Brief Announcement: An Improved Distributed Approximate Single-Source Shortest Paths Algorithm

Nairen Cao
nairen@ir.cs.georgetown.edu
Georgetown University
Washington D.C., USA

Jeremy T. Fineman
jfineman@cs.georgetown.edu
Georgetown University
Washington D.C., USA

Katina Russell
katina.russell@.cs.georgetown.edu
Georgetown University
Washington D.C., USA

ABSTRACT

This brief announcement presents an algorithm for $(1+\epsilon)$ approximate single-source shortest paths for directed graphs with non-negative real edge weights in the CONGEST model that runs in $\tilde{O}((n^{1/2}+D+n^{2/5+o(1)}D^{2/5})\log W/\epsilon^2)$ rounds, where W is the ratio between the largest and smallest non-zero edge weights.

CCS CONCEPTS

• **Theory of computation** → **Shortest paths**; **Distributed algorithms**.

KEYWORDS

distributed algorithm; congest model; shortest paths

ACM Reference Format:
Nairen Cao, Jeremy T. Fineman, and Katina Russell. 2021. Brief Announcement: An Improved Distributed Approximate Single-Source Shortest Paths Algorithm. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21), July 26–30, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3465084.3467945

1 INTRODUCTION

This paper presents an improved algorithm for approximate single-source shortest paths (SSSP) in the CONGEST model. The approximate single-source shortest paths problem is as follows. Given a directed graph $G=(V,E,w)$ with non-negative real edge weights, a source node $s \in V$, and approximation parameter ϵ, compute $(1+\epsilon)$ approximations of the distances from s to all nodes in the graph. In particular, for every node $v \in V$, the problem is to compute a distance estimate $\tilde{d}(s,v)$ satisfying $d(s,v) \le \tilde{d}(s,v) \le (1+\epsilon)d(s,v)$, where $d(s,v)$ is the true shortest-path distance from s to v.

The CONGEST model [12] is a distributed model consisting of an undirected communication network corresponding to an n-node undirected graph $N=(V,L)$. Each node has a unique $O(\log n)$-bit ID. Each link $(u,v) \in L$ indicates a bidirectional communication link between nodes u and v. Nodes communicate in synchronous rounds. In each round, every node may send and receive a $B=\Theta(\log n)$-bit message to and from each of its neighbors. The node may send different messages to each neighbor. The complexity of an algorithm in this model is measured by the number of rounds; the cost of local computation is ignored. Typical bounds depend on n and D, where D is the unweighted diameter of the undirected network N.

For graph problems in the CONGEST model, the network N is the same as the graph G except that in G the edges are directed, and in N the edges are undirected. For approximate SSSP in CONGEST, each node v must learn its distance estimate $\tilde{d}(s,v)$, but these distances need not be communicated back to s. To start, each node knows its set of incoming and outgoing edges and their weights, as well as whether it is the source node. Since every node can learn n in $O(D)$ rounds, we assume all nodes already know n.

Our algorithm follows the framework from Forster and Nanongkai [8] for distributed shortest paths. One of the steps in their framework involves computing shortest paths to a carefully selected subset of the vertices by simulating a parallel algorithm for SSSP in CONGEST. We achieve our improved bound by replacing the algorithm used in this step, instead adapting the parallel algorithm of Cao et al. (CFR) [2, 3] to the CONGEST model.

1.1 Related Work

Peleg and Rubinovich [13] showed that $\tilde{\Omega}(\sqrt{n}+D)$ is required for SSSP in the CONGEST model, where $\tilde{\Omega}$ hides polylogarithmic factors. The Bellman Ford algorithm for SSSP [5] can be used in the CONGEST model, and runs in $O(n)$ rounds. This result was the fastest known algorithm for a long time until Elkin [6] provided a randomized algorithm that runs in $\tilde{O}(n^{5/6})$ rounds for $D=\tilde{O}(\sqrt{n})$ and $\tilde{O}(D^{1/3}n^{2/3})$ rounds for larger D.

For undirected graphs, the approximate version of the problem has been well studied. The state-of-the-art is a deterministic algorithm from Becker et al. [1] which computes $(1+\epsilon)$-approximate shortest paths in $\tilde{O}(\sqrt{n}+D)$ rounds.

For directed graphs in the CONGEST model, recent progress has been made on the exact version of the problem. Ghaffari and Li [10] presented two randomized algorithms for graphs with polynomially bounded integer edge weights that run in $\tilde{O}(D^{1/4}n^{3/4})$ rounds and $\tilde{O}(n^{3/4+o(1)}+min\{n^{3/4}D^{1/6},n^{6/7}\}+D)$ rounds. At the same time, Forster and Nonongkai (FN) [8] provided two randomized algorithms for graphs with polynomially bounded integer edge weights that run in $\tilde{O}(\sqrt{nD})$ rounds and $\tilde{O}(n^{1/2}D^{1/4}+n^{3/5}+D)$ rounds. Chechik and Mukhtar [4] showed a randomized algorithm that achieves $\tilde{O}(\sqrt{n}D^{1/4}\log^2(W)+D)$ rounds. For approximate shortest paths, FN [8] show a randomized algorithm that runs in $\tilde{O}((n^{1/2}D^{1/4}+D)\log W/\epsilon)$ rounds.

1.2 Our results and technique

Our main result in this brief announcement is captured by the following theorem.

PODC '21, July 26–30, 2021, Virtual Event, Italy
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8548-0/21/07.
https://doi.org/10.1145/3465084.3467945

Theorem 1. *In the CONGEST model, there exists a randomized algorithm that solves $(1+\epsilon)$-approximate single-source shortest-paths problem for directed n-node graph G with non-negative real weights, in $\tilde{O}((\sqrt{n} + D + D^{2/5}n^{2/5+o(1)}) \log W/\epsilon^2)$ rounds, with high probability, where D is the undirected diameter of G and W is the ratio between heaviest and non-zero lightest edge weight.*

Across much of the range of network diameters, our algorithm's round complexity beats previous algorithms by a polynomial factor (albeit a very small one). Note that we only solve the approximate version of the problem, whereas some of the prior art solves that exact version. For polynomial bounded W, when the diameter is $\tilde{o}(n^{1/4})$, our algorithm only takes $\tilde{O}(\sqrt{n})$ rounds, which matches the lower bound. When the diameter of the network D is $O(\sqrt{n})$, the best previous result was $\tilde{O}(n^{5/8})$ rounds [8], whereas our algorithm completes in $\tilde{O}(n^{3/5+o(1)})$ rounds.

2 PRELIMINARIES

For a graph $G = (V, E, w)$, V is the set of vertices, E is the set of edges, and $w : E \longrightarrow R$ is a weight function. We consider graphs with non-negative real weighted edges and we assume the lightest non-zero edge weight is 1 and the heaviest edge weight is W. Otherwise, we can normalize all edge weights. The number of nodes is $n = |V|$. For a subset $V' \subset V$, we denote the induced graph on V' as $G[V']$.

For a pair of nodes $u, v \in V$, the shortest path distance in G from u to v is denoted $d_G(u, v)$. The h-hop shortest path distance from u to v in G is the shortest path from u to v that contains at most h edges and is denoted $d_G^{(h)}(u, v)$. We omit the subscripts in $d_G(u, v)$ and $d_G^{(h)}(u, v)$ when G is clear from the context.

Distance d related node sets. For a directed graph $G = (V, E)$ and vertices $u, v \in V$, denote $R_d^+(G, v) = \{u | d_G(v, u) \leq d\}$ and $R_d^-(G, v) = \{u | d_G(u, v) \leq d\}$ to be the set of nodes which can be reached by v within distance d, and can reach v within d-distance, respectively.

The following lemma is a standard result for distributed computation in the CONGEST model.

Lemma 2. *[12] Suppose each $v \in V$ holds $k_v \geq 0$ messages of $O(\log n)$ bits each, for a total of $K = \sum_{v \in V} k_v$. Then all nodes in the network can receive these K messages within $O(K + D)$ rounds.*

3 ALGORITHM

Next we present an overview of the algorithm, which extends the FN [8] framework. The algorithm is parameterized by α, to be set later. Steps 1, 3 and 5 are the same as FN [8], and step 2 is similar. In step 2, FN computes distance estimates from each skeleton node to each node in the original graph. Our step 2 does the same computation and additionally computes the distance from each node in the original graph to each skeleton node. The additional distances estimates are used in the computation of step 4. The main difference in the algorithm is step 4. Both algorithms solve SSSP on the skeleton graph, however we use a different algorithm to compute SSSP. The algorithm for computing step 4 is discussed in the next section.

(1) Select each node $v \in V$ to be in the set of skeleton nodes S with probability $\tilde{O}(\alpha/n)$. Add the source s to S. If $|S| > \tilde{\Omega}(\alpha)$, abort the algorithm.
(2) Let $g = \tilde{O}(n/\alpha)$. For a pair of nodes u, v, define a $(1 + O(\epsilon))$-approximate g-hop distance estimate $\tilde{d}(u, v)$ to be an estimate of $d_G^{(g)}(u, v)$ such that $d_G^{(g)}(u, v) \leq \tilde{d}(u, v) \leq (1 + O(\epsilon))d_G^{(g)}(u, v)$. For each $u \in S, v \in V$, both u and v learn $\tilde{d}(u, v)$.
(3) Construct the skeleton graph $G_S = (S, E_S, w_s)$, where $E_S = SxS$, and $w_s(u, v) = \tilde{d}(u, v)$. For nodes $u, v \in S$, both u and v know $w_s(u, v)$.
(4) Solve approximate SSSP on the skeleton graph G_S with s as the source, i.e. for each $v \in S$, compute $d'(s, v)$, where $d_{G_S}(s, v) \leq d'(s, v) \leq (1 + O(\epsilon))d_{G_S}(s, v)$.
(5) For each $v \in V$, compute $\hat{d}(s, v) = min_{u \in S}(d'(s, u) + \tilde{d}(u, v))$.

In steps 2 and 3 of the algorithm, we require that both nodes u and v know the distance estimate $\tilde{d}(u, v)$. FN does not have this requirement. Also, the distance estimates computed in step 2 should be consistent, meaning that the distance estimate $\tilde{d}(u, v)$ that u knows should be equal to the distance estimate $\tilde{d}(u, v)$ that v knows.

The correctness of the algorithm follows from FN [8].

Theorem 3. *For any directed input graph $G = (V, E, w)$ with fixed source node s, the algorithm above consisting of Steps 1–5 computes, for every node $v \in V$, a distance estimate $\hat{d}(s, v)$ such that $d_G(s, v) \leq \hat{d}(s, v) \leq (1 + O(\epsilon))d_G(s, v)$.*

3.1 Step 4

Steps 1, 3 and 5 are the same as FN [8]. In Step 2, FN computes distance estimates from each skeleton node to each node in the original graph. Our step 2 does the same computation and additionally computes the distance from each node in the original graph to each skeleton node. The additional distances estimates are used in the computation of step 4.

The main difference in our algorithm is step 4. Both algorithms solve SSSP on the skeleton graph, however we use a different technique from FN. We adapt an algorithm from CFR [2] for parallel approximate shortest paths to solve approximate SSSP on the skeleton graph. Their algorithm constructs a $(\beta = n^{1/2+o(1)}, \epsilon)$-hopset, and then solves parallel approximate shortest paths on the graph with the hopset edges added. For a directed graph $G = (V, E)$, a (β, ϵ)-hopset is a set of weighted edges E' such that, for each pair of nodes $u, v \in V$, there exists a path p from u to v that contains at most β edges and $d_G(u, v) \leq d_{G'}(p) \leq (1 + \epsilon)d_G(u, v)$, where $G' = (V, E \cup E')$. For step 4, we construct a (β, ϵ)-hopset using the CFR algorithm, then run BFS on the skeleton graph with the hopset edges added to the graph to solve approximate SSSP. Next we will describe computing shortest paths on the limited-depth skeleton graph. Then we will give an overview of the CFR algorithm, and discuss how to make it work in the CONGEST model. In the full version of the paper, we give more details of the CFR algorithm in the CONGEST model.

One of the difficulties in computing SSSP on the skeleton graph is that an edge $(u, v) \in G_S$ may not be an edge in the original graph and thus not have a direct communication link. We will require that each node in the skeleton graph knows its incoming and outgoing edges. Once we have a limited depth skeleton graph, we can simulate BFS as follows.

Lemma 4. *Given a graph $G = (V, E)$ with diameter D, and a skeleton graph G_T over a subset of nodes $T \subset V$ with integer weights, for a source node $s \in T$, there is an algorithm such that each node $v \in R_h^+(G_T, s)$, including s itself, learns the distance $d_{G_T}(s, v)$ in $O(Dh + |R_h^+(G_T, s)|)$ rounds and $O(|R_h^+(G_T, s)|)$ congestion on each edge.*

Proof. We simulate BFS on the skeleton graph. The algorithm is divided into levels, and at each level i, the goal is for nodes at distance i to learn their distance from s. To start, each node $v \in T \backslash \{s\}$ sets $d(s, v) = \infty$, and s sets $d(s, s) = 0$. At level $i \in [0, h]$, if a node v learns its distance $d(s, v) = i$, it will broadcast $d(s, v) = i$ to the whole graph. By Lemma 2, each level takes $O(D + K_i)$ time where $K_i = |R_i^+(G_T, s) \backslash R_{i-1}^+(G_T, s)|$. Thus, each node v can learn its distance in $O(Dh + |R_h^+(G_T, s)|)$ rounds. After h levels, all nodes whose distance have been updated broadcast their distances and s learns the distance updates. This broadcast can be done in $O(D + |R_h^+(G_T, s)|)$ rounds. The total amount of information sent is $O(|R_h^+(G_T, s)|)$, and thus the algorithm takes $O(Dh + |R_h^+(G_T, s)|)$ rounds and $O(|R_h^+(G_T, s)|)$ congestion on each edge. □

Overview of CFR Algorithms. [2, 3]. The algorithms from CFR are based on two prior algorithms for parallel reachability by Fineman [7] and Jambulapati et al. (JLS) [11]. Their first algorithm is for constructing a $(\beta = n^{1/2+o(1)}, \epsilon)$-hopset with $\tilde{O}(n)$ edges in $\tilde{O}(m)$ work [2]. They also give a parallel algorithm constructing the hopset and use it to solve parallel approximate shortest paths on the graph with the hopset edges added. In a follow-up paper, they extend this result to construct a $(\beta = \tilde{O}(n^{1/2+o(1)/\rho}, \epsilon)$-hopset of size $\tilde{O}(n\rho^2)$ in $\tilde{O}(m\rho^2 n\rho^4)$ work and $\tilde{O}(n^{1/2+o(1)}/\rho)$ span, where $\rho \in [1, \sqrt{n}]$ is a tradeoff parameter [3]. Next we give an overview of their hopset algorithm [2]. The work span tradeoff algorithm [3] works similarly.

The CFR algorithm for constructing hopset runs as follows. The algorithm is parameterized by a distance guess D, and shortcuts all paths of this distance. It then repeats for all guesses of the distance. At each level of recursion the algorithm chooses some nodes to be pivots and some to be shortcutters, where the pivots are a subset of the shortcutters. Each shortcutter x computes the set $R_d^+(G, x)$ and adds edges from x to each node $v \in R_d^+(G, x)$ with weight equal to the distance from x to v to the hopset. Symmetrically, each shortcutter x computes the set $R_d^-(G, x)$ and adds edges from each node $v \in R_d^+(G, x)$ to x with weight equal to the distance from v to x to the hopset. Next each pivot w adds the label w_{Anc} to each node in $R_d^+(G, w)$, the label w_{Desc} to each node in $R_d^-(G, w)$, and an X label to any node in $R_d^+(G, w) \cap R_d^-(G, w)$. The graph is partitioned in to subgraphs such that two nodes are in the same subgraph if and only if they have the exact same set of labels, and any subgraph that contains an X label is removed from the graph. Each subgraph is recursed on with a decreased search distance. Finally the entire algorithm is repeated for each possible guess of the distance.

Adapting CFR to the CONGEST model. The CFR algorithm follows a similar structure to the one of the JLS algorithm [11] for parallel single-source reachability. JLS extends their algorithm to the CONGEST model. We use their techniques to adapt CFR to the CONGEST model. Next we will give an overview of the CONGEST JLS algorithm, and then discuss CFR in the CONGEST model.

The JLS algorithm for reachability in the CONGEST model builds a skeleton graph and then essentially simulates their parallel algorithm, which adds extra edges to the graph in order to reduce the diameter. The simulate their parallel algorithm by each node broadcasting the information from its computation, such as when the node is reached in BFS searches, the subproblem labels and IDs, the new shortcut edges, and when it becomes a shortcutter. There are two important parameters for the CONGEST JLS algorithm. First, each BFS takes $O(Dh + \sum_s |R_h^+(G, s)|)$ rounds on the skeleton graph, so the depth h of the BFS must be limited. Second, the total number of new edges added to the graph is $\sum_s |R_h^+(G, s)|$. They balance these parameters and get $\tilde{O}(D\alpha^{1/2+o(1)})$ and $\tilde{O}(\alpha)$, which are the costs of BFS and broadcasting all the new edges, respectively. After simulating their parallel algorithm on the skeleton graph, they simulate BFS on the skeleton graph and locally compute reachability for each node. Their algorithm runs in $D^{2/3}n^{1/3+o(1)}$ rounds.

We use the same simulation technique as JLS to adapt CFR to the CONGEST model. The total cost of step 4 is $\tilde{O}(\frac{\alpha\rho^2}{\epsilon^2}\log W + \frac{D\alpha^{1/2+o(1)}}{\rho\epsilon}\log W)$. The details of the algorithm and the running time are deferred to the full version of the paper. For $D = O(n^{1/4})$, the CFR [2] algorithm can be used to achieve the desired bound. For $D = \omega(n^{1/4})$, we use the CFR algorithm that has a work span tradeoff [3]. Section 3.2 gives the specific parameter settings.

3.2 Cost of the Algorithm

Steps 1-3 and 5 can be performed the same as FN [8]. The complexity of the algorithm is as follows. Step 1 takes $\tilde{O}(\alpha + D)$ rounds to broadcast S to all nodes. In step 2, computing the distance estimates can be done in $\tilde{O}(\alpha + n\log W/(\alpha\epsilon) + D)$ rounds. FN only computes forwards distance estimates, but the backwards distance estimates can be computed symmetrically. Step 3 computes the skeleton graph and broadcasts it to the graph, which can be done in $\tilde{O}(\alpha + D)$ rounds. In the full version of the paper we show that step 4 can be implemented in $\tilde{O}(\frac{\alpha\rho^2}{\epsilon^2}\log W + \frac{D\alpha^{1/2+o(1)}}{\rho\epsilon}\log W)$ rounds, where ρ is a parameter which is in range $[\tilde{O}(1), \alpha^{1/2+o(1)}]$. Finally step 5 is computed internally for each node.

The total number of rounds for the algorithm is $\tilde{O}(D + \frac{n}{\alpha\epsilon}\log W + \frac{\alpha\rho^2\log W}{\epsilon^2} + \frac{D\alpha^{1/2+o(1)}\log W}{\rho\epsilon})$, which reduces to as follows:

Case 1. If $D = o(n^{1/4})$, set $\rho = \tilde{\Theta}(1)$ and $\alpha = \tilde{\Theta}(\sqrt{n})$. The whole algorithm takes $\tilde{O}(\sqrt{n}\log W/\epsilon^2)$ rounds.

Case 2. If $D = \omega(n^{2/3})$, set $\rho = \alpha^{1/2+o(1)}$ and $\alpha = \tilde{O}(n^{1/3})$. The whole algorithm takes $\tilde{O}(D\log W/\epsilon^2)$ rounds.

Case 3. Otherwise, set $\rho = \tilde{O}(\frac{D^{2/5}}{n^{1/10-o(1)}})$ and $\alpha = \tilde{O}(\frac{n^{3/5+o(1)}}{D^{2/5}})$. The whole algorithm takes $\tilde{O}(D^{2/5}n^{2/5+o(1)}\log W/\epsilon^2)$ rounds.

Combining all three cases together, the algorithm solves approximate SSSP in $\tilde{O}(\sqrt{n} + D + D^{2/5}n^{2/5+o(1)})\log W/\epsilon^2)$ rounds. This shows the running time of Theorem 1.

Acknowledgements

This research was supported in part by NSF grants CCF-1918989 and 1718700.

REFERENCES

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

[2] Nairen Cao, Jeremy T. Fineman, and Katina Russell. Efficient construction of directed hopsets and parallel approximate shortest paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, page 336–349, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Nairen Cao, Jeremy T. Fineman, and Katina Russell. Improved work span tradeoff for single source reachability and approximate shortest paths. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, page 511–513, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Shiri Chechik and Doron Mukhtar. Single-source shortest paths in the congest model with improved bound. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, page 464–473, New York, NY, USA, 2020.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[6] Michael Elkin. Distributed exact shortest paths in sublinear time. *J. ACM*, 67(3), May 2020.

[7] Jeremy T. Fineman. Nearly work-efficient parallel algorithm for digraph reachability. In *Proceedings of the 50th Annual ACM SIGACT Symposium on the Theory of Computation*, pages 457–470, 2018.

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

[9] Mohsen Ghaffari. Near-optimal scheduling of distributed algorithms. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, page 3–12, New York, NY, USA, 2015. Association for Computing Machinery.

[10] Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 431–444, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Y. P. Liu, A. Jambulapati, and A. Sidford. Parallel reachability in almost linear work and square root depth. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science*, pages 1664–1686, 2019.

[12] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[13] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30:1427–1442, 01 2000.

APPENDIX

The hopsets algorithm is given in Algorithm 1. This algorithm is essentially a simulation of the CFR parallel hopsets algorithm [2]. The input of the algorithm is a skeleton graph, and the output of the algorithm is the skeleton graph with a (β', ϵ)-hopset added to the graph. The parameters $\delta = O(\epsilon), k = \Theta(\log n), \lambda = \Theta(1), c = O(\log_k(1/\epsilon))$ are set the same as CFR [2] and the parameter L can be set to any integer greater than $17 - \log_k \epsilon$. Choosing a larger L results in a smaller hopbound but larger size hopset. The correctness of the algorithm follows directly from CFR [2], but we will show the running time of the algorithm. The remainder of this section will show the following theorem.

Theorem 5. *Given a graph $G = (V, E)$ with diameter D, and a n-node skeleton graph G_T over a subset of nodes $T \subset V$, for a source node $s \in T$, Algorithm 1 takes $\tilde{O}(Dn^{1/2+o(1/\log k)} \log W/(k^{L/2}\epsilon) + nk^{L+1} \log W/\epsilon^2)$ rounds w.h.p. in the CONGEST model.*

By setting $k = O(\log n)$ and $\rho = k^{L/2}$, for a α-node graph, the bound from Theorem 5 is $\tilde{O}(D\alpha^{1/2+o(1)} \log W/(\rho\epsilon) + \frac{\alpha\rho^2 \log W}{\epsilon^2})$, which is the bound that we use in Section 3.2. To prove Theorem 5, we will use the following Lemma which is a standard result in the CONGEST model.

Lemma 6. *[9] Consider k distributed algorithms $A_1, ..., A_k$. Let **dilation** be such that each algorithm A_i finishes in dilation rounds if it runs individually. Let **congestion** be such that there are at most congestion messages, each of size $O(\log n)$, sent through each edge (counted over all rounds), when we run all algorithms together. There is a distributed algorithm that can execute $A_1, ..., A_k$ in $\tilde{O}($**dilation**$+$**congestion**$)$ rounds in the CONGEST model.*

Algorithm 1 first sets up the hopset H and the hopbound β'. It repeats Lines 4-36 $\lambda \log^2 n$ times. Line 4 starts a loop that attempts to shortcut paths $\hat{P}$ containing β' to $2\beta'$ edges, where $w(\hat{P}) \in [2^i, 2^{i+1})$. Next, in Lines 6-7, the algorithm rounds each edge up. After this step, a path $\hat{P}$ will contain only integer weighted edges and the new length of $\hat{P}$ is $\hat{w}(\hat{P})$ which is at most $O(\beta')$. Thus, when the algorithm later does BFS, the depth β' can be bounded.

Next the algorithm starts adding edges to the hopset. Lines 9-10 gives each node a label $l(v)$ and from Line 11 to 13, each node will do BFS with depth $\tilde{O}(\beta'/\delta)$. Recall that Lemma 4, shows how to run single-source BFS. By combining Lemma 4 with Lemma 6, the algorithm can run BFS from multiple sources. Using these two Lemmas, the number of rounds for Lines 11-13 is $\tilde{O}(Dh + \sum_{v,\ell(v)\leq l} |R_h^+(G, s) \cup R_h^-(G, s)|$, where $h = 8(1+\epsilon)\beta'/\delta = O(\beta'/\epsilon)$. Since each node reached in a BFS adds an edge to the hopset, bounding the size of the hopset gives a bound on the number of nodes reached in all the searches. Fortunately, CFR [2] gives following theorem on the size of the hopset, which gives an upper bound on the congestion of $\sum_{v,\ell(v)\leq l} |R_h^+(G, s) \cup R_h^-(G, s)| = \tilde{O}(nk^{L+1}/\epsilon^2)$.

Theorem 7. *[2] One execution of Line 5-34 DHopset$(G = (V, E))$ with parameter k, where $n = |V|, m = |E|$, produces a $(n^{1/2+O(1/\log k)}/k^{L/2}, \epsilon)$-hopset of size $\tilde{O}(nk^{L+1}/\epsilon^2)$ with high probability.*

Next, in Lines 14-34, the algorithm divides the graph into smaller graphs and adds edges on the smaller skeleton graphs. CFR implements Lines 14-34 recursively, while Algorithm 1 is iterative, which is done for clarity. At the start of the algorithm, there is only one skeleton graph $\hat{G}$. Then in each iteration, the algorithm runs Lines 18-34 on each skeleton graph. Lines 18-20 are the same as Lines 11-13, except for different nodes. Since the congestion is already bounded, the upper bound of the running time is the same. Lines 21-34 try to construct smaller skeleton graphs. For each pivot, i.e. nodes v where $\ell(v) = r$, the algorithm first chooses a "good" distance, where "good" is defined in Line 25. In Line 25, each pivot runs BFS to depth $\rho_{max}D_r$. Again, we can use the analysis from CFR to bound the congestion.

Lemma 8. *[2] Consider an execution of Line 5-34 DHopset$(G = (V, E))$. W.h.p. at least $1 - n^{-0.7\lambda+3}$, the following holds for all $v \in \tilde{G}$,*

$$|R^+_{\rho_{max}D_r}(\tilde{G}, v)| \leq nk^{-r}, |R^-_{\rho_{max}D_r}(\tilde{G}, v)| \leq nk^{-r}.$$

Lemma 8 implies that for each r, and each $\tilde{G}$, the congestion for each v in Lines 19-20 is at most $O(nk^{-r})$. Then we can count the number of nodes in all skeleton graphs $\tilde{G}$ that run BFS. If we knew the total number of nodes for each r, we could count the number of nodes with label $\ell(v) = r$, since the probability of each node being label $\ell(v) = r$ is $\tilde{O}(k^r/n)$. The difficulty of counting the nodes in

Algorithm 1 Distributed hopset algorithm for weighted directed skeleton graphs. δ, k, λ, c, L are parameters.

```
1:  function DHopset($G = (V, E)$)
2:      $H \leftarrow \emptyset, \beta' \leftarrow \beta/(\lambda^2 k)^{(L-17+\log_k \epsilon)/2}$
3:      repeat $\lambda \log^2 n$ times
4:          for each $i \in [-2, \log(n^2 W)]$
5:              $\hat{w} = \delta \cdot 2^{i-1}/\beta, \hat{H}' \leftarrow \emptyset, D = 4(1+\delta)\beta'/(\delta k^c)$
6:              for each $v \in V, e = (u, v)$ or $(v, u) \in E$      ▷ Construct a new skeleton graph $\hat{G} = (\hat{V} = V, \hat{E} = E)$
7:                  $\tilde{w}(e) = \begin{cases} +\infty & \text{if } w(e) \geq 2^{i+1} \\ \left\lceil \frac{w(e)}{\hat{w}} \right\rceil & \text{if } w(e) < 2^{i+1} \\ 1 & \text{if } w(e) = 0 \end{cases}$
8:              for each $v \in \hat{V}$
9:                  for each $i' \in [0, \log_k n]$
10:                     With probability $(\lambda k^{i'+1} \log n)/n$, set $\ell(v)$ to $i'$, break if setting successfully.
11:                 if $\ell(v) \leq L$ then      ▷ Run multiple source BFS on $\hat{G}$
12:                     for each $u \in R^+_{8(1+\delta)\beta'/\delta}(\hat{G}, v)$ add edge $(v, u)$ to $\hat{H}'$ with weight $d_{\hat{G}}(v, u)$
13:                     for each $u \in R^-_{8(1+\delta)\beta'/\delta}(\hat{G}, v)$ add edge $(u, v)$ to $\hat{H}'$ with weight $d_{\hat{G}}(u, v)$
14:             $S_0 = \{\hat{G}\}$
15:             for each $r \in [0, \log_k n]$
16:                 $S_{r+1} = \emptyset, D_r \leftarrow D/(\lambda^r k^{r/2})$
17:                 for each $\tilde{G} = (\tilde{V}, \tilde{E}) \in S_r$
18:                     for each $v \in V$ with $\ell(v) = r + L$      ▷ Run multiple source BFS on all $\tilde{G}$
19:                         for each $u \in R^+_{32\lambda^2 k^2 D_r \log^2 n}(\tilde{G}, v)$ add edge $(v, u)$ to $H'$ with weight $dist_{\tilde{G}}(v, u)$
20:                         for each $u \in R^-_{32\lambda^2 k^2 D_r \log^2 n}(G, v)$ add edge $(u, v)$ to $H'$ with weight $dist_{\tilde{G}}(u, v)$
21:                     for each $v \in \tilde{V}$ with $\ell(v) = r$      ▷ Construct next level skeleton graphs
22:                         Choose $\sigma_v$ uniformly at random from $[1, 4\lambda^2 k \log^2 n]$, set $\rho_{max} = 20\lambda^2 k^2 \log^2 n$
23:                         for each $u \in R^+_{\rho_{max} D_r}(\tilde{G}, v)$ , add edge $(v, u)$ with weight $dist_{\tilde{G}}(v, u)$
24:                         for each $u \in R^-_{\rho_{max} D_r}(\tilde{G}, v)$ , add edge $(v, u)$ with weight $dist_{\tilde{G}}(u, v)$
25:                         Minimize $|R_{(\rho_v+1)D_r}(\tilde{G}, v) \backslash R_{(\rho_v-1)D_r}(\tilde{G}, v)|$ such that $\rho_v \in [16\lambda^2 k^2 \log^2 n + 4k(\sigma_v - 1), 16\lambda^2 k^2 \log^2 n + 4k\sigma_v)$
26:                         for each $u \in R^+_{\rho_v D_r}(\tilde{G}, v)$ add label $v^{Des}$ to vertex $u$
27:                         for each $u \in R^-_{\rho_v D_r}(\tilde{G}, v)$ add label $v^{Anc}$ to vertex $u$
28:                         for each $u \in R^+_{\rho_v D_r}(\tilde{G}, v) \cap R^-_{\rho_v D_r}(\tilde{G}, v)$ add label $X$ to vertex $u$
29:                         $V_v^{\textbf{fringe}} \leftarrow R_{(\rho_v+1)D_r}(\tilde{G}, v) \backslash R_{(\rho_v-1)D_r}(\tilde{G}, v)$
30:                         $S_{r+1} = S_{r+1} \cup \{\tilde{G}[V_v^{\textbf{fringe}}]\}$      ▷ If $u$ in $v$'s fringe node, mark it
31:                     for each $u \in V$ that has a $X$ label, remove $u$
32:                     $V_1, V_2, ..., V_t \leftarrow$ partition based on labels
33:                     for each $i \in [1, t]$
34:                         $S_{r+1} = S_{r+1} \cup \{\tilde{G}[V_i]\}$      ▷ if $u$ is in $\tilde{G}[V_i]$, mark it
35:             $H \leftarrow H \cup (\hat{w} \cdot \hat{H}')$
36:         $E \leftarrow E \cup H$
37:     return $H$
```

this way comes from the overlapping skeleton graphs, and thus the total number of nodes in all $\tilde{G} \in S_r$ could be quite large. In the following Lemma, CFR [2] gives an upper bound of nodes in all $\tilde{G} \in S_r$ for each r.

Lemma 9. *[2] Consider an execution of Line 15-34 DHopset($G = (V, E)$). For all $r \in [0, \log_k n]$, the number of nodes in all $\tilde{G} \in S_r$ is $\tilde{O}(n)$ with high probability.*

Thus, the congestion for all pivots in Lines 19-20 is $\tilde{O}(n)$. This implies that multiple source BFS performed by all the pivots runs in $\tilde{O}(Dh + n)$ rounds. After ρ_v is decided and broadcast to each node u, node u adds appropriate label v^{Des}, v^{Anc}, or X to itself. Lines 29-30 construct the fringe skeleton graph and Lines 33-34 partition the graph into smaller graphs. One last complication is that when the algorithm runs on the smaller skeleton graphs, the graphs may overlap with one another. The algorithm must distinguish different skeleton graph searches. The algorithm can overcome this issue by adding marks to each smaller skeleton graph. Also, if a node is in a fringe problem, it marks itself fringe and marks which fringe problem it is in. Likewise for the core problem, a node marks itself core and marks which labels it gets. When then algorithm runs BFS, each node must append its marks to the messages it sends out. CFR [2] has the following bound on the size of the marks for each r,

Lemma 10. *Consider an execution of Line 5-34 DHopset*$(G = (V, E))$. *For all* $v \in \tilde{V}, r \in [0, \log_k n]$, *w.h.p., the number of pivots* $\ell(u) = r$, *such that* $v \in R_{(\rho_u+1)D_r}(\tilde{G}, u)$ *is* $O(k)$.

Thus, the overhead for marks at each level is $O(k)$ and $\tilde{O}(k)$ for all r. Lines 35 and 36 add the hopset edges to the original graph G. This can be done easily since each node knows the incoming and outgoing edges of the hopset from running BFS.

By combining all these steps together, Lines 5-34 take $\tilde{O}(Dn^{1/2+O(1/\log k)}/(k^{L/2}\epsilon$ $nk^{L+1}/\epsilon^2)$ rounds. There is an additional $\log W$ factor from guessing the weight of the path up to $n^2 W$. This gives the running time in Theorem 5.

Once the (β', ϵ)-hopset is constructed and the hopset edges are added to the skeleton graph, we can solve approximate SSSP on the new skeleton graph. By guessing the SSSP distance and rounding up the weight of each edge, we can again do BFS on the skeleton graph with depth $O(\beta'/\epsilon)$. Recall there will be at most n nodes related to s in the skeleton graph. Therefore, BFS will take $\tilde{O}(Dn^{1/2+O(1/\log k)} \log W/(k^{L/2}\epsilon) + n \log W)$ rounds. This is at most the running time of Algorithm 1, which implies Theorem 5 gives up an upper bound for the running time of step 4.