Distributed Weighted All Pairs Shortest Paths Through Pipelining

Udit Agarwal
Dept. of Computer Science, UT Austin
Austin, TX, USA
udit@cs.utexas.edu

Vijaya Ramachandran
Dept. of Computer Science, UT Austin
Austin, TX, USA
vlr@cs.utexas.edu

***Abstract*—We present new results for the distributed computation of all pairs shortest paths (APSP) in the CONGEST model in an n-node graph with moderate non-negative integer weights. Our methods can handle zero-weight edges which are known to present difficulties for distributed APSP algorithms. The current best deterministic distributed algorithm in the CONGEST model that handles zero weight edges is the $\tilde{O}(n^{3/2})$-round algorithm of Agarwal et al. [3] that works for arbitrary edge weights.**

Our new deterministic algorithms run in $\tilde{O}(W^{1/4} \cdot n^{5/4})$ rounds in graphs with non-negative integer edge-weight at most W, and in $\tilde{O}(n \cdot \triangle^{1/3})$ rounds for shortest path distances at most $\triangle$. These algorithms are built on top of a new pipelined algorithm we present for this problem that runs in at most $2n\sqrt{\triangle} + 2n$ rounds. Additionally, we show that the techniques in our results simplify some of the procedures in the earlier APSP algorithms for non-negative edge weights in [3], [13]. We also present new results for computing h-hop shortest paths from k given sources, including the notion of consistent h-hop shortest path trees, and we present an $\tilde{O}(n/\epsilon^2)$-round deterministic $(1+\epsilon)$ approximation algorithm for graphs with non-negative $poly(n)$ integer weights, improving results in [16], [18] that hold only for positive integer weights.

I. INTRODUCTION

Designing distributed algorithms for various network and graph problems related to shortest paths [3], [12], [13], [15], [17] is a extensively studied area of research. The CONGEST model (described in Sec I-B) is a widely-used model for these algorithms, see [3], [8], [13], [17]. In this paper we consider distributed algorithms in the CONGEST mode for computing all pairs shortest paths (APSP) and related problems in a graph with non-negative edge weights. In this model, we need to develop a distributed algorithm where each node in the graph computes the shortest path distance to it from each source as well as the last edge on such a shortest path.

In sequential computation, shortest paths can be computed much faster in graphs with non-negative edge-weights (including zero weights) using the classic Dijkstra's algorithm [7] than in graphs with negative edge weights. Additionally, negative edge-weights raise the possibility of negative weight cycles in the graph, which usually do not occur in practice, and hence are not modeled by real-world weighted graphs. Thus, in the distributed setting, it is of importance to design fast shortest path algorithms that can handle non-negative edge-weights, including edges of weight zero.

This work was supported in part by NSF Grant CCF-1320675. The first author's research was also partially supported by a UT Austin Graduate School Summer Fellowship.

The presence of zero weight edges creates challenges in the design of distributed algorithms as observed in [13]. (We review related work in Section I-C.) One approach used for positive integer edge weights is to replace an edge of weight d with d unweighted edges and then run an unweighted APSP algorithm such as [12], [17] on this modified graph. This approach is used in approximate APSP algorithms [16], [18]. However such an approach fails when zero weight edges may be present. There are a few known algorithms that can handle zero weights, such as the $\tilde{O}(n^{3/2})$-round deterministic APSP algorithm of Agarwal et al. [3] for graphs with arbitrary edge weights, and the randomized weighted APSP algorithms of Huang et al. [13] (for polynomially bounded non-negative integer edge weights), and of Elkin [8] (and very recently of Bernstein and Nanongkai [5]) for arbitrary edge weights. However no previous sub-$n^{3/2}$-round deterministic algorithm was known for weighted APSP that can handle zero weights.

A. Our Results

All of our results hold for both directed and undirected graphs and we will assume w.l.o.g. that G is directed. Here is a summary of our results.

1. A Pipelined APSP Algorithm for Weighted Graphs. An h-hop shortest path from u to v in G is a path from u to v of minimum weight among all paths with at most h edges (or *hops*). The central algorithm we present is for computing h-hop APSP, or more generally, (h, k)-SSP, the h-hop shortest path problem for k given sources (this problem is called the k-source short-range problem in [13]). We sometimes add an additional constraint that the shortest paths have distance at most $\triangle$ in G.

Our pipelined Algorithm 1 in Sec. II is compact and easy to implement, and has no large hidden constant factors in its bound on the number of rounds. It can be viewed as a (substantial) generalization of the pipelined method for unweighted APSP given in [12], which is a refinement of [17]. Our algorithm uses key values that depend on both the weighted distance and the hop length of a path, and it can store multiple distance values for a source at a given node, with the guarantee that the shortest path distance will be identified. This algorithm (Alg. 1) achieves the bounds in the following theorem.

Theorem I.1. *Let $G = (V, E)$ be a directed or undirected edge-weighted graph, where all edge weights are non-negative integers (with zero-weight edges allowed). The following deterministic bounds can be obtained in the* CONGEST *model*

1530-2075/19/$31.00 ©2019 IEEE
DOI 10.1109/IPDPS.2019.00014

IEEE computer society

for shortest path distances at most $\triangle$.
(i) (h,k)-SSP *in* $2\sqrt{\triangle kh}+k+h$ *rounds.*
(ii) APSP *in* $2n\sqrt{\triangle}+2n$ *rounds.*
(iii) k-SSP *in* $2\sqrt{\triangle kn}+n+k$ *rounds.*

2. Faster Deterministic APSP for Non-negative, Moderate Integer Weights. We improve on the bounds given in (ii) and (iii) of Theorem I.1 by combining our pipelined Algorithm 1 with a modified version of the deterministic APSP algorithm in [3]. This gives our improved Algorithm 3, with the bounds stated in the following Theorems I.2 and I.3. To obtain these improved bounds we also present an improved deterministic distributed algorithm to find a *blocker set* [3].

In our improved blocker set method we define the notion of a *consistent collection of h-hop trees, CSSSP* (Definition III.3 in Section III-A), and a simple method to compute such a collection. This result may be of independent interest.

Theorem I.2. *Let $G=(V,E)$ be a directed or undirected edge-weighted graph, where all edge weights are non-negative integers bounded by W (with zero-weight edges allowed). The following deterministic bounds can be obtained in the* CONGEST *model.*
(i) APSP in $O(W^{1/4}\cdot n^{5/4}\log^{1/2} n)$ rounds.
(ii) k-SSP in $O(W^{1/4}\cdot nk^{1/4}\log^{1/2} n)$ rounds.

Theorem I.3. *Let $G=(V,E)$ be a directed or undirected edge-weighted graph, where all edge weights are non-negative integers (with zero edge-weights allowed), and the shortest path distances are bounded by $\triangle$. The following deterministic bounds can be obtained in the* CONGEST *model.*
(i) APSP in $O(n(\triangle\log^2 n)^{1/3})$ rounds.
(ii) k-SSP in $O((\triangle kn^2\log^2 n)^{1/3})$ rounds.

The range of values for W and Δ for which our results in Theorem I.2 and I.3 improve on the $\tilde{O}(n^{3/2})$ deterministic APSP bound of Agarwal et al. [3] are stated in the following Corollary.

Corollary I.4. *Let $G=(V,E)$ be a directed or undirected edge-weighted graph with non-negative edge weights (and zero-weight edges allowed). The following deterministic bounds hold for the* CONGEST *model for $1\geq\epsilon\geq 0$.*
(i) If the edge weights are bounded by $W=n^{1-\epsilon}$, then APSP can be computed in $O(n^{3/2-\epsilon/4}\log^{1/2} n)$ rounds.
(ii) For shortest path distances bounded by $\Delta=n^{3/2-\epsilon}$, APSP can be computed in $O(n^{3/2-\epsilon/3}\log^{2/3} n)$ rounds.

The corresponding bounds for the weighted k-SSP problem are: $O(n^{5/4-\epsilon/4}k^{1/4}\log^{1/2} n)$ (when $W=n^{1-\epsilon}$) and $O(n^{7/6-\epsilon/3}k^{1/3}\log^{2/3} n)$ (when $\Delta=n^{3/2-\epsilon}$). Note that the result in (i) is independent of the value of Δ (it depends only on W) and the result in (ii) is independent of the value of W (it depends only on Δ).

3. Simplifications to Earlier Algorithms. Our techniques give simpler methods for some of procedures in two previous distributed weighted APSP algorithms that handle zero weight edges. In Section II-C we present simple deterministic algorithms that match the congest and dilation bounds in [13] for two of the three procedures used there: the *short-range* and *short-range-extension* algorithms. Our simplified algorithms are both obtained using a streamlined single-source version of our pipelined APSP algorithm (Algorithm 1).

A key contribution in the deterministic APSP algorithm in [3] is a fast deterministic distributed algorithm for computing a *blocker set*. The performance of the blocker set algorithm in [3] does not suffice for our faster APSP algorithms (Theorems I.2 and I.3). In Section III we present a faster blocker set algorithm, which is also a simplification of the blocker set algorithm in [3]. The improved bound that we obtain here for computing a blocker set will not improve the overall bound in [3], but our method could be used there to achieve the same bound with a more streamlined algorithm.

4. Approximate APSP for Non-negative Edge Weights. In Section IV we present an algorithm that matches the earlier bound for computing approximate APSP in graphs with *positive* integer edge weights [16], [18] by obtaining the same bound for non-negative edge weights.

Theorem I.5. *Let $G=(V,E)$ be a directed or undirected edge-weighted graph, where all edge weights are non-negative integers polynomially bounded in n, and where zero-weight edges are allowed. Then, for any $\epsilon>0$ we can compute $(1+\epsilon)$-approximate APSP in $O((n/\epsilon^2)\cdot\log n)$ rounds deterministically in the* CONGEST *model.*

5. Randomized APSP for Arbitrary Edge-Weights. We present a randomized APSP algorithm for directed graphs with arbitrary edge-weights that runs in $\tilde{O}(n^{4/3})$ rounds, w.h.p. in n. No nontrivial sub-$n^{3/2}$ round algorithm was known prior to this result.

Theorem I.6. *Let $G=(V,E)$ be a directed or undirected edge-weighted graph with arbitrary edge weights. Then, we can compute weighted APSP in G in the* CONGEST *model in $\tilde{O}(n^{4/3})$ rounds, w.h.p. in n.*

The corresponding bound for k-SSP is $\tilde{O}(n+n^{2/3}k^{2/3})$. This result improves the prior $\tilde{O}(n^{3/2})$-round (deterministic) bound in [3] but it has been subsumed by a very recent result in [5] that gives an $\tilde{O}(n)$ rounds randomized algorithm for weighted APSP, so we do not describe our algorithm here. See the full paper [2] for a description of this algorithm.

B. Congest Model

In the CONGEST model, there are n independent processors interconnected in a network by bounded-bandwidth links. We refer to these processors as nodes and the links as edges. This network is modeled by graph $G=(V,E)$ where V refers to the set of processors and E refers to the set of links between the processors. Here $|V|=n$ and $|E|=m$.

Each node is assigned a unique ID between 1 and $poly(n)$ and has infinite computational power. Each node has limited topological knowledge and only knows about its incident edges. For the weighted APSP problem we consider, each edge has a positive or zero integer weight that can be represented with $B=O(\log n)$ bits. Also if the edges are directed, the

TABLE I: Table comparing our new results for non-negative edge-weighted graphs (including zero edge weights) with previous known results. Here W is the maximum edge weight and Δ is the maximum weight of a shortest path in G.

PROBLEM: EXACT WEIGHTED APSP					
Author	Arbitrary/ Integer weights	handle zero weights	Randomized/ Deterministic	Undirected/ (Directed & Undirected)	Round Complexity
Huang et al. [13]	Integer	Yes	Randomized	Directed & Undirected	$\tilde{O}(n^{5/4})$
Elkin [8]	Arbitrary	Yes	Randomized	Undirected	$\tilde{O}(n^{5/3})$
Bernstein & Nanongkai [5]	Arbitrary	Yes	Randomized	Directed & Undirected	$\tilde{O}(n)$ **(very recent)**
Agarwal et al. [3]	Arbitrary	Yes	Deterministic	Directed & Undirected	$\tilde{O}(n^{3/2})$
This paper	**Integer**	**Yes**	**Deterministic**	**Directed & Undirected**	$\mathbf{\tilde{O}(n^{3/2-\epsilon/4})}$ **(when $\mathbf{W \leq n^{1-\epsilon}}$)** $\mathbf{\tilde{O}(n^{3/2-\epsilon/3})}$ **(when $\mathbf{\Delta \leq n^{3/2-\epsilon}}$)**
	Arbitrary	**Yes**	**Randomized**	**Directed & Undirected**	$\mathbf{\tilde{O}(n^{4/3})}$

PROBLEM: $(1+\epsilon)$-APPROXIMATION WEIGHTED APSP					
Nanongkai [18]	Integer	No	Randomized	Directed & Undirected	$\tilde{O}(n/\epsilon^2)$
Lenzen & Patt-Shamir [16]	Integer	No	Deterministic	Directed & Undirected	$\tilde{O}(n/\epsilon^2)$
This paper	**Integer**	**Yes**	**Deterministic**	**Directed & Undirected**	$\mathbf{\tilde{O}(n/\epsilon^2)}$

corresponding communication channels are bidirectional and hence the communication network can be represented by the underlying undirected graph U_G of G (as in [11]–[13]). It turns out that our basic pipelined algorithm does not need this feature though our faster algorithm does.

The computation proceeds in rounds. In each round each processor can send a message of size $O(\log n)$ along edges incident to it, and it receives the messages sent to it in the previous round. The model allows a node to send different message along different edges though we do not need this feature in our algorithm. The performance of an algorithm in the CONGEST model is measured by its round complexity, which is the worst-case number of rounds of distributed communication. As noted earlier, for shortest path problems, each node in the network needs to compute its shortest path distance from each source as well as the last edge on such a shortest path.

C. Related Work

Weighted APSP. The current best bound for the weighted APSP problem is in the randomized algorithm of Huang et al. [13] that runs in $\tilde{O}(n^{5/4})$ rounds. This algorithm works for graphs with polynomially bounded integer edge weights (including zero-weight edges), and the result holds with w.h.p. in n. For graphs with arbitrary edge weights, the recent result of Agarwal et al. [3] gives a deterministic APSP algorithm that runs in $\tilde{O}(n^{3/2})$ rounds. This is the current best bound (both deterministic and randomized) for graphs with arbitrary edge weights as well as the best deterministic bound for graphs with integer edge weights. Note that in a very recent result the randomized complexity of the weighted APSP problem with arbitrary edge weights has been improved to $\tilde{O}(n)$. This is very close to the $\Omega(n)$ lower bound, which holds even for unweighted APSP [6], but it is not deterministic.

In this paper we present a deterministic algorithm for non-negative integer edge-weights (including zero-weighted edges) that runs in $\tilde{O}(n\triangle^{1/3})$ rounds when the shortest path distances are at most $\triangle$ and in $\tilde{O}(n^{5/4}W^{1/4})$ rounds when the edge weights are bounded by W. This result improves on the $\tilde{O}(n^{3/2})$ deterministic APSP bound of Agarwal et al. [3] when either edge weights are at most $n^{1-\epsilon}$ or shortest path distances are at most $n^{3/2-\epsilon}$, for any $\epsilon > 0$. We also give a simple randomized algorithm for APSP in graphs with arbitrary edge weights that runs in $\tilde{O}(n^{4/3})$ rounds, w.h.p. in n.

Weighted k-SSP. The current best bound for the weighted k-SSP problem is due to the Huang et al's [13] randomized algorithm that runs in $\tilde{O}(n^{3/4}{\cdot}k^{1/2}{+}n)$ rounds. This algorithm is also randomized and only works for graphs with integer edge weights. The recent deterministic APSP algorithm in [3] can be shown to give an $O(n \cdot \sqrt{k \log n})$ round deterministic algorithm for k-SSP. In this paper, we present a deterministic algorithm for positive including zero integer edge-weighted graphs that runs in $\tilde{O}((\triangle{\cdot}n^2{\cdot}k)^{1/3})$ rounds where the shortest path distances are at most $\triangle$ and in $\tilde{O}((Wk)^{1/4}n)$ rounds when the edge weights are bounded by W.

$(1+\epsilon)$-Approximation Algorithms. For graphs with positive integer edge weights, deterministic $\tilde{O}(n/\epsilon^2)$-round algorithms for a $(1{+}\epsilon)$-approximation to APSP are known [16], [18]. But these algorithms do not handle zero weight edges. In this paper we present a deterministic algorithm that handles zero-weight edges and matches the $\tilde{O}(n/\epsilon^2)$-round bound for approximate APSP known before for positive edge weights.

Roadmap. Section II describes our pipelined algorithm for the (h, k)-SSP problem, and also gives simplifed short-range [13] algorithms based on it in Section II-C. In Section III we present our faster k-SSP algorithm which builds on our pipelined algorithm in Section II and an improved algorithm for computing a blocker set. In Section IV we present our algorithm for approximate APSP and we end with the conclusion in Section V.

II. THE PIPELINED APSP ALGORITHM

We present a pipelined distributed algorithm to compute weighted APSP for shortest path distances at most Δ. The starting point for our algorithm is the distributed algorithm

for *unweighted* APSP in [12], which is a streamlined variant of an earlier APSP algorithm [17]. This unweighted APSP algorithm is very simple: each source initiates its distributed BFS in round 1. Each node v retains the best (i.e., shortest) distance estimate it has received for each source, and stores these estimates in sorted order (breaking ties by source id). Let $d(s)$ (or $d_v(s)$) denote the shortest distance estimate for source s at v and let $pos(s)$ be its position in sorted order ($pos(s) \geq 1$). In a general round r, node v sends out a shortest distance estimate $d(s)$ if $r = d(s) + pos(s)$. Since $d(s)$ is nondecreasing and $pos(s)$ is increasing, there will be at most one $d(s)$ at v that can satisfy this condition. It is shown in [12] that shortest distances for all sources arrive at v in at most $2n$ rounds under this schedule and only one message is sent out by v for each source. The key to the $2n$-round bound is that if the current best distance estimate $d(s)$ for a source s reaches v in round r then $r < d(s) + pos(s)$. Since $d(s) < n$ for any source s and $pos(s)$ is at most n, shortest path values for all sources arrive at any given node v in less than $2n$ rounds.

For our weighted case, since $d(s)$ is at most $\triangle$ for all s, v, it appears plausible that the above pipelining method would apply here as well. Unfortunately, this does not hold since we allow zero weight edges in the graph. The key to the guarantee that a $d(s)$ value arrives at v before round $d(s) + pos(s)$ in the unweighted case in [12] is that the predecessor y that sent its $d_y(s)$ value to v must have had $d_y(s) = d_v(s) - 1$. (Recall that in the unweighted case, $d_y(s)$ is simply the hop-length of the path taken from s to y.) If we have zero-weight edges this guarantee no longer holds for the weighted path length, and it appears that the key property of the unweighted pipelining methodogy no longer applies. Since edge weights larger than 1 are also possible (as long as no shortest path distance exceeds $\triangle$), the hop length of a path can be either greater than or less than its weighted distance.

A. Our (h, k)-SSP algorithm

Algorithm 1 is our pipelined algorithm for a directed graph $G = (V, E)$ with non-negative edge-weights. The input is G, together with the subset S of k vertices for which we need to compute h-hop SSPs. An innovative feature of this algorithm is that the key κ it uses for a path is not its weighted distance, but a function of *both* its hop length l and its weighted distance d. More specifically, $\kappa = d \cdot \gamma + l$, where $\gamma = \sqrt{kh/\Delta}$. This allows the key to inherit some of the properties from the algorithm in [12] through the fact that the hop length is part of κ's value, while also retaining the weighted distance which is the actual value that needs to be computed.

The new key κ by itself is not sufficient to adapt the algorithm for unweighted APSP in [12] to the weighted case. In fact, the use of κ can complicate the computation since one can have two paths from s to v, with weighted distances $d_1 < d_2$, and yet for the associated keys one could have $\kappa_1 > \kappa_2$ (because the path with the smaller weight can have a larger hop-length). Our algorithm handles this with another unusual feature: it may maintain several (though not all) of the key values it receives, and may also send out several key values, even some that it knows cannot correspond to a shortest distance. These features are incorporated into a carefully tailored algorithm that terminates in $O(\sqrt{\triangle kh})$ rounds with all h-hop shortest path distances from the k sources computed.

It is not difficult to show that eventually every shortest path distance key arrives at v for each source from which v is reachable when Algorithm 1 is executed. In order to establish the bound on the number of rounds, we show that our pipelined algorithm maintains two important invariants:

> **Invariant 1:** If an entry Z is added to $list_v$ in round r, then $r < \lceil Z.\kappa + pos(Z) \rceil$, where $Z.\kappa$ is $Z's$ key value.
> **Invariant 2:** The number of entries for a given source s at $list_v$ is at most $\sqrt{\triangle h/k} + 1$.

Invariant 1 is the natural generalization of the unweighted algorithms [12], [17] for the key κ that we use. On the other hand, to the best of our knowledge, Invariant 2 has not been used before, nor has the notion of storing multiple paths or entries for the same source at a given node. By Invariant 2, the number of entries in any list is at most $\sqrt{\triangle kh} + k$, so $pos(Z) \leq \sqrt{\triangle kh} + k$ for every list at every round. Since the value of any κ is at most $\triangle \cdot \gamma + h$, by Invariant 1 every entry is received by round $2\sqrt{\triangle kh} + k + h$.

We now give the details of Algorithm 1 starting with a step-by-step description followed by its analysis. Recall that the key value we use for a path π is $\kappa = d \cdot \gamma + l$, where $\gamma = \sqrt{kh/\Delta}$, d is the weighted path length, and l is the hop-length of π. At each node v our algorithm maintains a list, $list_v$, of the entries and associated data it has retained. Each element Z on $list_v$ is of the form $Z = (\kappa, d, l, x)$, where x is the source vertex for the path corresponding to κ, d, and l. The elements on $list_v$ are ordered by key value κ, with ties first resolved by the value of d, and then by the label of the source vertex. We use $Z.\nu$ to denote the number of keys for source x stored on $list_v$ at or below Z. The position of an element Z in $list_v$ is given by $pos(Z)$, which gives the number of elements at or below Z on $list_v$. If the vertex v and the round r are relevant to the discussion we will use the notation $pos_v^r(Z)$, but we will remove either the subscript or the superscript (or both) if they are clear from the context. We also have a flag $Z.\mathit{flag}\text{-}d^*$ which is set if Z has the smallest (d, κ) value among all entries for source x (so d is the shortest weighted distance from s to v among all keys for x on $list_v$). A summary of our notation is in Table II.

Initially, when round $r = 0$, $list_v$ is empty unless v is in the source set S. Each source vertex $x \in S$ places an element $(0, 0, 0, x)$ on its $list_x$ to indicate a path of weight 0 and hop length 0 from x to x, and $Z.\mathit{flag}\text{-}d^*$ is set to $true$. In Step 1 of the Initialization round 0, node v initializes the distance from every source to ∞. In Step 2 every source vertex initializes the distance from itself to 0 and adds the corresponding entry in its list. There are no Sends in round 0.

In a general round r, in Step 1 of Algorithm 1, v checks if $list_v$ contains an entry Z with $\lceil Z.\kappa + pos_v(Z) \rceil = r$. If there is such an entry Z then v sends Z to its neighbors, along with $Z.\nu$ and $Z.flag\text{-}d^*$ in Step 2. Steps 3-13 describe the steps

TABLE II: Notations

Global Parameters:	
S	set of sources
k	number of sources, or $\|S\|$
h	maximum number of hops in a shortest path
$\triangle$	maximum weighted distance of a shortest path
n	number of nodes
γ	parameter equal to $\sqrt{hk/\Delta}$
Local Variables at node v:	
d_x^*	current shortest path distance from x to v; same as $d_{x,v}^*$
$list_v$	list at v for storing the SP and non-SP entries

Variables/Parameters for entry $Z = (\kappa, d, l, x)$ in $list_v$:	
κ	key for Z; $\kappa = d \cdot \gamma + h$
d	weight (distance) of the path associated with this entry
l	hop-length of the path associated with this entry
x	start node (i.e. source) of the path associated with this entry
p	parent node of v on the path associated with this entry
ν	number of entries for source x at or below Z in $list_v$ (not stored explicitly)
flag-d*	flag to indicate if Z is the current SP entry for source x
pos	position of Z in $list_v$ in a round r; same as pos^r, pos_v^r
SP	shortest path

taken at v after receiving a set of incoming messages I from its neighbors. In Step 7 an entry Z is created from an incoming message M, updated to reflect the d and l values at v. Step 9 checks if Z has a shorter distance than the current shortest path entry, Z^*, at v, or a shorter hop-length (if the distance is the same), or a parent with smaller ID (if both distance and hop-length are same). And if so, then Z is marked as SP in Step 10 and is then inserted in $list_v$ in Step 11. Otherwise, if Z is a non-SP it is inserted into $list_v$ in Step 13 only if the number of entries on $list_v$ for source x with key $< Z.\kappa$ in $list_v$ is less than $Z^-.\nu$. This is the rule that decides if a received entry that is not the SP entry is inserted into $list_v$.

INITIALIZATION: Initialization procedure for Algorithm 1 at node v

Input: set of sources S

1: **for each** $x \in S$ **do** $d_x^* \leftarrow \infty$
2: **if** $v \in S$ **then** $d_v^* \leftarrow 0$; add an entry $Z = (0, 0, 0, v)$ to $list_v$; Z.*flag-d*$^* \leftarrow true$

Algorithm 1 Pipelined (h, k)-SSP algorithm at node v for round r

Input: A set of sources S

1: **send [Steps 1-2]: if** there is an entry Z with $\lceil Z.\kappa + pos_v^r(Z)\rceil = r$
2:     **then** compute $Z.\nu$ and form the message $M = \langle Z, Z.\text{flag-}d^*, Z.\nu\rangle$ and send M to all neighbors
3: **receive [Steps 3-13]:** let I be the set of incoming messages
4: **for each** $M \in I$ **do**
5:     let $M = (Z^- = (\kappa^-, d^-, l^-, x), Z^-.\text{flag-}d^*, Z^-.\nu)$ and let the sender be y.
6:     $\kappa \leftarrow \kappa^- + w(y, v) \cdot \gamma + 1$; $d \leftarrow d^- + w(y, v)$; $l \leftarrow l^- + 1$
7:     $Z \leftarrow (\kappa, d, l, x)$; $Z.\text{flag-}d^* \leftarrow false$; $Z.p \leftarrow y$ (Z may be added to $list_v$ in Step 11 or 13)
8:     let Z^* be the entry for x in $list_v$ such that $Z^*.\text{flag-}d^* = true$, if such an entry exists (otherwise $d_x^* = \infty$)
9:     **if** $Z^-.\text{flag-}d^* = true$ and $l \leq h$ and $((d < d_x^*)$ or $(d = d_x^*$ and $Z.\kappa < Z^*.\kappa)$ or $(d = d_x^*$ and $Z.\kappa = Z^*.\kappa$ and $Z.p < Z^*.p))$ **then**
10:         $d_x^* \leftarrow d$; $Z.\text{flag-}d^* \leftarrow true$; $Z^*.\text{flag-}d^* \leftarrow false$ (if Z^* exists)
11:         INSERT(Z)
12:     **else**
13:         **if** there are less than $Z^-.\nu$ entries for x with $key \leq Z.\kappa$ **then** INSERT(Z)

INSERT(Z): Procedure for adding Z to $list_v$

1: insert Z in $list_v$ in sorted order of (κ, d, x)
2: **if** $\exists$ an entry Z' for x in $list_v$ such that $Z'.\text{flag-}d^* = false$ and $pos(Z') > pos(Z)$ **then**
3:     find Z' with smallest $pos(Z')$ such that $pos(Z') > pos(Z)$ and $Z'.\text{flag-}d^* = false$
4:     remove Z' from $list_v$

Steps 1-4 of procedure INSERT perform the addition of a new entry Z to $list_v$. In Step 1 Z is inserted in $list_v$ in the sorted order of (κ, d, x). The algorithm then moves on to remove an existing entry for source x on $list_v$ if the condition in Step 2 holds. This condition checks if there is a non-SP entry above Z in $list_v$. If so then the closest non-SP entry above Z is removed in Steps 3-4.

Algorithm 1 performs these steps in successive rounds. We next analyze it for correctness and we also show that it terminates with all shortest distances computed before round $r = \lceil 2\sqrt{\triangle kh} + k + h\rceil$.

B. Correctness of Algorithm 1

We now provide a sketch for correctness of Alg. 1. The complete proofs are in the full paper [1]. The initial Observations and Lemmas given below establish useful properties of an entry Z in a $list_v$ and of $pos_v^r(Z)$ and its relation to $pos_y^r(Z^-)$. We then present the key lemmas. In Lemma II.9, we show that the collection of entries for a given source x in $list_v$ can be mapped into (d, l) pairs with non-negative l values such that $d = d^*$ for the shortest path entry, and the d values for all other entries are distinct and larger than d^*. (It turns out that we cannot simply use the d values already present in Z's entries for this mapping since we could have two different entries for source x on $list_v$, Z_1 and Z_2, that have the same d value.) Once we have Lemma II.9 we are able to bound the number of entries for a given source at $list_v$ by $\frac{h}{\gamma} + 1$ in Lemma II.11, and this establishes Invariant 2 (which is stated in Sec II-A). Lemma II.12 establishes Invariant 1. In Lemma II.13 we establish that all shortest path values reach node v. With these results in hand, the final Lemma II.14 for the round bound for computing (h, k)-SSP with shortest path distances at most $\triangle$ is readily established, which then gives Theorem I.1.

Observation II.1. *Let Z be an entry for a source $x \in S$ added to $list_v$ in round r. Then if Z is removed from $list_v$ in a round $r' \geq r$, it was replaced by another entry for x, Z', such that $pos_v^{r'}(Z) > pos_v^{r'}(Z')$ and $Z.\kappa \geq Z'.\kappa$.*

Lemma II.2. *Let Z be an entry in $list_v$. Then $pos_v^{r'}(Z) \geq pos_v^r(Z)$ for all rounds $r' > r$, for which Z exists in v's list.*

Observation II.3. *Let Z be an entry for source x that was added to $list_v$. If there exists a non-SP entry for x above Z in $list_v$, then the closest non-SP entry above Z will be removed.*

Observation II.4. *Let Z^- be an entry for source x sent from y to v in round r, and let Z be the corresponding entry created for possible addition to $list_v$ in Step 7 of Algorithm 1. If Z is not added to $list_v$, then there is an entry $Z' \neq Z$ for source x in $list_v$ with Z'.flag-d*$= true$, and there are at least $Z^-.\nu$ entries for x with $key \leq Z.\kappa$ at the end of round r.*

Lemma II.5. *Let Z be an entry for source x that is present on $list_v$ in round r. Let $r' > r$, and let c and c' be the number of entries for source x on $list_v$ that have key value less than Z's key value in rounds r and r' respectively. Then $c' \geq c$.*

Lemma II.5 holds for every round greater than r, even if Z is removed from $list_v$.

Lemma II.6. *Let Z^- be an entry for source x sent from y to v and suppose the corresponding entry Z (Step 7 of Algorithm 1) is added to $list_v$ in round r. Then there are at least $Z^-.\nu$ entries at or below Z in $list_v$ for source x.*

Proof. Assume inductively that this result holds for all entries on $list_v$ and $list_y$ with key value at most $Z.\kappa$ at all previous rounds and at y in round r as well. (It trivially holds initially.)

Let Z_1^- be the $(Z^-.\nu - 1)$-th entry for source x in $list_y$. Since Z_1^- has a key value smaller than Z^- it was sent to v in an earlier round r'. If the corresponding entry Z_1 created for possible addition to $list_v$ in Step 7 of Algorithm 1, was inserted in $list_v$ then by inductive assumption there were at least $Z_1^-.\nu = Z^-.\nu - 1$ entries for x at or below Z_1 in $list_v$. And by Lemma II.5 this holds for round r as well and hence the result follows since Z is present above Z_1 in $list_v$.

And if Z_1 was not added to $list_v$ in round r', then by Observation II.4 there were already $Z^-.\nu - 1$ entries for x with key $\leq Z_1.\kappa$ and by Lemma II.5 there are at least $Z^-.\nu - 1$ entries for x with key $\leq Z_1.\kappa \leq Z.\kappa$ on $list_v$ at round r and hence the result follows. □

Lemma II.7. *Let Z^- be an entry sent from y to v in round r and let Z be the corresponding entry created for possible addition to $list_v$ in Step 7 of Algorithm 1. For each source $x_i \in S$, let there be exactly c_i entries for x_i at or below Z^- in $list_y$. If Z is added to $list_v$, then for each $x_i \in S$, there are at least c_i entries for x_i at or below Z in $list_v$.*

Corollary II.8. *Let Z^- be an entry sent from y to v in round r and let Z be the corresponding entry created for possible addition to $list_v$ in Step 7 of Algorithm 1. If Z is added to $list_v$, then $pos^r_y(Z^-) \leq pos^r_v(Z)$.*

Lemma II.9. *Let $\mathcal{C}$ be the entries for a source $x \in S$ in $list_v$ in round r. Then the entries in $\mathcal{C}$ can be mapped to (d, l) pairs such that each $l \geq 0$ and each $Z \in \mathcal{C}$ is mapped to a distinct d value with $Z.\kappa = d \cdot \gamma + l$. Also $d = d^*_x$ if Z is a current shortest path entry, otherwise $d > d^*_x$.*

Proof. (Sketch.) By induction on j, the number of entries in $\mathcal{C}$. When $j = 1$, we can map d and l to the pair in the single entry Z. Assume true till $j - 1$, and consider the first time $|\mathcal{C}|$ becomes j at $list_v$, and let this occur when node y sends Z^- to v and this is updated and inserted as Z in $list_v$ in round r.

If Z is inserted as a new shortest path entry with distance value d^*, then we can again map d and l to the pair in Z, since d^* is smaller than all other d values for x at v.

If Z is inserted as a non-SP entry and its d value has already been assigned to one of the $j-1$ entries for source x on $list_v$, consider the entries for source x with key value at most $Z^-.\kappa$ in $list_y$. Step 13 of Algorithm 1 ensures that there are j such values. Inductively these j entries have j distinct d^- values assigned to them, and we transform these into j distinct values for $list_v$ by adding $w(y, v) \cdot \gamma + 1$ to each of them. At least one of these j values, say d', differs from the $j - 1$ mapped d values at v, and it is also readily seen that the associated l value for d' must be greater than 0. So we can map Z to (d', l) (for more details, see [1]).

The general case when the number of entries remains at j both before and after the insert can be handled similarly. □

Lemma II.10. *Let Z be the current shortest path distance entry for a source $x \in S$ in v's list. Then the number of entries for x below Z in $list_v$ is at most h/γ.*

Proof. By Lemma II.9, we know that the keys of all the entries for x can be mapped to (d, l) pairs such that each entry is mapped to a distinct d value and $l > 0$.

We have $Z.\kappa = d^*_x \cdot \gamma + l^*_x$, where l^*_x is the hop-length of the shortest path from x to v. Let Z'' be an entry for x below Z in v's list. Then, $Z''.\kappa \leq Z.\kappa$. It implies that $d'' \cdot \gamma \leq d^*_x \cdot \gamma + (l^*_x - l'') < d^*_x \cdot \gamma + h$ which gives $d'' < d^*_x + h/\gamma$. Since $d'' \geq d^*_x$, there can be at most h/γ entries for x below Z in $list_v$. □

Lemma II.11. *For each source $x \in S$, v's list has at most $h/\gamma + 1$ entries for x.*

In Lemmas II.12-II.13 we establish an upper bound on the round r by which v receives a shortest path entry Z^*.

Lemma II.12. *If an entry Z is added to $list_v$ in round r then $r < \lceil Z.\kappa + pos^r_v(Z) \rceil$.*

Proof. The lemma holds in the first round since all entries have non-negative κ, any received entry has hop length at least 1, and the lowest position is 1 so for any entry Z received by v in round 1, $\lceil Z.\kappa + pos^1_v(Z) \rceil \geq 1 + 1 > 1$.

Let r be the first round (if any) in which the lemma is violated, and let it occur when entry Z is added to $list_v$. So $r \geq \lceil Z.\kappa + pos^r_v(Z) \rceil$. Let $r_1 = \lceil Z.\kappa + pos^r_v(Z) \rceil$ (so $r_1 \leq r$ by assumption).

Since Z was added to $list_v$ in round r, Z^- was sent to v by a node y in round r. So by Step 1 $r = \lceil Z^-.\kappa + pos^r_y(Z^-) \rceil$. But $Z.\kappa > Z^-.\kappa$ and $pos^r_v(Z) \geq pos^r_y(Z^-)$, hence r must be less than $\lceil Z.\kappa + pos^r_v(Z) \rceil$. □

Lemma II.13. *Let $\pi^*_{x,v}$ be a shortest path from source x to v with the minimum number of hops among h-hop shortest paths from x to v. Let $\pi^*_{x,v}$ have l^* hops and shortest path distance $d^*_{x,v}$. Then v receives an entry $Z^* = (\kappa, d^*_{x,v}, l^*, x)$ by round $r < \lceil Z^*.\kappa + pos^r_v(Z^*) \rceil$.*

Proof. If an entry $Z^* = (\kappa, d^*_{x,v}, l^*, x)$ is placed on $list_v$ by v then by Lemma II.12 it is received before round $\lceil Z^*.\kappa + pos^r_v(Z^*) \rceil$ and hence it will be sent in round $r = \lceil Z^*.\kappa + pos^r_v(Z^*) \rceil$ in Step 2. It remains to show that an entry for path $\pi^*_{x,v}$ is received by v. We establish this for all pairs x, v by induction on key value κ. (See [1] for the full proof.) □

In Lemma II.14 we establish an upper bound on the round r by which Algorithm 1 terminates.

Lemma II.14. *Let Δ be the maximum shortest path distance in the h-hop paths. Algorithm 1 correctly computes the h-hop shortest path distances from each source $x \in S$ to each node $v \in V$ by round $\lceil \triangle\gamma + h + \triangle \cdot \gamma + k \rceil$.*

Proof. An h-hop shortest path has weight at most $\triangle$, hence a key corresponding to a shortest path entry will have value

at most $\triangle\gamma + h$. Thus by Lemma II.13, for every source $x \in S$ every node $v \in V$ should have received the shortest path distance entry, Z^*, for source x by round $r = \lceil \triangle\gamma + h + pos_v^r(Z^*) \rceil$.

Now we need to bound the value of $pos_v^r(Z^*)$. By Lemma II.11, we know that there are at most $h/\gamma + 1$ entries for each source $x \in S$ in a node v's list. Now as there are k sources, v's list has at most $(h/\gamma + 1) \cdot k \leq \gamma \cdot \triangle + k$ entries, thus $pos_v^r(Z^*) \leq \gamma \cdot \triangle + k$ and hence $r \leq \lceil \Delta\gamma + h + \gamma \cdot \Delta + k \rceil$. □

Since $\gamma = \sqrt{hk/\Delta}$, Lemma II.14 establishes Theorem I.1.

C. *Simplified Versions of Short-Range Algorithms in [13]*

We describe here simplified versions of the *short-range* and *short-range-extension* algorithms used in the randomized $\tilde{O}(n^{5/4})$ round APSP algorithm in Huang et al. [13]. Our short-range Algorithm 2 is implicit in our pipelined APSP algorithm (Algorithm 1) and is much simpler than it since it is for a single source.

Given a hop-length h and a source vertex x, the short-range algorithm in [13] computes the h-hop shortest path distances from source x in a graph G' (obtained through 'scaling') where $\Delta \leq n - 1$. The scaled graph has different edge weights for different sources, and hence h-hop APSP is computed through n h-hop SSSP (or *short-range*) computations, each of which runs with *dilation* (i.e., number of rounds) $\tilde{O}(n\sqrt{h})$ and *congestion* (i.e., maximum number of messages along an edge) $O(\sqrt{h})$. By running this algorithm using each vertex as source, h-hop APSP is computed in G' in $O(n\sqrt{h})$ rounds w.h.p. in n using a scheduling result in Ghaffari's framework [10], which gives a randomized method to execute this collection of different short-range executions simultaneously in $\tilde{O}(\text{dilation} + n \cdot \text{congestion}) = \tilde{O}(n\sqrt{h})$ rounds.

The short-range algorithm in [13] for a given source runs in two stages:. Initially every zero edge-weight is increased to a positive value $\alpha = 1/\sqrt{h}$ and then h-hop SSSP is computed using a BFS variant in $\tilde{O}(n/\alpha) = \tilde{O}(n\sqrt{h})$ rounds. This gives an approximation to the h-hop SSSP where the additive error is at most $h\alpha = \sqrt{h}$. This error is then fixed by running the Bellman-Ford algorithm [4] for h rounds. The total round complexity of this SSSP algorithm is $\tilde{O}(n\sqrt{h})$ and the congestion is $O(\sqrt{h})$.

Algorithm 2 Round r of short-range algorithm for source x (initially $d^* \leftarrow 0$; $l^* \leftarrow 0$ at source x)

(at each node $v \in V$**)**
1: **send: if** $\lceil d^* \cdot \sqrt{h} + l^* \rceil = r$ **then** send (d^*, l^*) to all the neighbors
2: **receive [Steps 2-6]:** let I be the set of incoming messages
3: **for each** $M \in I$ **do**
4: let $M = (d^-, l^-)$ and let the sender be y.
5: $d \leftarrow d^- + w(y, v)$; $l \leftarrow l^- + 1$
6: **if** $d < d^*$ or ($d = d^*$ and $l < l^*$) **then** set $d^* \leftarrow d$; $l^* \leftarrow l$

We now present a simplified short-range algorithm (Algorithm 2) with the same dilation $O(n\sqrt{h})$ and congestion $O(\sqrt{h})$. Here d^* is the current best estimate for the shortest path distance from x at node v and l^* is the hop-length of the corresponding path. Source node x initializes d^* and l^* values to zero and sends these values to its neighbors in round 0 (Step 1). At the start of a round r, each node v checks if its current d^* and l^* values satisfy $\lceil d^* \cdot \sqrt{h} + l^* \rceil = r$, and if so, it sends this estimate to each of its neighbors. To bound the number of such messages v sends throughout the entire execution, we note that v will send another message in a future round only if it receives a smaller d^* value with higher $\lceil d^* \cdot \sqrt{h} + l^* \rceil$ value. But since $l^* \leq h$ and d^* values are non-negative integers, v can send at most $\sqrt{h}$ messages to its neighbors throughout the entire execution. A proof similar to [12] (a simplified version of Lemma II.12) shows that as long as edge-weights are non-negative, v will always receive the message that creates the pair d^*, l^* at v before round $\lceil d^* \cdot \sqrt{h} + l^* \rceil$.

If shortest path distances are bounded by Δ, Algorithm 2 runs in $\lceil \Delta \cdot \sqrt{h} + h \rceil$ rounds with congestion at most $\sqrt{h}$. And if $\Delta \leq n - 1$ (as in [13]), then we can compute shortest path distances from x to every node v in $O(n\sqrt{h})$ rounds.

We can similarly simplify the short-range-extension algorithm in [13], where some nodes already know their distance from source x and the goal is to compute shortest paths from x by extending these already computed shortest paths to u by another h hops. To implement this, we only need to modify the initialization in Algorithm 2 so that each such node u initializes d^* with this already computed distance. The round complexity is again $O(\Delta\sqrt{h})$ and the congestion per source is $O(\sqrt{h})$. This gives us the following result.

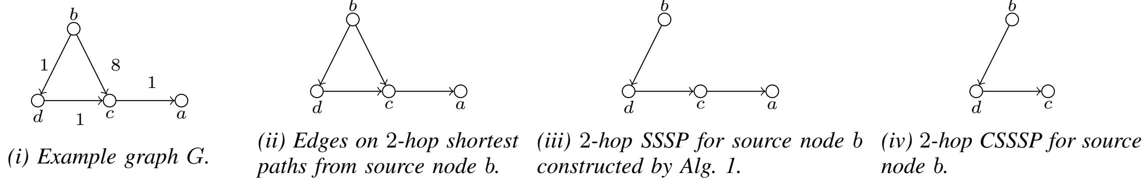
Lemma II.15. *Let $G = (V, E)$ be a directed or undirected graph, where all edge weights are non-negative distances (and zero-weight edges are allowed), and where shortest path distances are bounded by Δ. Then by using Algorithm 2, we can compute h-hop SSSP and h-hop extension in $O(\Delta\sqrt{h})$ rounds with congestion bounded by $\sqrt{h}$.*

As in [13] we can now combine our Algorithm 2 with Ghaffari's randomized framework [10] to compute h-hop APSP and h-hop extensions (for all source nodes) in $\tilde{O}(\Delta\sqrt{h} + n\sqrt{h})$ rounds w.h.p. in n. The result can be readily modified to include the number of sources, k, by sending the current estimates (d^*, l^*) in round $\lceil d^* \cdot \gamma + l^* \rceil$, where $\gamma = \sqrt{hk/\Delta}$ as in Algorithm 1 (instead of $\lceil d^* \cdot \sqrt{h} + l^* \rceil$), and the resulting algorithm runs in $O(\sqrt{\Delta hk})$ rounds with congestion bounded by $\sqrt{\Delta h/k}$. Then we can compute h-hop k-SSP and h-hop extensions for all k sources in $\tilde{O}(\sqrt{\Delta hk})$ rounds.

III. Faster k-SSP Algorithm using blocker set

In this section we give faster APSP and k-SSP algorithms. The overall Alg. 3 has the same structure as the recent deterministic $O(n^{3/2} \cdot \sqrt{\log n})$ round weighted APSP algorithm in [3] but we use a variant of Alg. 1 in place of Bellman-Ford, and we also present new methods within two of the steps.

In our improved Alg. 3, Steps 3-5 are unchanged from the algorithm in [3]. However we give an alternate method for Step 1, which computes h-hop CSSSP (see Sec. III-A), since the method in [3] takes $\Theta(n \cdot h)$ rounds, which is too large for our purposes. Our new method is very simple and uses Alg. 1 and runs in $O(\sqrt{\triangle hk})$ rounds. (An implementation



(i) Example graph G. *(ii) Edges on 2-hop shortest paths from source node b.* *(iii) 2-hop SSSP for source node b constructed by Alg. 1.* *(iv) 2-hop CSSSP for source node b.*

Fig. 1: This figure gives an example graph G where the union of the edges on the 2-hop shortest paths from source node b differs from the 2-hop SSSP constructed by both Bellman-Ford and Alg. 1, and both are different from the 2-hop CSSSP generated for source node b.

using Bellman-Ford [4] would give an $O(n \cdot h)$-round bound, which could be used in [3] to simplify a step in that algorithm.)

Step 2 computes a *blocker set*, defined as follows.

Definition III.1 (Blocker Set [3], [14]). *Let H be a collection of rooted h-hop trees in a graph $G = (V, E)$. A set $Q \subseteq V$ is a* blocker set *for H if every root to leaf path of length h in every tree in H contains a vertex in Q. Each vertex in Q is called a* blocker vertex *for H.*

For Step 2 we use the overall blocker set algorithm from [3], which runs in $O(n \cdot h + (n^2 \log n)/h)$ rounds and computes a blocker set of size $q = O((n \log n)/h)$ for the h-hop trees constructed in Step 1 of algorithm 3. But this gives only an $\tilde{O}(n^{3/2})$ bound for Step 2 (by setting $h = \tilde{O}(\sqrt{n})$), so it will not help us to improve the bound on the number of rounds for APSP beyond Algorithm 1. Instead we modify and improve a key step where that earlier blocker set algorithm has a $\Theta(n \cdot h)$ round preprocessing step. (Our improved method here will not help to improve the bound in [3] but does help to obtain a better bound here in conjunction with Algorithm 1.) We give the details of our method for Step 2 in Section III-B.

Algorithm 3 Overall k-SSP algorithm (adapted from [3])

Input: set of sources S, number of hops h
1: Compute h-hop CSSSP rooted at each source $x \in S$ (described in Section III-A).
2: Compute a blocker set Q of size $\Theta(\frac{n \log n}{h})$ for the h-hop CSSSP computed in Step 1 (described in Section III-B).
3: **for each** $c \in Q$ **in sequence:** compute SSSP tree rooted at c.
4: **for each** $c \in Q$ **in sequence:** broadcast $ID(c)$ and the shortest path distance values $\delta_h(x, c)$ for each $x \in S$.
5: **Local Step at node** $v \in V$**:** for each $x \in S$ compute the shortest path distance $\delta(x, v)$ using the received values.

Lemma III.2. *Algorithm 3 computes k-SSP in $O(\frac{n^2 \log n}{h} + \sqrt{\triangle hk})$ rounds.*

Proof. The correctness of Algorithm 3 is established in [3]. Step 1 runs in $O(\sqrt{\triangle hk})$ rounds by Lemma III.5 in Section III-A. In Section III-B we will give an $O(n \cdot q + \sqrt{\triangle hk})$ rounds algorithm to find a blocker set of size $q = O(\frac{n \log n}{h})$. Simple $O(n \cdot q)$ round algorithms for Steps 3 and 4 are given in [3]. Step 5 has no communication. Hence the overall bound for Algorithm 3 is $O(n \cdot q + \sqrt{\triangle hk})$ rounds. Since $q = O(\frac{n \log n}{h})$ this gives the desired bound. ☐

Proofs of Theorem I.3 and I.2:. Using $h = \frac{n^{4/3} \cdot \log^{2/3} n}{(2k \cdot \triangle)^{1/3}}$ in Lemma III.2 we obtain the bounds in Theorem I.3.

If edge weights are bounded by W, the weight of any h-hop path is at most hW. Hence by Lemma III.2, the k-SSP algorithm (Alg. 3) runs in $O(\frac{n^2 \log n}{h} + h\sqrt{Wk})$ rounds. Setting $h = n \log^{1/2} n/(W^{1/4}k^{1/4})$ we obtain the bounds stated in Theorem I.2. ☐

A. Computing Consistent h-hop trees

Recall that an h-hop shortest path from a source s to a vertex v in G is a path of minimum weight from s to v among all paths with at most h hops. If we consider the graph consisting of an h-hop shortest path from a source s to every vertex in G reachable from s within h hops, it need not form a tree since the prefix of an h-hop shortest path may not itself be an h-hop shortest path. The parent pointers for the h-hop shortest paths computed by our pipelined (h, k)-SSP algorithm (Alg. 1) as well as by using Bellman-Ford [4] suffer from a similar problem: the tree constructed by the parent pointers could have height greater than h (see Fig 1).

Within the algorithm for computing blocker set in Step 2 in Algorithm 3 (see Section III-B for details), there are algorithms for updating the 'scores' of the ancestor and descendant nodes of a newly chosen blocker node in the collection of trees that contain h-hop shortest paths. The efficient methods used in these algorithms are based on having a consistent set of paths across all trees in the collection. In order to create a consistent collection of paths across all sources, we introduce the following definition of an *h-hop Consistent SSSP (CSSSP)* collection. This notion is implicit in [3] but is not explicitly defined there.

Definition III.3 (CSSSP). *Let H be a collection of rooted trees of height h in a graph $G = (V, E)$. Then H is an h-hop* CSSSP collection *(or simply an h-hop* CSSSP*) if for every $u, v \in V$ the path from u to v is the same in each of the trees in H (in which such a path exists), and is the h-hop shortest path from u to v in the h-hop tree T_u rooted at u. Further, each T_u contains every vertex v that has a path with at most h hops from u in G that has distance $\delta(u, v)$.*

Neither Algorithm 1 nor running h iterations of Bellman-Ford is guaranteed to construct a CSSP collection. At the same time, we observe that the trees in an h-hop CSSSP collection may not contain all h-hop shortest paths: In particular, if every shortest path from source s to a vertex x has more than h hops, then the h-hop tree for source s in the CSSSP collection is not required to have x in it. See Fig. 1.

Our method to construct an h-hop CSSSP collection is very simple: We execute Algorithm 1 to construct $2h$-hop SSSPs instead of h-hop SSSPs. Our CSSSP collection will retain the initial h hops of each of these $2h$-hop SSSPs. In other words, each vertex v willl set the parent pointer $p(v)$ to NIL for a source s if the hop-length of the corresponding path is greater than h. We now show that this simple construction results in an h-hop CSSSP collection. Thus we are able to construct h-

hop CSSSPs by incurring just a constant factor overhead in the number of rounds over the bound for Algorithm 1.

Lemma III.4. *Consider running Algorithm 1 using the hop-length bound $2h$. Let $\mathcal{C}$ be the collection of h-hop trees formed by retaining the initial h hops in each of these $2h$-hop SSSPs. Then the collection $\mathcal{C}$ forms an h-hop CSSSP collection.*

Proof. If not, then there exist vertices u, v and trees T_x, T_y such that the paths from u to v in T_x and T_y are different. Let $\pi^x_{u,v}$ and $\pi^y_{u,v}$ be the corresponding paths in these trees.

There are three possible cases: (1) when $wt(\pi^x_{u,v}) \neq wt(\pi^y_{u,v})$ (2) when paths $\pi^x_{u,v}$ and $\pi^y_{u,v}$ have same weight but different hop-lengths (3) when both $\pi^x_{u,v}$ and $\pi^y_{u,v}$ have same weight and hop-length.

(1) $wt(\pi^x_{u,v}) \neq wt(\pi^y_{u,v})$: w.l.o.g. assume that $wt(\pi^x_{u,v}) < wt(\pi^y_{u,v})$. Now if we replace $\pi^y_{u,v}$ in T_y with $\pi^x_{u,v}$, we get a path of smaller weight from y to v of hop-length at most $2h$. But then node v should have picked this lighter path in Step 9 of Algorithm 1 (Step 1 of Algorithm 3 for computing $2h$-hop SSSPs), resulting in a contradiction.

(2) paths $\pi^x_{u,v}$ and $\pi^y_{u,v}$ have same weight but different hop-lengths. W.l.o.g. assume that path $\pi^x_{u,v}$ has smaller hop-length than $\pi^y_{u,v}$. Then $\kappa(\pi^x_{u,v}) < \kappa(\pi^y_{u,v})$ and hence $\kappa(\pi^y_{y,u} \circ \pi^x_{u,v}) < \kappa(\pi^y_{y,u} \circ \pi^y_{u,v})$. And again v would have picked the path $\pi^y_{y,u} \circ \pi^x_{u,v}$ as the shortest path from y in Step 9 of Algorithm 1 where we prefer paths with smaller key even if they have same weighted distance.

(3) both $\pi^x_{u,v}$ and $\pi^y_{u,v}$ have same weight and hop-length. W.l.o.g. assume that these two paths have the smallest hop-length for which the paths differ. Let (a, v) be the last edge on the path $\pi^x_{u,v}$ and let (b, v) be the last edge on the path $\pi^y_{u,v}$. W.l.o.g. assume that $ID(a) < ID(b)$ (a cannot equal b since the resulting smaller hops subpaths $\pi^x_{u,a}$ and $\pi^y_{u,a}$ would be different, which is not possible). Then in Step 9 of Algorithm 1 v would have chosen the path $\pi^y_{y,u} \circ \pi^x_{u,v}$ as the shortest path from y instead of $\pi^y_{y,v}$, where we prefer paths with smaller parent ID even if they have same κ value. □

It is readily seen that a similar construction can be used with the Bellman-Ford a algorithm.

Lemma III.5. *h-hop CSSSPs can be computed in $O(\sqrt{\Delta hk})$ rounds using Algorithm 1 and in $O(nh)$ rounds using the Bellman-Ford algorithm.*

We now show two properties of an h-hop CSSSP collection that we will use in our blocker set algorithm in the next section. (Lemma III.7 is established in [3]). In the following, we call a tree T rooted at a vertex c an out-tree if all the edges incident to c are outgoing edges from c and we call T an in-tree if all the edges incident to c are incoming edges.

Lemma III.6. *Let $\mathcal{C}$ be an h-hop CSSSP collection. Let c be a vertex in G and let T be the union of the edges in the collection of subtrees rooted at c in the trees in $\mathcal{C}$. Then T forms an out-tree rooted at c.*

Proof. If not, there exist nodes u and v and trees T_x and T_y such that the path from c to u in T_x and path from c to v in T_y first diverge from each other after starting from c and then coincide again at some vertex z. But since $\mathcal{C}$ is an h-hop CSSSP collection, by Lemma III.4 the path from c to z in the collection $\mathcal{C}$ is unique. □

Lemma III.7 ([3]). *Let $\mathcal{C}$ be an h-hop CSSSP collection. Let c be a vertex in G and let T be the union of the edges on the tree-path from the root of each tree in $\mathcal{C}$ to c (for the trees that contain c). Then T forms an in-tree rooted at c.*

B. Computing a Blocker Set

Our overall blocker set algorithm runs in $O(\frac{n^2 \log n}{h} + \sqrt{\triangle hk})$ rounds. It differs from the blocker set algorithm in [3] by developing faster algorithms for two steps that take $O(nh)$ rounds in [3].

The first step in [3] that takes $O(nh)$ rounds is the step that computes the initial 'scores' at all nodes for all h-hop trees in the CSSSP collection. The score of node v in an h-hop tree is the number of v's descendants in that tree. Here we compute scores for all trees at all nodes in $O(\sqrt{\triangle hk})$ rounds with a timestamp pipelining technique introduced in [12] for propagating values from descendants to ancestors in the shortest path trees within the same bound as the APSP algorithm.

To explain the second $O(nh)$-round step in [3], we first give a brief recap of the blocker set algorithm in [3]. This algorithm picks nodes to be added to the blocker set greedily. The next node that is added to the blocker set is one that lies in the maximum number of paths in the h-hop trees that have not yet been covered by the already selected blocker nodes. To identify such a node, the algorithm maintains at each node v a count (or *score*) of the number of descendant leaves in each tree, since the sum of these counts is precisely the number of root-to-leaf paths in which v lies. Once all vertices have their overall score, the new blocker node c can be identified as one with the maximum score. It now remains for each node v to update its scores to reflect the fact that paths through c no longer exist in any of the trees. This update computation is divided into two steps in [3]. In both steps, the main challenge is for a given node to determine, in each tree T_x, whether it is an ancestor of c, a descendant of c, or unrelated to c.

1. *Updates at Ancestors.* For each v, in each tree T_x where v is an ancestor of c, v needs to reduce its score for T_x by c's score for T_x since all of those descendant leaves have been eliminated. In [3] an $O(n)$-round pipelined algorithm (using the in-tree property for CSSSP in Lemma III.7) is given for this update at all nodes in all trees, and this suffices for our purposes.

2. *Updates at Descendants.* For each v, in each tree T_x where v is a descendant of c, v needs to reduce its score for T_x to zero, since all descendant leaves are eliminated once c is removed. In [3] this computation is performed by an $O(nh)$-round precomputation in which each vertex identifies all of its ancestors in all of the h-hop trees and thereafter can readily identify the trees in which it is a descendant of a newly chosen blocker node c once c broadcasts its identity to all nodes. But this is too expensive for our purposes.

Here, we perform no precomputation but instead in Algorithm 4 we use the property in Lemma III.6 for CSSSP to

develop a method similar to the one for updates at ancestors. Initially c creates a list, $list_c$, where it adds the IDs of all the source nodes x such that c lies in tree T_x. In round i, c sends the i-th entry $\langle x \rangle$ in $list_c$ to all its children in T_x. Since T (in Lemma III.6) is a tree, every node v receives at most one message in a given round r. If v receives the message for source x in round r, it forwards this message to all its children in T_x in the next round, $r+1$, and also sets its score for source x to 0. Similar to the algorithm for updating ancestors of c [3], it is readily seen that every descendant of c in every tree T_x receives a message for x by round $k+h-1$.

Algorithm 4 Pipelined Algorithm for updating scores at v in trees T_x in which v is a descendant of newly chosen blocker node c

Input: Q: blocker set, c: newly chosen blocker node, S: set of sources

(only for c)

1: **Local Step at c:** create $list_c$ to store the ID of each source $x \in S$ such that $score_x(c) \neq 0$; **for each** $x \in S$ **do** set $score_x(c) \leftarrow 0$; set $score(c) \leftarrow 0$

2: **Send: Round** i: let $\langle x \rangle$ be the i-th entry in $list_c$; send $\langle x \rangle$ to c's children in T_x.

(round $r > 0$: for vertices $v \in V - Q - \{c\}$)

3: **send[lines 3-4]:** **if** v received a message $\langle x \rangle$ in round $r-1$ **then**

4: **if** $v \neq x$ **then** send $\langle x \rangle$ to v's children in T_x

5: **receive[lines 5-6]:** **if** v receives a message $\langle x \rangle$ **then**

6: $score(v) \leftarrow score(v) - score_x(v)$; $score_x(v) \leftarrow 0$

Lemma III.8. *Algorithm 4 correctly updates the scores of all nodes v in every tree T_x in which v is a descendant of c in $k+h-1$ rounds.*

IV. Additional Results

We consider the problem of finding a $(1+\epsilon)$-approximate solution to the weighted APSP problem. If edge-weights are strictly positive, the following result is known.

Theorem IV.1 ([16], [18]). *There is a deterministic algorithm that computes $(1+\epsilon)$-approximate APSP on graphs with positive polynomially bounded integer edge weights in $O((n/\epsilon^2) \cdot \log n)$ rounds.*

The above result does not hold when *zero weight edges* are present. Here we match the deterministic $O((n/\epsilon^2) \cdot \log n)$-round bound for this problem with an algorithm that also handles zero edge-weights.

We first compute reachability between all pairs of vertices connected by zero-weight paths. This is readily computed in $O(n)$ rounds, e.g., using [12], [17] while only considering only the zero weight edges (and ignoring the other edges).

We then consider shortest path distances between pairs of vertices that have no zero-weight path connecting them. The weight of any such path is at least 1. To approximate these paths we increase the zero edge-weights to 1 and transform every non-zero edge weight $w(e)$ to $n^2 \cdot w(e)$. Let this modified graph be $G' = (V, E, w')$. Thus the weight of an l-hop path p in G', $w'(p)$, satisfies $w'(p) \leq w(p) \cdot n^2 + l$. Since the modified graph G' has polynomially bounded positive edge weights, we can use the result in Theorem IV.1 to compute $(1+\epsilon/3)$-approximate APSP on this graph in $\tilde{O}(9n/\epsilon^2)$ rounds.

Fix a pair of vertices u, v. Let p be a shortest path from u to v in G with hop-length l. Then $w'(p) \leq n^2 \cdot w(p) + l$. Let p' be a $(1+\epsilon/3)$-approximate shortest path from u to v. Then $w'(p') \leq (1+\epsilon/3) \cdot w'(p) \leq (1+\epsilon/3)(n^2 \cdot w(p) + l)$. Dividing $w'(p')$ by n^2 gives us $w'(p')/n^2 < (1+\epsilon/3)(w(p)+(l/n^2)) < w(p) + w(p)\epsilon/3 + 2/n \leq w(p)(1+\epsilon)$ (as long as $\epsilon > 3/n$ and since $w(p) \geq 1$), and this establishes Theorem I.5.

V. Conclusion

We have presented new deterministic distributed algorithms for weighted shortest paths (both APSP, and for k sources) in graphs with moderate non-negative integer weights. Our work leaves a couple of major open problems. We could obtain a deterministic $\tilde{O}(n^{4/3})$-round APSP algorithm with non-negative polynomially bounded integer weights if our pipelined strategy can be made to works with Gabow's scaling technique [9]. Our current algorithm assumes that all sources see the same weight on each edge, while in the scaling algorithm each source sees a different edge weight on a given edge. While this can be handled with n different SSSP computations in conjunction with the randomized scheduling result of Ghaffari [10], it will be very interesting to see if a deterministic pipelined strategy could achieve the same result.

The other major open problem left by our work and the result in [3] is to is to investigate further improvements to the deterministic distributed computation of a blocker set.

References

[1] U. Agarwal and V. Ramachandran. A deterministic distributed algorithm for weighted apsp through pipelining. *arXiv preprint arXiv:1807.08824.v2*, 2018.

[2] U. Agarwal and V. Ramachandran. New and simplified distributed algorithms for weighted all pairs shortest paths. *arXiv preprint arXiv:1810.08544*, 2018.

[3] U. Agarwal, V. Ramachandran, V. King, and M. Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in $\tilde{O}(n^{3/2})$ rounds. In *Proc. PODC*, pages 199–205. ACM, 2018.

[4] R. Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[5] A. Bernstein and D. Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In *Proc. STOC*. ACM, 2019.

[6] K. Censor-Hillel, S. Khoury, and A. Paz. Quadratic and near-quadratic lower bounds for the congest model. In *Proc. DISC*, 2017.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[8] M. Elkin. Distributed exact shortest paths in sublinear time. In *Proc. STOC*, pages 757–770. ACM, 2017.

[9] H. N. Gabow. Scaling algorithms for network problems. *J. Comp. Sys. Sci.*, 31(2):148–168, 1985.

[10] M. Ghaffari. Near-optimal scheduling of distributed algorithms. In *Proc. PODC*, pages 3–12. ACM, 2015.

[11] M. Ghaffari and J. Li. Improved distributed algorithms for exact shortest paths. In *Proc. STOC*, pages 431–444. ACM, 2018.

[12] L. Hoang, M. Pontecorvi, R. Dathathri, G. Gill, B. You, K. Pingali, and V. Ramachandran. A round-efficient distributed betweenness centrality algorithm. In *Proc. PPoPP*. ACM, 2019.

[13] C.-C. Huang, D. Nanongkai, and T. Saranurak. Distributed exact weighted all-pairs shortest paths in $\tilde{O}(n^{5/4})$ rounds. In *Proc. FOCS*, pages 168–179. IEEE, 2017.

[14] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. FOCS*, pages 81–89. IEEE, 1999.

[15] S. Krinninger and D. Nanongkai. A faster distributed single-source shortest paths algorithm. In *Proc. FOCS*. IEEE, 2018.

[16] C. Lenzen and B. Patt-Shamir. Fast partial distance estimation and applications. In *Proc. PODC*, pages 153–162. ACM, 2015.

[17] C. Lenzen and D. Peleg. Efficient distributed source detection with limited bandwidth. In *Proc. PODC*, pages 375–382. ACM, 2013.

[18] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. STOC*, pages 565–573. ACM, 2014.