The Sparsest Additive Spanner via Multiple Weighted BFS Trees

Keren Censor-Hillel[‡]   Ami Paz[§]   Noam Ravid[‡]

Abstract

Spanners are fundamental graph structures that sparsify graphs at the cost of small stretch. In particular, in recent years, many sequential algorithms constructing additive all-pairs spanners were designed, providing very sparse small-stretch subgraphs. Remarkably, it was then shown that the known (+6)-spanner constructions are essentially the sparsest possible, that is, larger additive stretch cannot guarantee a sparser spanner, which brought the stretch-sparsity trade-off to its limit. Distributed constructions of spanners are also abundant. However, for additive spanners, while there were algorithms constructing (+2) and (+4)-all-pairs spanners, the sparsest case of (+6)-spanners remained elusive.

We remedy this by designing a new sequential algorithm for constructing a (+6)-spanner with the essentially-optimal sparsity of $\tilde{O}(n^{4/3})$ edges. We then show a distributed implementation of our algorithm, answering an open problem in [13].

A main ingredient in our distributed algorithm is an efficient construction of multiple weighted BFS trees. A weighted BFS tree is a BFS tree in a weighted graph, that consists of the lightest among all shortest paths from the root to each node. We present a distributed algorithm in the CONGEST model, that constructs multiple weighted BFS trees in $|S|+D-1$ rounds, where S is the set of sources and D is the diameter of the network graph.

Keywords: Distributed graph algorithms, congest model, weighted BFS trees, additive spanners

---

[‡]Department of Computer Science, Technion, Israel.
{`ckeren,noamrvd`}`@cs.technion.ac.il`.
[§]Faculty of Computer Science, University of Vienna, Austria. `ami.paz@univie.ac.at`.

1   Introduction

A spanner of a graph G is a spanning subgraph H of G that approximately preserves distances. Spanners find many applications in distributed computing [12, 14, 47, 48, 51], and thus their distributed construction is the center of many research papers. We focus on spanners that approximately preserve distances between all pairs of nodes, and where the stretch is only by an additive factor (*purely-additive all-pairs* spanners).

Out of the abundant research on distributed constructions of spanners, only two papers discuss the construction of *purely additive* spanners in the CONGEST model: the construction of (+2)-spanners is discussed in [39], and the construction of (+4)-spanners and (+8)-spanners in [13], along with other types of additive spanners and lower bounds. However, the distributed construction of (+6)-spanners remained elusive, stated explicitly as an open question in [13]. This is especially important since additive factors greater then 6 cannot yield essentially sparser spanners [2].

In this paper, we give a distributed algorithm for constructing a (+6)-spanner, with an optimal number of edges up to sub-polynomial factors; our spanner is even sparser than the (+8)-spanner presented in [13]. Several sequential algorithms building (+6)-spanners were presented, but none of them seems to be appropriate for a distributed setting. Thus, to achieve our result we also present a new, simple sequential algorithm for constructing (+6)-spanners, a result that could be of independent interest.

As a key ingredient, we provide a distributed construction of *multiple weighted BFS trees*. Constructing a breadth-first search (BFS) tree is a central task in many computational settings. In the classic synchronous distributed setting, constructing a BFS tree from a given source is straightforward. Due to its importance, this task has received much attention in additional distributed settings, such as the asynchronous setting (see, e.g., [45] and references therein). Moreover, at the heart of many distributed applications lies a graph structure that represents the edges of *multiple* BFS trees [34, 38], which are rooted at the nodes of a given subset $S \subseteq V$, where $G = (V, E)$ is the underlying communication graph. Such a structure is used in distance computation and estimation [32, 34, 38], routing table construction [38], spanner construction [13, 38, 39], and more.

When the bandwidth is limited, constructing multiple BFS trees efficiently is a non-trivial task. Indeed, distributed constructions of multiple BFS trees in the CONGEST model [45], where in each round of communication every node can send $O(\log n)$-bit messages to each of its neighbors, have been given in [34, 38]. They showed that it is possible to build BFS trees from a set of sources S in $O(|S| + D)$ rounds, where D is the diameter of the graph G; it is easy to show that this is asymptotically tight.

In some cases, different edges of the graph may have different attributes, which can be represented using edge weights. The existence of edge weights has been extensively studied in various tasks, such as finding or approximating lightest paths [4, 5, 10, 22, 24, 29, 31, 35, 38, 42], finding a minimum spanning tree (MST) in the graph [6, 15, 27], finding a maximum matching [15, 40], and more. However, as far as we are aware, no study addresses the problem of constructing multiple weighted BFS (WBFS) trees, where the goal is not to find the lightest paths from the sources to the nodes, but rather the *lightest shortest paths*. That is, the path in a WBFS tree from the source s to a node v is the lightest among all shortest paths from s to v in G.

Thus, we provide an algorithm that constructs multiple WBFS trees from a set of source nodes S in the CONGEST model. Our algorithm completes in $|S| + D - 1$ rounds, which implies that no overhead is needed for incorporating the existence of weights.

While our multiple WBFS algorithm can be used for graphs that have initial edge weights, we actually use it to construct (+6)-spanners in unweighted graphs, to which we artificially add edge weights to distinguish "desired" edges from "undesired" ones. More generally, this algorithm can be used to find *consistent shortest paths* in an unweighted graph: a set of shortest

paths is consistent if, given four nodes s, t, u and v such that u and v lie on the shortest (s, t)-path, the shortest (u, v)-path is a subpath of the shortest (s, t)-path. Such structures have many applications, but achieving them is non-trivial, especially in the distributed setting. Our WBFS algorithm can be used for this goal, by assigning the edges of an unweighted graph random weights of polynomial size. The *isolation lemma* [41] then guarantees that with high probability, the lightest shortest paths derived from this weight function will constitute a set of consistent shortest paths.

1.1 Our contribution

At a high level, our approach for building multiple WBFS trees is to generalize the algorithm of Lenzen et al. [38] in order to handle weights. In [38], the messages are pairs consisting of a source node and a distance, which are prioritized by the distance traversed so far. When incorporating weights into this framework it makes sense to use triplets instead of pairs, where each triplet also contains the weight of the respective path. However, it may be that a node v needs to send multiple messages that correspond to the same source and the same distance but contain different weights, since congestion over edges may cause the respective messages to arrive at v in different rounds and, in the worst case, in a decreasing order of weights. The challenge in generalizing this framework therefore lies in guaranteeing that despite the need to consider weights, we can carefully choose a total order to prioritize triplets, such that not too many messages need to be sent, allowing us to handle congestion. Our construction and its proof appear in Section 3, giving the following.

Theorem 1. *Given a weighted graph $G = (V, E, w)$ and a set of nodes $S \subseteq V$, there exists an algorithm for the* CONGEST *model that constructs a WBFS tree rooted at s, for every $s \in S$, in $|S| + D - 1$ rounds.*

Our multiple WBFS trees construction can have many applications, as it allows consistent tie-breaking. Here, we present one such application: pinning down the question of constructing $(+6)$-spanners in the CONGEST model. The construction of additive spanners in the CONGEST model was studied beforehand [13, 39], but the $+6$ case remained unresolved, for reasons we describe below. Naturally, the quality of a spanner is measured by its sparsity, which is the motivation for allowing some stretch in the distances to begin with, and different spanners present different tradeoffs between stretch and sparsity. The properties of our $(+6)$-spanner construction algorithm are summarized in the following theorem.[i]

Theorem 2. *There exists an algorithm for the* CONGEST *model that constructs a $(+6)$-spanner with $O\left(n^{4/3} \log^{4/3} n\right)$ edges in $O\left(\frac{n^{2/3}}{\log^{1/3} n} + D\right)$ rounds and succeeds w.h.p.*

Previous distributed algorithms for spanners similar to ours, i.e., purely additive all-pairs spanners, construct a $(+2)$-spanner with $\tilde{O}(n^{3/2})$ edges in $\tilde{O}(n^{1/2} + D)$ rounds [39], a $(+4)$-spanner with $\tilde{O}(n^{7/5})$ edges in $\tilde{O}(n^{3/5} + D)$ rounds [13], and a $(+8)$-spanner with $\tilde{O}(n^{15/11})$ edges in $\tilde{O}(n^{7/11} + D)$ rounds [13]. Hence, our algorithm is currently the best non-trivial spanner construction algorithm in terms of density, sparser even than the previous $(+8)$-spanner. The parameters of our algorithm are tuned to achieve the best sparsity possible, and interestingly, one can change the algorithm and achieve a worst sparsity of $\tilde{O}(n^{15/11})$ edges in $\tilde{O}(n^{7/11} + D)$ rounds. These are the same parameters of the $(+8)$-spanner algorithm, but with a better stretch. The option of getting even sparser spanners by allowing more stretch was essentially ruled out [2], while the question of improving the running time remains open for all stretch parameters.

---

[i]We use w.h.p. to indicate a probability that is at least $1 - 1/n^c$ for some constant $c \geq 1$ of choice.

1.2 Other spanner construction algorithms

Previous distributed spanner construction algorithms all build upon known sequential algorithms, and present distributed implementations of them, or of a slight variant of them [13, 39]. For example, many sequential algorithms start in a clustering phase, where stars around high-degree nodes are added to the spanner one by one. Implementing this directly in the distributed setting will take too long; instead, we use a classical approach of choosing cluster centers at random, which yields almost as good results, and can be implemented in a constant time. Similar methods are used for implementing other parts of the construction. However, the approach of finding a distributed implementation for a sequential algorithm fails for all known (+6)-spanner algorithms, as described next. Thus, we introduce a new sequential algorithm for the problem, and then present its distributed implementation.

There are three known approaches for the design of sequential (+6)-spanner algorithms. The first, presented by Baswana et al. [8], is based on measuring the quality of paths in terms of *cost* and *value*, and adding to the spanner only paths which are "affordable". This approach was later extended by Kavitha [36] to other families of additive spanners. The second approach, presented by Woodruff [52], uses a subroutine that finds almost-shortest paths between pairs of nodes, and obtains a faster algorithm at the expense of a slightly worst sparsity guarantee. The third approach, presented by Knudsen [37], is based on repeatedly going over pairs of nodes, and adding a shortest path between a pair of nodes to the spanner if their current distance in the spanner is too large.

Unfortunately, direct implementation in the CONGEST model of the known sequential algorithms is highly inefficient. We are not aware of fast distributed algorithms that allow the computation of the cost and value of paths needed for the algorithm of [8]. Similarly, for [52], the almost-shortest paths subroutine seems too costly for the CONGEST model. The algorithm of [37] needs repeated updates of the distances in the spanner between pairs of nodes after every addition of a path to it, which is a sequential process in essence, and thus we do not find it suitable for an efficient distributed implementation.

A different approach for the distributed construction of (+6)-spanners could be to adapt a distributed algorithm with different stretch guarantees to construct a (+6)-spanner. This approach does not seem to work: the distributed algorithms for constructing (+2)-spanners [39] and (+4)-spanner [13] are both very much tailored for achieving the desired stretch, and it is not clear how to change them in order to construct sparser spanners with higher stretch. The (+8)-spanner construction algorithm [13] starts with clustering, and then constructs a (+4)-*pairwise spanner* between the cluster centers. Replacing the (+4)-pairwise spanner by a (+2)-pairwise spanner will indeed yield a (+6)-all-pairs spanner, as desired. However, even using the sparsest (+2)-pairwise spanners [1, 13], the resulting (+6)-spanner may have $\tilde{O}(n^{5/3})$ edges, denser than our new (+6)-spanner and than the known (+8)-spanner [13].

Thus, we start by presenting a new sequential algorithm for the construction of (+6)-spanners, an algorithm that is more suitable for a distributed implementation, and then discuss its distributed implementation. Our construction starts with a clustering phase, and then adds paths that minimize the number of additional edges that need to be added to the spanner. To implement our construction in the CONGEST model, we assign weights to the edges and use our WBFS algorithm to find shortest paths with as few edges as possible that are not yet in the spanner. Note that although the graph and the spanner we construct for it are both unweighted, the ability of our multiple WBFS algorithm to handle weights is crucial for our solution.

A (+6)-spanner must contain $n^{4/3}/2^{O(\sqrt{\log n})}$ edges [2]. The best sequential algorithms [8, 37] construct a spanner with $O(n^{4/3})$ edges. Our distributed algorithm constructs a spanner with $O(n^{4/3} \log^{4/3} n)$ edges, which is slightly denser than optimal but still sparser than the $O(n^{4/3} \log^3 n)$ edges in the fast sequential construction of [52].

1.3 Related work

A natural approach for building multiple (unweighted) BFS trees in the CONGEST model is to start the construction simultaneously from all sources, sending pairs composed of a source id and the distance the message have traversed so far. The main challenge in this approach appears when multiple such pairs should be sent on one edge in a single round. In this case, the algorithm needs to locally prioritize the message, in a way that will not compromise the correctness of the constructed BFS trees.

One approach for prioritizing the messages is by source id, i.e., have a queue at each node sorted by source id. However, this alone does not work and might create incorrect BFS trees. Attempts to overcome the problems of this approach by having a different queue for each outgoing edge [34] are also insufficient [33].

An elegant way to resolve this is to sort messages not by source id, but by the distance they traversed so far, with a preference to messages that traversed smaller distances. Such an algorithm was suggested by Lenzen and Peleg [38], finishing in $(|S| + D - 1)$ rounds when executed form a set S of sources. We base our multiple WBFS construction algorithm on this algorithm. A variant of the Lenzen-Peleg algorithm, with a reduced message complexity, was recently suggested in [50].

Spanners were first introduced in 1989 [46, 47], and since then have been a topic for wide research due to their abundant applications. Prime examples for the need for sparse spanners can be found in synchronizing distributed networks [47], information dissemination [12], compact routing schemes [14, 48, 51], and more.

Distributed constructions of various spanners have been widely studied [7–9, 11, 13, 16–21, 23, 25, 26, 28, 30, 38, 39, 43, 44, 49]. Lower bounds were given in [3, 13, 49]. However, obtaining an efficient and sparse (+6)-all-pairs spanner has remained an open question [13].

Several lower bounds for the time complexity of spanner construction in the CONGEST model were presented in [13], but these are applicable only to pairwise spanners with a bounded number of pairs, and not to all-pairs spanners. A lower bound from [49] states that the construction of a spanner with $\tilde{O}(n^{4/3})$ edges, such as the one we build, must take $\tilde{\Omega}(n^{3/8})$ rounds. This lower bound does not take into account the bandwidth restrictions at all (it is proven for the LOCAL model), and so we believe that a higher lower bound for the CONGEST model should apply, but this is left as an intriguing open question.

2 Preliminaries

All graphs in this work are simple, connected and undirected. A graph can be unweighted, $G = (V, E)$, or weighted $G = (V, E, w)$ with $w : E \to \{0, \dots, W\}$, in which case we assume $W \in \text{poly}(n)$. Given a path ρ in a weighted graph G, we use $|\rho|$ to denote the *length* of ρ, which is the number of edges in it, and $w(\rho)$ to denote the *weight* of the path, which is the sum of its edge-weights. The *distance* between two nodes u, v in a graph G, denoted $\delta_G(u, v)$, is the minimum length of a path in G connecting u and v. The *diameter* of a graph (weighted or unweighted) is $D = \max_{u,v \in V} \{\delta_G(u, v)\}$.

We consider the CONGEST model of computation [45], where the nodes of a graph communicate synchronously by exchanging $O(\log n)$-bit messages along the edges. The goal is to distributively solve a problem while minimizing the number of communication rounds.

WBFS trees: We are interested in a weighted BFS tree, which consists of all *lightest shortest paths* from the root, formally defined as follows.

Definition 3. *Given a connected, weighted graph $G = (V, E, w)$ and a node $s \in V$, a weighted BFS tree (WBFS) for G rooted at s is a spanning tree T_s of G satisfying the following properties:*

(i) *For each $v \in V$, the path from s to v in T is a shortest path in G between s and v.*

(ii) *For each $v \in V$, no shortest path from s to v in G is lighter than the path from s to v in T.*

We emphasize that this is different than requiring a subgraph containing all *lightest paths* from the root. One may wonder if a WBFS tree always exists, but this is easily evident by the following refinement of a (sequential) BFS search, returning a WBFS tree: go over the nodes in an order of non-decreasing distances from the source s, starting with $w(s) = 0$; each node v chooses as a parent a neighbor u that was already processed and minimizes $w(v) = w(u) + w(u, v)$, and adds the edge $\{u, v\}$ to the tree. Each node has a single parent, so this is indeed a tree; the node ordering guarantees that this is indeed a BFS tree, assuring (i); and the parent choice guarantees the paths are lightest among the shortest, assuring (ii). In fact, this can be seen as an algorithm implementing a consistent tie-breaking strategy between paths of equal lengths, and thus, it is not surprising its output is a tree.

Spanners: Given a graph $G = (V, E)$, a subgraph $H = (V, E')$ of G is called an (α, β)-*spanner* if for every $u, v \in V$ it holds that $\delta_H(u, v) \leq \alpha \delta_G(u, v) + \beta$. The parameters α and β are called the *stretch parameters*.

When $\alpha = 1$, such a spanner is called a *purely additive spanner*. In this paper we focus on purely additive $(+6)$-spanners, i.e., $\alpha = 1$ and $\beta = 6$.

For completeness, we mention that when $\beta = 0$, such a spanner is called a *multiplicative spanner*. In addition, while sometimes the stretch parameters need to be guaranteed only for some subset of all the pairs of nodes of the graph (such as in *pairwise spanners*), we emphasize that our construction provides the promise of a $+6$ stretch for *all* pairs.

3 Multiple Weighted BFS Trees

In the CONGEST model, the problem of finding a WBFS tree requires each node to know its parent in the WBFS tree, and the unweighted and weighted distances to the source within the tree. This allows the node to send messages to the source node through the lightest among all shortest paths. When there are multiple sources, each node should know the parent leading to each of the sources in S.

We define data structures for representing multiple WBFS trees. Given a node $v \in V$, the *S-proximity-list* (or *proximity list* for short) of v, noted PL_v^*, is an ascending lexicographically ordered list of triples $(d(s, v), s, w(s, v))$, where $d(s, v)$ and $w(s, v)$ are the length and weight of the path from s to v in T_s. Two different triples are ordered such that $(d(s, v), s, w(s, v)) < (d(t, v), s, w(t, v))$ if $d(s, v) < d(t, v)$, or $d(s, v) = d(t, v)$ and $s < t$, where s and t may be compared by any predefined order on the node identifiers. Note that T_s contains a single path from s to v, so PL_v^* cannot contain two triplets with $d(s, v) = d(t, v), s = t$ and $w(s, v) \neq w(t, v)$.

The *S-path-map* (or *path-map* for short) of v is a mapping from each source $s \in S$ to the parent of v in T_s, noted by PM_v^*. The list PM_v^* is sorted with respect to the order of PL_v^*, such that the first records of PM_v^* belong to sources closest to v.

Algorithm 1, which constructs multiple WBFS trees from a set S in the CONGEST model, is based on carefully extending the distributed Bellman-Ford-based algorithm of Lenzen et al. [38]. The heart of the algorithm is a loop (Line 7), and each iteration of it takes a single round in the CONGEST model. We show that $|S| + D - 1$ iterations of the loop suffice in order to construct the desired WBFS trees.

The algorithm builds the WBFS trees by gradually updating the proximity list and the path map of each node. Each round is composed of two phases: updating the neighbors about changes in the proximity list, and receiving updates from other nodes. The path map is only used by the current node, and therefore changes to it are not sent.

Algorithm 1: Weighted distributed Bellman-Ford algorithm for node v

---

1 $L_v \leftarrow ()$
2 **for** $s \in S$ **do**
3 $\quad$ $\mathtt{PM}_v(s) \leftarrow \perp$
4 **if** $v \in S$ **then**
5 $\quad$ $\mathtt{PL}_v \leftarrow ((0, v, 0))$
6 $\quad$ $\mathtt{sent}_v(0, v, 0) \leftarrow \mathtt{FALSE}$ $\qquad\qquad$ /* A variable marking sent triplets */
7 **for** $|S| + D - 1$ *rounds* **do**
8 $\quad$ **if** $\exists (d_s, s, w_s) \in \mathtt{PL}_v$ *such that* $\mathtt{sent}_v(d_s, s, w_s) = \mathtt{FALSE}$ **then**
9 $\quad\quad$ $(d_s, s, w_s) \leftarrow \min \{(d_t, t, w_t) \in \mathtt{PL}_v \text{ such that } \mathtt{sent}_v(d_t, t, w_t) = \mathtt{FALSE}\}$
10 $\quad\quad$ send (d_s, s, w_s) to all neighbors
11 $\quad\quad$ $\mathtt{sent}_v(d_s, s, w_s) \leftarrow \mathtt{TRUE}$
12 $\quad$ **for** *received* (d_s, s, w_s) *from* $u \in V$ **do**
13 $\quad\quad$ $d_s \leftarrow d_s + 1$
14 $\quad\quad$ $w_s \leftarrow w_s + w(u, v)$
15 $\quad\quad$ **if** $\nexists (d'_s, s, w'_s) \in \mathtt{PL}_v$ *such that* $(d'_s < d_s$ *or* $(d'_s = d_s$ *and* $w'_s < w_s))$ **then**
16 $\quad\quad\quad$ $\mathtt{PL}_v \leftarrow \mathtt{PL}_v \setminus \{(\cdot, s, \cdot)\}$
17 $\quad\quad\quad$ $\mathtt{PL}_v \leftarrow \mathtt{PL}_v \cup \{(d_s, s, w_s)\}$
18 $\quad\quad\quad$ $\mathtt{PM}_v(s) \leftarrow u$
19 $\quad\quad\quad$ $\mathtt{sent}_v(d_s, s, w_s) \leftarrow \mathtt{FALSE}$

Ideally, each node would update its neighbors regarding all the changes made to its proximity list. However, due to bandwidth restrictions, a node cannot send the entire list in each round. Therefore, at each round each node sends to all of its neighbors the lexicographically smallest triplet in its proximity list that it has not yet sent, while maintaining a record noting which triplets have been sent and which are waiting. Each triplet is only sent once, though a node may send multiple triplets regarding a single source.

A node uses the messages received in the current round in order to update its proximity list and path map for the next round. A triplet (d_s, s, w_s) received by a node v from a neighbor u represents the length d_s and weight w_s of some path ρ from s to u in the graph. The node v then considers the extended path $\rho' = \rho \circ v$ from s to v, compares it to its currently known best path from s to v, and updates the proximity list and path map in case a shorter path has been found, or a lighter path with the same length.

To prove correctness, we generalize the proof of [38] to handle weights, and show that our algorithm solves the *weighted (S, d, k)-detection* problem: each node should learn which are the sources from S closest to it, but at most k of them and only up to distance d. This is formally defined as follows.

Definition 4. *Given a weighted graph $G = (V, E, w)$, a subset $S \subseteq V$ of source nodes, and a node $v \in V$, let $\mathtt{PL}_v^*$ denote the S-proximity-list and let $\mathtt{PM}_v^*$ denote the path map of the node v. The* weighted (S, d, k)-detection problem *requires that each node $v \in V$ learns the first $\min \{k, \lambda_v^d\}$ entries of $\mathtt{PL}_v^*$ and $\mathtt{PM}_v^*$, where λ_v^d is the number of sources $s \in S$ such that $d(s, v) \leq d$.*

Given a node v, $\mathtt{PL}_v$ is a variable in Algorithm 1 holding the proximity list of v, and we denote by $\mathtt{PL}_v^{(r)}$ the state of the list $\mathtt{PL}_v$ at the beginning of round r of the algorithm, and by $\mathtt{PL}_v^{(\infty)}$ the value of $\mathtt{PL}_v$ at the end of the algorithm. Recall that $\mathtt{PL}_v^*$ is the true proximity list, so our goal is proving $\mathtt{PL}_v^{(\infty)} = \mathtt{PL}_v^*$, i.e., proving that the algorithm obtains the correct values of the proximity list.

We use similar notations for the path map $\mathtt{PM}_v^*$. Since the records of $\mathtt{PM}_v$ are updated under the same conditions as the records of $\mathtt{PL}_v$, the correctness of $\mathtt{PM}_v$ at the end of the algorithm with respect to $\mathtt{PM}_v^*$ immediately follows, and we omit the details.

We start by showing that if there was no bound on the number of rounds, then the values of $\mathtt{PL}_v$ would have eventually converged to the true values of $\mathtt{PL}_v^*$.

Lemma 5. *Given a graph $G = (V, E, w)$ and a set $S \subseteq V$, if we let the for loop in Line 7 of Algorithm 1 to run forever, then there exists a round $r_0 \in \mathbb{N}$ such that no node $v \in V$ sends messages or modifies $\mathtt{PL}_v$ after round r_0. Moreover, $\mathtt{PL}_v^{(r_0)} = \mathtt{PL}_v^*$, i.e., for every $(d_s, s, w_s) \in \mathtt{PL}_v^{(r_0)}$, it holds that $d_s = d(v, s)$ and $w_s = \min \{w(\rho) \mid \rho$ connects v with s, and $|\rho| = d_s\}$.*

Proof. In each iteration of the algorithm, each node $v \in V$ sends to all of its neighbors the first triplet (d_s, s, w_s) such that $\mathtt{sent}_v(d_s, s, w_s) = \mathtt{FALSE}$. Each triplet received is sent at most once. Therefore, if we show the existence of a round r_0 where for each $v \in V$, all messages in $\mathtt{PL}_v^{(r_0)}$ have been sent in previous rounds, it implies that no message is sent in round r_0, and hence $\mathtt{PL}_v^{(r_0+1)} = \mathtt{PL}_v^{(r_0)}$. This claim is applied inductively, concluding that for any round $r > r_0$, it holds that $\mathtt{PL}_v^{(r)} = \mathtt{PL}_v^{(r_0)}$. We prove the existence of the round r_0 by showing that the number of messages that are sent by each node $v \in V$ is finite.

Each triple (d_s, s, w_s) is inserted into $\mathtt{PL}_v$ when a message is received from another node $u \in V$, and such a message implies that a path from s to v through u of length d_s and weight w_s exists. Thus, the number of messages (d_s, s, w_s) sent by a node v is upper bounded by the number of paths from s to v of length d_s. Furthermore, the algorithm does not insert a message into $\mathtt{PL}_v$ if it has already inserted a lexicographically smaller message from the same source. As the graph is finite, the number of paths is bounded, and eventually no node adds further triplets to its lists or sends additional messages.

It remains to show that for all $s \in S$ and $v \in V$ it holds that $(d(s, v), s, w(s, v)) \in \mathtt{PL}_v^{(r_0)}$.

First, we show that if a triplet (d_s, s, w_s) is added to $\mathtt{PL}_v$ in some round r, then there exists a path ρ from s to v such that $d_s = |\rho|$ and $w_s = w(\rho)$. For a source $s \in S$, we insert the triplet $(0, s, 0)$ into $\mathtt{PL}_s$ at the beginning of the algorithm, so the claim is true at initialization. Assume there exists a round where a triplet (d_s, s, w_s) is inserted into $\mathtt{PL}_v$ but no corresponding path exists, and let r be the first such round. This implies that there exists a node $u \in V$ that is a neighbor of v, which sends the message $(d_s - 1, s, w_s - w(u, v))$ to v in round r. The triplet $(d_s - 1, s, w_s - w(u, v))$ must have been inserted into $\mathtt{PL}_u$ in some round $r' < r$, and by the minimality of r there exists a path ρ from s to u where $|\rho| = d_s - 1$ and $w(\rho) = w_s - w(u, v)$. Since u is a neighbor of v, the path $\rho' = \rho \circ v$ is valid, satisfying $|\rho'| = d_s$ and $w(\rho') = w_s$, contradicting the assumption.

To complete the proof, we claim that the correct triplet $(d(s, v), s, w(s, v))$ is indeed added to $\mathtt{PL}_v$ at some round of the algorithm, and is not removed. Consider the path from s to v in the WBFS tree T_s, denoted by $\rho = (v_0 = s, v_1, \ldots, v_{d(s,v)} = v)$. At initialization, the triplet $(0, s, 0)$ is added to $\mathtt{PL}_s$. From then, at each round there exists some $i \le d(s, v)$ such that $(d(s, v_i), s, w(s, v_i)) \in \mathtt{PL}_{v_i}$ and $\mathtt{sent}_{v_i}(d(s, v_i), s, w(s, v_i)) = \mathtt{FALSE}$. Since we proved that this message is eventually sent, this implies that in the beginning of the next round, the triplet $(d(s, v_{i+1}), s, w(s, v_{i+1}))$ is added to $\mathtt{PL}_{v_{i+1}}$. By the definition of a WBFS tree, all other paths from s to v must be longer or not lighter, implying the triplet cannot be discarded for a lexicographically smaller triplet. This concludes that for any source s and node v, it holds that $(d(s, v), s, w(s, v)) \in \mathtt{PL}_v^{(r_0)}$. $\qquad\square$

Lemma 5 shows that without the limit on the number of rounds, the algorithm would compute the right values; however, it does not bound the number of rounds needed for this to occur. Next, we show that $|S| + D - 1$ rounds suffice. We cannot apply the claims of [38] directly, since the existence of weights restricts the number of viable solutions even further, causing more updates to the proximity list and an increase in the number of messages sent. However, we do

use a similar technique: we bound the number of rounds in which the k smallest entries of $\mathtt{PL}_v$ can change.

For an entry $(d_s, s, w_s) \in \mathtt{PL}_v^{(r)}$, let $\ell_v^{(r)}(d_s, s, w_s)$ denote the index of the entry in the lexicographically ordered list $\mathtt{PL}_v^{(r)}$ at the beginning of round r. For completeness, we define $\ell_v^{(r)}(d_s, s, w_s) = -\infty$ if (d_s, s, w_s) did not appear in $\mathtt{PL}_v$ at the beginning of round r, and $\mathtt{PL}_v = \infty$ if the triplet was removed from $\mathtt{PL}_v$ before the beginning of this round. Note that a removed triplet is never returned to the list, since the lexicographical order is transitive.

Lemma 6. *For a triplet (d_s, s, w_s), the following holds:*

(i) $\ell_v^{(r)}(d_s, s, w_s)$ *is non-decreasing with r.*

(ii) *When a triplet (d_s, s, w_s) is sent from a node u to a node v at round r, it causes the addition of a new triplet (d_s', s, w_s') to $\mathtt{PL}_v$ at the end of round r. This triplet satisfies $d_s' = d_s + 1$, $w_s' = w_s + w(u, v)$, and $\ell_u^{(r)}(d_s, s, w_s) \leq \ell_v^{(r+1)}(d_s', s, w_s')$.*

Part (i) follows from the fact that the number of triplets below (d_s, s, w_s) cannot decrease. To prove part (ii), we show that all the triplets below (d_s, s, w_s) in $\mathtt{PL}_u$ are sent from u to v and added to $\mathtt{PL}_v$ before (d_s, s, w_s) is sent and added.

Proof. Part (i) is a consequence of the method used by our algorithm for managing the list $\mathtt{PL}_v$. According to our algorithm, triplets are not removed from $\mathtt{PL}_v$ when they are sent. The only case in which a triplet (d_t, t, w_t) is removed from $\mathtt{PL}_v$ is when a lexicographically smaller triplet (d_t', t, w_t') is added to the list instead. When this happens in round r, it holds that $\ell_v^{(r)}(d_t, t, w_t) \geq \ell_v^{(r+1)}(d_t', t, w_t')$, since the new triplet is lexicographically smaller. Hence, for every other triplet $(d_s, s, w_s) \in \mathtt{PL}_v$, the number of lexicographically smaller triplets in $\mathtt{PL}_v$ cannot decrease throughout the algorithm.

We now turn to prove part (ii) of the lemma. Note that the only claim which does not follows trivially from the algorithm is the inequality $\ell_u^{(r)}(d_s, s, w_s) \leq \ell_v^{(r+1)}(d_s', s, w_s')$. By the fact that the triplet (d_s, s, w_s) is sent by the node u in round r, we conclude that the $\ell_u^{(r)}(d_s, s, w_s) - 1$ triplets preceding it in the list $\mathtt{PL}_u^{(r)}$ have already been sent by u in earlier rounds, and arrived at the node v. For each such triplet (d_t, t, w_t), either $d_t \leq d_s$, or $t < s$ and $d_t = d_s$. Therefore, when added to $\mathtt{PL}_v$ as $(d_t + 1, t, w_t + w(u, v))$ it is lexicographically smaller than (d_s', s, w_s'). At round r, either $(d_t + 1, t, w_t + w(u, v))$ is in $\mathtt{PL}_v^{(r)}$ or it was replaced by a lexicographically smaller triplet containing t. Thus, there are at least $\ell_u^{(r)}(d_s, s, w_s) - 1$ triplets smaller than (d_s', s, w_s') in $\mathtt{PL}_v^{(r+1)}$, and hence $\ell_u^{(r)}(d_s, s, w_s) \leq \ell_v^{(r+1)}(d_s', s, w_s')$. $\square$

Lemma 6 implies that as the algorithm progresses, messages at higher indexes of the proximity list are sent and updated. This can be used to obtain an upper bound on the round in which a triplet at a certain index of the proximity list can be sent or received, as formalized by the next lemma.

Lemma 7. *In round $r \in \mathbb{N}$ of Algorithm 1, a node $v \in V$ can:*

(i) *send a message (d_s, s, w_s) only if*

$$d_s + \ell_v^{(r)}(d_s, s, w_s) \geq r$$

(ii) *add to $\mathtt{PL}_v$ a triplet (d_s, s, w_s) only if*

$$d_s + \ell_v^{(r+1)}(d_s, s, w_s) > r$$

Part (i), when put in words, is rather intuitive: while a triplet might need to wait before being sent, the waiting time is bounded from above by the distance the triplet has traversed from its source, plus the number of triplets that were to be sent before it. Part (ii) is complementary to part (i): the time before a triplet is added, is, once more, bounded by the distance it traversed plus the number of lexicographically smaller triplets.

Proof. We start by showing that, for a given round r, if Lemma 7(i) holds for all nodes then Lemma 7(ii) holds as well. Consider a triplet (d'_s, s, w'_s) that is added to $\mathtt{PL}_v$ as a result of a message (d_s, s, w_s) sent from u to v in round r, where $d'_s = d_s + 1$ and $w'_s = w_s + w(u, v)$. Lemma 7(i) implies that $d_s + \ell_u^{(r)}(d_s, s, w_s) \geq r$, and by Lemma 6(ii) we have that $\ell_v^{(r+1)}(d'_s, s, w'_s) \geq \ell_u^{(r)}(d_s, s, w_s)$. As $d'_s > d_s$, we conclude

$$d'_s + \ell_v^{(r+1)}(d'_s, s, w'_s) > d_s + \ell_u^{(r)}(d_s, s, w_s) \geq r,$$

which implies Lemma 7(ii).

Next, we prove by induction that both parts of the lemma hold. In round 1, Lemma 7(i) holds trivially, since by definition $\ell_v^{(1)}(d_s, s, w_s) \geq 1$. Assume that Lemma 7 holds at round $r - 1$; we show the lemma holds at round r. Since Lemma 7(i) implies Lemma 7(ii), it is sufficient to show that every message (d_s, s, w_s) sent by some node $v \in V$ in round r satisfies $d_s + \ell_v^{(r)}(d_s, s, w_s) \geq r$.

Observe that if (d_s, s, w_s) is sent by a node v in round r, then the triplet must have been added to $\mathtt{PL}_v$ in some round $r' \leq r - 1$. If $r' = r - 1$, according to the induction hypothesis, Lemma 7(ii) holds and $d_s + \ell_v^{(r)}(d_s, s, w_s) > r - 1$, implying $d_s + \ell_v^{(r)}(d_s, s, w_s) \geq r$, since all the terms are integers.

Otherwise $r' < r - 1$. In this case, in round $r - 1$ the triplet (d_s, s, w_s) appeared in $\mathtt{PL}_v$ and was not yet sent. Since (d_s, s, w_s) is sent in round r, a different triplet (d_t, t, w_t) with $t \neq s$ must have been sent in round $r - 1$, implying:

$$d_s + \ell_v^{(r-1)}(d_s, s, w_s) > d_t + \ell_v^{(r-1)}(d_t, t, w_t).$$

By Lemma 6(i), we have that $\ell_v^{(r)}(d_s, s, w_s) \geq \ell_v^{(r-1)}(d_s, s, w_s)$, and combined with the induction hypothesis for Lemma 7(i) in round $r - 1$ we conclude:

$$d_s + \ell_v^{(r)}(d_s, s, w_s) \geq d_s + \ell_v^{(r-1)}(d_s, s, w_s) > d_t + \ell_v^{(r-1)}(d_t, t, w_t) \geq r - 1.$$

This gives that $d_s + \ell_v^{(r)}(d_s, s, w_s) \geq r$, since all the terms are integers. $\qquad \square$

Lemma 5 implies that eventually, the lists $\mathtt{PL}_v$ converge to contain the correct values, and Lemma 7 restricts the number of rounds in which specific list entries may change. From this, we conclude that the algorithm solves the weighted (S, d, k)-detection problem.

Lemma 8. *Given an instance of the weighted (S, d, k)-detection problem, for every $v \in V$ and round r of an execution of Algorithm 1 with*

$$r \geq \min\{d, D\} + \min\{k, |S|\},$$

the truncation of $\mathtt{PL}_v^{(r)}$ to the first $\min\{k, \lambda_v^d\}$ entries, where λ_v^d is the number sources $s \in S$ such that $d(s, v) \leq d$, solves weighted (S, d, k)-detection problem.

This lemma says that the truncated list is correct at the beginning of the relevant round. To prove it, we use Lemma 7(ii) to show that the values in the truncated list cannot change at round r or later, and Lemma 5 to deduce they are correct.

Proof. Assume w.l.o.g that $d \leq D$, as D bounds the distance to any source, and $k \leq |S|$, as otherwise v needs to learn about all sources.

By Lemma 5, there is a round r_0 when all entries of $\mathtt{PL}_v^{(r_0)}$ are correct, and let (d_s, s, w_s) be a triplet in one of the first $\min\left\{k, \lambda_v^d\right\}$ entries of $\mathtt{PL}_v^{(r_0)}$. Since (d_s, s, w_s) is one of the first λ_v^d entries and $\mathtt{PL}_v^{(r_0)} = \mathtt{PL}_v^*$, we have $d_s \leq d$.

Let r be the round when (d_s, s, w_s) is inserted to the list $\mathtt{PL}_v$. By Lemma 7(ii), $r < d_s + \ell_v^{(r+1)}(d_s, s, w_s)$. By Lemma 6(i), when the triplet is inserted to the list, it is already placed in one of the first $\min\left\{k, \lambda_v^d\right\}$ entries, i.e., $\ell_v^{(r+1)}(d_s, s, w_s) \leq \min\left\{k, \lambda_v^d\right\} \leq k$. Hence,

$$r < d_s + \ell_v^{(r+1)}(d_s, s, w_s) \leq d + k.$$

Since this claim holds for any of the first $\min\left\{k, \lambda_v^d\right\}$ entries, these were all correct at the beginning of round $d + k$, and in all the succeeding rounds. $\square$

The construction of multiple WBFS trees is an instance of the $(S, D, |S|)$-detection problem. Lemma 8 shows that after $|S| + D - 1$ rounds of Algorithm 1 on such an instance, all the entries of the list $\mathtt{PL}_v^{(|S|+D)}$ are correct, yielding the main result of this section.

Theorem 1. *Given a weighted graph $G = (V, E, w)$ and a set of nodes $S \subseteq V$, there exists an algorithm for the* CONGEST *model that constructs a WBFS tree rooted at s, for every $s \in S$, in $|S| + D - 1$ rounds.*

4   A $(+6)$-Spanner Construction

In this section we discuss the distributed construction of $(+6)$-spanners. First, we present a template for constructing a $(+6)$-spanner and analyze the stretch and sparsity of the constructed spanner. Then, we provide an implementation of our template in the CONGEST model and analyze its running time.

A *cluster* C_i around a *cluster center* $c_i \in V$ is a subset of the set of neighbors of c_i in G. A node belonging to a cluster is *clustered*, while the other nodes are *unclustered*.

Our algorithm starts by randomly choosing cluster centers, and adding edges between them to their neighbors, where each neighbor arbitrarily chooses a single center to connect to. Then, additional edges are added, to connect each unclustered node to all its neighbors. Next, shortest paths between clusters are added to the spanner. In order to find these shortest paths in the CONGEST model, we use the WBFS construction algorithm to build WBFS trees from random sources. At the heart of our algorithm stands the path-hitting framework of Woodruff [52]: a shortest path in the graph which has many edges between clustered nodes, must go through many clusters. This fact is used in order to show that a path with many missing edges (edges not in H) is more likely to have an adjacent source of a WBFS tree, and thus it is well approximated by a path within the spanner.

Woodruff's algorithm starts with a similar clustering step. However, in order to add paths between clusters, it uses an involved subroutine that finds light almost-shortest paths between pairs of nodes. This subroutine seems too global to be implemented efficiently in a distributed setting, so in our construction it is replaced by only considering lightest shortest paths, which we do using the WBFS trees defined earlier.

Our algorithm constructs a $(+6)$-spanner with $O(n^{4/3} \log^{4/3} n)$ edges in $\tilde{O}(n^{2/3} + D)$ rounds, as stated next.

Theorem 2. *There exists an algorithm for the* CONGEST *model that constructs a $(+6)$-spanner with $O\left(n^{4/3} \log^{4/3} n\right)$ edges in $O\left(\frac{n^{2/3}}{\log^{1/3} n} + D\right)$ rounds and succeeds w.h.p.*

11

Lemmas 9 and 10 analyze the size and stretch of Algorithm `6AP` given below. The number of rounds of its distributed implementation is analyzed in Lemma 11, giving Theorem 2. We use $c > 2$ to denote a constant that can be chosen according to the desired exponent of $1/n$ in the failure probability.

Algorithm `6AP`

Input: a graph $G = (V, E)$, a constant $c > 2$;
Output: a subgraph H of G;
Initialization: $n \leftarrow |V|$; $H \leftarrow (V, \emptyset)$; $k \leftarrow 1$

Clustering. Pick each node as a *cluster center* w.p. $\frac{c}{n^{1/3} \log^{1/3} n}$, and denote the set of selected nodes by $\mathcal{C} = \{c_1, c_2, \ldots\}$. For each c_i, initialize a cluster $C_i \leftarrow \emptyset$.

For each node $v \in V$, choose a neighbor c_i of v which is a cluster center, if such a neighbor exists, add the edge (v, c_i) to H, and add v to C_i. If none of the neighbors of v is a cluster center, add to H all the edges adjacent to v. Let $H_0 \leftarrow H$.

Path Buying.
While $k \leq \frac{8cn^{2/3}}{\log^{1/3} n}$ do:

1. $S_k \leftarrow \emptyset$

2. Add each cluster center $c_i \in \mathcal{C}$ to S_k w.p. $\frac{8c^2 \log n}{k}$, independently of the other centers

3. For each pair $(c_i, c_j) \in \mathcal{C} \times S_k$:

   (a) $A \leftarrow \emptyset$ /* A is a set of paths */

   (b) For each $v \in C_j$:

       i. Among all the shortest paths from c_i to v, let P_v be a path with minimum $|P_v \setminus H_0|$

       ii. If $|P_v \setminus H_0| < 2k$, add P_v to A

   (c) If $A \neq \emptyset$, add to H one of the shortest among the paths of A

4. $k \leftarrow 2k$

Lemma 9. *Algorithm* `6AP` *outputs a subgraph H of G with $O(n^{4/3} \log^{4/3} n)$ edges, with probability at least $1 - O(n^{-c+1})$.*

Proof. The algorithm starts with $H = (V, \emptyset)$ and only adds edges from G, so H is indeed a subgraph of G over the same node set.

In the first part of the clustering phase, each node adds to H at most one edge, connecting it to a single cluster center, for a total of $O(n)$ edges. Then, the probability that a node of degree at least $n^{1/3} \log^{4/3} n$ is left unclustered is at most $\left(1 - \frac{c}{n^{1/3} \log^{1/3} n}\right)^{n^{1/3} \log^{4/3} n}$, which is $O(n^{-c})$. A union bound implies that all nodes of degree at least $n^{1/3} \log^{4/3} n$ are clustered w.p. $1 - O(n^{-c+1})$, and thus the total number of edges added to H by unclustered nodes in the second part of the clustering phase is $O(n^{4/3} \log^{4/3} n)$, w.p. $1 - O(n^{-c+1})$.

We start the analysis of the path buying phase by bounding the size of $\mathcal{C}$. A node $v \in V$ is added to $\mathcal{C}$ w.p. $\frac{c}{n^{1/3} \log^{1/3} n}$, so $\mathbb{E}[|\mathcal{C}|] = \frac{cn^{2/3}}{\log^{1/3} n}$. A Chernoff bound implies that

$$\Pr\left[|\mathcal{C}| > \frac{4cn^{2/3}}{\log^{1/3} n}\right] \leq \exp\left(-\frac{cn^{2/3}}{\log^{1/3} n}\right) = o(n^{-c}).$$
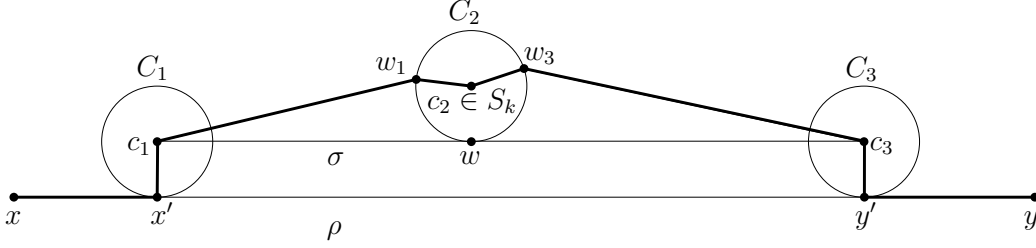
Figure 1: Illustration of the proof of Lemma 10

Similarly, for each value of k, we have $\mathbb{E}[|S_k|] = \frac{8c^2 n^{2/3} \log^{2/3} n}{k}$, and

$$\Pr\left[|S_k| > \frac{32c^2 n^{2/3} \log^{2/3} n}{k}\right] \leq \exp\left(-\frac{8c^2 n^{2/3} \log^{2/3} n}{k}\right) = O(n^{-c}),$$

where the last equality follows since $k \leq \frac{n^{2/3}}{\log^{1/3} n}$. A union bound implies that $|\mathcal{C}| = O\left(\frac{n^{2/3}}{\log^{1/3} n}\right)$ and $|S_k| = O\left(\frac{n^{2/3} \log^{2/3} n}{k}\right)$ for all k, w.p. at least $1 - O(n^{-c+1})$.

Finally, for each k, for each $(c_i, c_j) \in \mathcal{C} \times S_k$ we add at most one path with less than $2k$ missing edges to H. Thus, for each value of k we add less than $|\mathcal{C}| \cdot |S_k| \cdot 2k = O(n^{4/3} \log^{1/3} n)$ edges to H, w.p. at least $1 - O(n^{-c+1})$. Summing over all $O(\log n)$ values of k, and adding the number of edges contributed by the clustering phase, we conclude that H has at most $O(n^{4/3} \log^{4/3} n)$ edges, w.p. at least $1 - O(n^{-c+1})$. □

Lemma 10. *The graph H constructed by Algorithm* 6AP *satisfies $\delta_H(x, y) \leq \delta_G(x, y) + 6$ for each pair $(x, y) \in V \times V$, with probability at least $1 - O\left(n^{-c+2}\right)$.*

Proof. Consider a shortest path ρ in G between two nodes $x, y \in V$ (see Figure 1). Let x' and y' be the first and last clustered nodes on ρ, respectively. If all nodes of ρ are unclustered, then ρ is fully contained in H_0 and we are done.

Let c_1 and c_3 be the centers of the clusters containing x' and y', respectively. Let σ be a shortest path in G between c_1 and c_3, and denote by k' the number of edges of $\sigma \setminus H_0$. Let k be the largest power of 2 such that $k \leq k'$.

An edge can be in $\sigma \setminus H_0$ only if it connects two clustered nodes. Hence, k', the number of edges in $\sigma \setminus H_0$, is smaller than the number of clustered nodes in σ. On the other hand, σ cannot contain more than three nodes of the same cluster: the distance between every two nodes in a cluster is at most two, so a shortest path cannot traverse more than three nodes of the same cluster. Thus, the number of clusters intersecting σ is at least $k'/3$. As $k'/3 \geq k/3$, the probability that none of the centers of these clusters is chosen to S_k is at most $\left(1 - \frac{8c^2 \log n}{k}\right)^{k/3} = O\left(n^{-c^2}\right)$. For each pair of nodes, a cluster center on a shortest path between them is chosen to S_k, for the appropriate value of k, with similar probability. A union bound implies that this claim holds for all pairs in $V \times V$ w.p. at least $1 - O(n^{-c^2+2})$.

Let w be a node on σ in a cluster C_2 such that $c_2 \in S_k$, if such a cluster exists. Denote by $\sigma[c_1, w]$ the sub-path of σ from c_1 to w. As there are $k' < 2k$ edges in $\sigma \setminus H_0$, there are also less than $2k$ edges in $\sigma[c_1, w] \setminus H_0$. Thus, in step 3($b$) of the path-buying phase for k, either the path $\sigma[c_1, w]$ or some other path between c_1 and w of length at most $\delta_G(c_1, w)$ is added to A. In step 3(c), a path from c_1 to some node $w_1 \in C_2$ is added to H, and this is a shortest path in A, so $\delta_H(c_1, w_1) \leq \delta_G(c_1, w)$. Similarly, a shortest path from c_3 to some $w_3 \in C_2$ is added to H, and $\delta_H(c_3, w_3) \leq \delta_G(c_3, w)$.

13

The path σ is a shortest path from c_1 to c_3 in G, so $|\sigma| \leq \delta_G(x', y') + 2$. As $\delta_G(c_1, w) + \delta_G(c_3, w) = |\sigma|$, we conclude $\delta_H(c_1, w_1) + \delta_H(c_3, w_3) \leq |\sigma| \leq \delta_G(x', y') + 2$.

Consider the path from x to y in H composed of the sub-path of ρ from x to x', the edge (x', c_1), the path from c_1 to w_1, the edges (w_1, c_2) and (c_2, w_3), the path from w_3 to c_3, the edge (c_3, y'), and finally, the sub-path of ρ from y' to y. This is a path from x to y in H, implying

$$\delta_H(x, y) \leq \delta_H(x, x') + 1 + \delta_H(c_1, w_1) + 2 + \delta_H(w_3, c_3) + 1 + \delta_H(y', y)$$
$$\leq \delta_G(x, x') + 4 + \delta_G(x', y') + 2 + \delta_G(y', y) = \delta_G(x, y) + 6,$$

as desired. $\qquad\square$

We now discuss the implementation of Algorithm 6AP in the CONGEST model.

Lemma 11. *Algorithm* 6AP *can be implemented in* $O\left(\frac{n^{2/3}}{\log^{1/3} n} + D\right)$ *rounds in the* CONGEST *model, with probability at least* $1 - o(n^{-c})$.

Proof. For the clustering phase, each node decides locally w.p. $\frac{c}{n^{1/3} \log^{1/3} n}$ to become a cluster center, and notifies its neighbors. Each node with a neighbor that is a cluster center now joins a cluster by sending a message to such a neighbor and adding the appropreate edge to the spanner. A node with no neighboring cluster centers notifies all its neighbors and adds all its edges to the spanner. This is done in a constant number of rounds.

Before the path buying phase, the nodes construct a single BFS tree, along which they compute an upper bound D' on D, satisfying $D \leq D' < 2D$, and count the number of cluster centers, $|\mathcal{C}|$. The nodes mark the edges of H_0 with weight 0 and the other edges with weight 1. Then, they construct a WBFS tree rooted at each cluster center by executing Algorithm 1 for $|\mathcal{C}| + D'$ many rounds. By the proof of Lemma 9, we have $|\mathcal{C}| \in O\left(\frac{n^{2/3}}{\log^{1/3} n}\right)$ w.p. at least $1 - o(n^{-c})$, and thus the construction of the WBFS trees takes $O\left(\frac{n^{2/3}}{\log^{1/3} n} + D\right)$ rounds with the same probability.

Each node v now knows about a "good" path to each cluster center c_i, i.e., a shortest path from c_i to v, with a minimal number of edges not in H after the clustering phase. A node v in a cluster C_j notifies its neighbor c_j about all the distances to other cluster centers in $\mathcal{C}$ and the number of missing edges in each such path. That is, each $v \in C_j$ sends $|\mathcal{C}|$ messages to c_j, which takes $O\left(\frac{n^{2/3}}{\log^{1/3} n}\right)$ rounds.

Each cluster center c_j decides locally to join each set S_k w.p. $\frac{8c^2 \log n}{k}$. For each other center $c_i \in \mathcal{C}$, c_j locally constructs the list A: for each $v \in C_j$, A contains the shortest path from c_i to $v \in C_j$ found by the WBFS algorithm, and the number of missing edges in it. Then, c_j chooses from A a path from c_i to some $v \in C_j$ with a minimal number of missing edges, and if it has at most $2k$ missing edges, c_j sends a "buy c_i" message to v.

Finally, all nodes simultaneously execute a "buy" phase, where "buy c_i" messages are sent up the WBFS tree. To avoid congestion, we assume that during the execution of Algorithm 1, each node keeps a record of the messages it got in each round and the WBFS source each message referred to. Each node v then sends messages in reversed order: if v has a message "buy c_i", and it got a message from u regarding c_i in the r-before-last round of Algorithm 1, then it sends the message "buy c_i" to u in round r of the "buy" phase. Then, u adds "buy c_i" to its list of messages, and adds the edge (u, v) to the spanner. This parts takes $O\left(\frac{n^{2/3}}{\log^{1/3} n} + D\right)$ rounds, just like the execution of Algorithm 1. $\qquad\square$

5 Discussion and Open Questions

While we present an application of WBFS trees, our algorithm also solves the weighted (S, d, k)-detection problem, a result that could be of independent interest.

The question of finding the *lightest* paths between all pairs of nodes in a graph, or computing their weights, is a fundamental question in many computational models. In the CONGEST model, a randomized algorithm for exactly computing these distances in $\tilde{O}(n)$ rounds was recently presented [10]. The exact time complexity of computing these distances *deterministically* in the CONGEST model is still open [4], and we hope our study of lightest shortest paths could facilitate future research on it.

While this paper settles the question of constructing sparse (+6)-spanners fast, the study of spanner construction in distributed environments still lags behind the study of sequential spanner construction algorithms. In the field of purely additive spanners, we still do not have fast algorithms, e.g., for the construction of sparse (+0)-pairwise spanners (a.k.a. pairwise preservers) and (+6)-pairwise spanners.

A more intriguing question is proving time lower bounds for the construction of spanners in the CONGEST model: while $\Omega(D)$ rounds are known to be necessary [49], lower bounds that depend on other parameters of the graph or the spanners exist only for pairwise spanners [13]. Finding a lower bound for the construction of all-pairs spanners in the CONGEST model, which does not depend on D, is still an open question. Such a lower bound could show that the $n^{3/2}$ term in the time bound of our construction is inevitable, or motivate the design of faster algorithms for the problem.

References

[1] Amir Abboud and Greg Bodwin. Error amplification for pairwise spanner lower bounds. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 841–854, 2016.

[2] Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017.

[3] Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *30th International Symposium on Distributed Computing, DISC*, pages 29–42, 2016.

[4] Udit Agarwal and Vijaya Ramachandran. Faster deterministic all pairs shortest paths in congest model. *CoRR*, abs/2005.09588, 2020. To appear in SPAA 2020.

[5] Udit Agarwal, Vijaya Ramachandran, Valerie King, and Matteo Pontecorvi. A deterministic distributed algorithm for exact weighted all-pairs shortest paths in $\tilde{O}(n^{3/2})$ rounds. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 199–205, 2018.

[6] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems. In *ACM Symposium on Theory of Computing, STOC*, pages 230–240, 1987.

[7] Leonid Barenboim, Michael Elkin, and Cyril Gavoille. A fast network-decomposition algorithm and its applications to constant-time distributed computation. *Theor. Comput. Sci.*, 751:2–23, 2018.

[8] Surender Baswana, Telikepalli Kavitha, Kurt Mehlhorn, and Seth Pettie. Additive spanners and (alpha, beta)-spanners. *ACM Trans. Algorithms*, 7(1):5, 2010.

[9] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algor.*, 30(4):532–563, 2007.

[10] Aaron Bernstein and Danupon Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 334–342, 2019.

[11] Keren Censor-Hillel and Michal Dory. Distributed spanner approximation. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 139–148, 2018.

[12] Keren Censor-Hillel, Bernhard Haeupler, Jonathan A. Kelner, and Petar Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In *44th Symposium on Theory of Computing Conference, STOC*, pages 961–970, 2012.

[13] Keren Censor-Hillel, Telikepalli Kavitha, Ami Paz, and Amir Yehudayoff. Distributed construction of purely additive spanners. *Distributed Computing*, 31(3):223–240, 2018.

[14] Shiri Chechik. Compact routing schemes with improved stretch. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 33–41, 2013.

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

[16] Bilel Derbel and Cyril Gavoille. Fast deterministic distributed algorithms for sparse spanners. *Theor. Comput. Sci.*, 399(1-2):83–100, 2008.

[17] Bilel Derbel, Cyril Gavoille, and David Peleg. Deterministic distributed construction of linear stretch spanners in polylogarithmic time. In *21st International Symposium on Distributed Computing, DISC*, pages 179–192, 2007.

[18] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *27th Annual ACM Symposium on Principles of Distributed Computing, PODC*, pages 273–282, 2008.

[19] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. Local computation of nearly additive spanners. In *23rd International Symposium on Distributed Computing, DISC*, pages 176–190, 2009.

[20] Devdatt P. Dubhashi, Alessandro Mei, Alessandro Panconesi, Jaikumar Radhakrishnan, and Aravind Srinivasan. Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons. *J. Comput. Syst. Sci.*, 71(4):467–479, 2005.

[21] Michael Elkin. Computing almost shortest paths. *ACM Trans. Algorithms*, 1(2):283–323, 2005.

[22] Michael Elkin. Distributed exact shortest paths in sublinear time. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 757–770, 2017.

[23] Michael Elkin and Shaked Matar. Near-additive spanners in low polynomial deterministic CONGEST time. *CoRR*, abs/1903.00872, 2019.

[24] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 128–137, 2016.

[25] Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2019.

[26] Michael Elkin and Jian Zhang. Efficient algorithms for constructing $(1 + \epsilon, \beta)$-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

[27] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[28] Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In *32nd International Symposium on Distributed Computing, DISC*, pages 29:1–29:17, 2018.

[29] Mohsen Ghaffari and Jason Li. Improved distributed algorithms for exact shortest paths. In *SIGACT Symposium on Theory of Computing, STOC*, pages 431–444, 2018.

[30] Ofer Grossman and Merav Parter. Improved deterministic distributed construction of spanners. In *31st International Symposium on Distributed Computing, DISC*, pages 24:1–24:16, 2017.

[31] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *ACM SIGACT Symposium on Theory of Computing, STOC*, pages 489–498, 2016.

[32] Stephan Holzer, David Peleg, Liam Roditty, and Roger Wattenhofer. Distributed 3/2-approximation of the diameter. In *28th International Symposium on Distributed Computing, DISC*, pages 562–564, 2014.

[33] Stephan Holzer and Roger Wattenhofer. Personal communication.

[34] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 355–364, 2012.

[35] Chien-Chung Huang, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed exact weighted all-pairs shortest paths in $\tilde{O}(n^{5/4})$ rounds. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 168–179, 2017.

[36] Telikepalli Kavitha. New pairwise spanners. *Theory Comput. Syst.*, 61(4):1011–1036, 2017.

[37] Mathias Bæk Tejs Knudsen. Additive spanners: A simple construction. In *14th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, pages 277–281, 2014.

[38] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Computing*, 32(2):133–157, 2019.

[39] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 375–382, 2013.

[40] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *J. ACM*, 62(5):38:1–38:17, 2015.

[41] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[42] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Symposium on Theory of Computing, STOC*, pages 565–573, 2014.

[43] Merav Parter. Vertex fault tolerant additive spanners. *Distributed Computing*, 30(5):357–372, 2017.

[44] Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In *DISC*, volume 121 of *LIPIcs*, pages 40:1–40:18, 2018.

[45] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2000.

[46] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[47] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

[48] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[49] Seth Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3):147–166, 2010.

[50] Matteo Pontecorvi and Vijaya Ramachandran. Distributed algorithms for directed betweenness centrality and all pairs shortest paths. *CoRR*, abs/1805.08124, 2018.

[51] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *ACM Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 1–10, 2001.

[52] David P. Woodruff. Additive spanners in nearly quadratic time. In *37th International Colloquium on Automata, Languages and Programming, ICALP*, pages 463–474, 2010.