot compute their diameter in sublinear time". In: *SODA*. 2012, pp. 1150–1162 (cit. on pp. i, 2).

[For56] Lester R. Ford. *Network Flow Theory*. Tech. rep. P-923. The Rand Corporation, 1956 (cit. on pp. 2, 6).

[GHS83] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. "A Distributed Algorithm for Minimum-Weight Spanning Trees". In: *ACM Trans. Program. Lang. Syst.* 5.1 (1983), pp. 66–77 (cit. on p. 1).

[GK13] Mohsen Ghaffari and Fabian Kuhn. "Distributed Minimum Cut Approximation". In: *Symposium on Distributed Computing (DISC)*. 2013, pp. 1–15 (cit. on p. 1).

[GKK+15] Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. "Near-Optimal Distributed Maximum Flow: Extended Abstract". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*. 2015, pp. 81–90 (cit. on p. 1).

[GKP98] Juan A. Garay, Shay Kutten, and David Peleg. "A Sublinear Time Distributed Algorithm for Minimum-Weight Spanning Trees". In: *SIAM Journal on Computing* 27.1 (1998). Announced at FOCS'93, pp. 302–316 (cit. on p. 1).

[GT89] Harold N. Gabow and Robert Endre Tarjan. "Faster Scaling Algorithms for Network Problems". In: *SIAM J. Comput.* 18.5 (1989), pp. 1013–1036 (cit. on p. 3).

[GU15] Mohsen Ghaffari and Rajan Udwani. "Brief Announcement: Distributed Single-Source Reachability". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*. 2015, pp. 163–165 (cit. on p. 2).

[Gab85] Harold N. Gabow. "Scaling Algorithms for Network Problems". In: *J. Comput. Syst. Sci.* 31.2 (1985). Announced at FOCS'83, pp. 148–168 (cit. on p. 3).

[Gaf85] Eli Gafni. "Improvements in the Time Complexity of Two Message-Optimal Election Algorithms". In: *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5-7, 1985*. 1985, pp. 175–185 (cit. on p. 1).

[Gha15] Mohsen Ghaffari. "Near-Optimal Scheduling of Distributed Algorithms". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*. 2015, pp. 3–12 (cit. on pp. 6, 7).

[Gol95] Andrew V. Goldberg. "Scaling Algorithms for the Shortest Paths Problem". In: *SIAM J. Comput.* 24.3 (1995). Announced at SODA'93, pp. 494–504 (cit. on p. 3).

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "A deterministic almost-tight distributed algorithm for approximating single-source shortest paths". In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. 2016, pp. 489–498 (cit. on pp. 1, 2).

[HW12] Stephan Holzer and Roger Wattenhofer. "Optimal Distributed All Pairs Shortest Paths and Applications". In: *Symposium on Principles of Distributed Computing (PODC)*. 2012, pp. 355–364 (cit. on pp. 2, 3).

[KKM+12] Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. "Efficient distributed approximation algorithms via probabilistic tree embeddings". In: *Distributed Computing* 25.3 (2012). Announced at PODC 2008, pp. 189–205 (cit. on p. 3).

[KKP13] Liah Kor, Amos Korman, and David Peleg. "Tight Bounds for Distributed Minimum-Weight Spanning Tree Verification". In: *Theory of Computing Systems* 53.2 (2013). Announced at STACS'11, pp. 318–340 (cit. on p. 1).

[KP98] Shay Kutten and David Peleg. "Fast Distributed Construction of Small k-Dominating Sets and Applications". In: *Journal of Algorithms* 28.1 (1998). Announced at PODC'95, pp. 40–66 (cit. on p. 1).

[LP13] Christoph Lenzen and David Peleg. "Efficient Distributed Source Detection with Limited Bandwidth". In: *Symposium on Principles of Distributed Computing (PODC)*. 2013, pp. 375–382 (cit. on pp. 2, 3, 7).

[LP15] Christoph Lenzen and Boaz Patt-Shamir. "Fast Partial Distance Estimation and Applications". In: *Symposium on Principles of Distributed Computing (PODC)*. 2015, pp. 153–162 (cit. on pp. i, 1, 2).

[LPS13] Christoph Lenzen and Boaz Patt-Shamir. "Fast Routing Table Construction Using Small Messages". In: *Symposium on Theory of Computing (STOC)*. 2013, pp. 381–390 (cit. on pp. i, 1, 2).

[NS14] Danupon Nanongkai and Hsin-Hao Su. "Almost-Tight Distributed Minimum Cut Algorithms". In: *International Symposium on Distributed Computing (DISC)*. 2014, pp. 439–453 (cit. on pp. 1, 22).

[Nan14] Danupon Nanongkai. "Distributed Approximation Algorithms for Weighted Shortest Paths". In: *Symposium on Theory of Computing (STOC)*. 2014, pp. 565–573 (cit. on pp. i, 1–3).

[PR00] David Peleg and Vitaly Rubinovich. "A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction". In: *SIAM Journal on Computing* 30.5 (2000). Announced at FOCS'99, pp. 1427–1442 (cit. on p. 1).

[PRS17] Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. "A Time- and Message-Optimal Distributed Algorithm for Minimum Spanning Trees". In: *Symposium on Theory of Computing, STOC*. 2017 (cit. on p. 1).

[PRT12] David Peleg, Liam Roditty, and Elad Tal. "Distributed Algorithms for Network Diameter and Girth". In: *ICALP (2)*. 2012, pp. 660–672 (cit. on p. 2).

[PT11] David Pritchard and Ramakrishna Thurimella. "Fast Computation of Small Cuts via Cycle Space Sampling". In: *ACM Transactions on Algorithms* 7.4 (2011). Announced at ICALP'08, 46:1–46:30 (cit. on p. 22).

[Pel00] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000. ISBN: 0-89871-464-8 (cit. on pp. 1, 7).

[Thu97] Ramakrishna Thurimella. "Sub-Linear Distributed Algorithms for Sparse Certificates and Biconnected Components". In: *Journal of Algorithms* 23.1 (1997). Announced at PODC'95, pp. 160–179 (cit. on p. 22).

[UY91] Jeffrey D. Ullman and Mihalis Yannakakis. "High-Probability Parallel Transitive-Closure Algorithms". In: *SIAM Journal on Computing* 20.1 (1991). Announced at SPAA'90, pp. 100–125 (cit. on p. 7).