Distrib. Comput. (2015) 28:309–320
DOI 10.1007/s00446-015-0246-7

Efficient distributed computation of distance sketches in networks

Atish Das Sarma[1] · Michael Dinitz[2] · Gopal Pandurangan[3]

Received: 20 November 2012 / Accepted: 24 April 2015 / Published online: 8 May 2015

Abstract Distance computation (e.g., computing shortest paths) is one of the most fundamental primitives used in communication networks. The cost of effectively and accurately computing pairwise network distances can become prohibitive in large-scale networks such as the Internet and Peer-to-Peer (P2P) networks. The idea behind distance sketches is to preprocess the graph and store a *small* amount of information (called as *sketch* or *distance label*) such that whenever a query for any pairwise distance is issued, the distance can be well approximated (i.e., with small *stretch*) very quickly in an online fashion. Specifically, the pre-processing (usually) involves storing this sketch with each node, such that at query time only the sketches of the concerned nodes need to be looked up to compute the approximate distance. In this paper, we present the first theoretical study of distance sketches derived from distance oracles in a distributed network. We first present a fast distributed algorithm for computing approximate distance sketches, based on a distributed implementation of the distance oracle scheme of (Thorup and Zwick, J ACM 52(1):1–24, 2005). We also show how to modify this basic construction to achieve different tradeoffs between the number of pairs for which the distance estimate is accurate, the size of the sketches, and the time and message complexity necessary to compute them. These tradeoffs can then be combined to give an efficient construction of small sketches with provable average-case as well as worst-case performance. Our algorithms use only small-sized messages and hence are suitable for bandwidth-constrained networks, and can be used in various networking applications such as topology discovery and construction, token management, load balancing, monitoring overlays, and several other problems in distributed algorithms.

Keywords Distributed network · Shortest paths · Distance labeling · Distributed algorithm

A preliminary version of this paper appeared in the proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Pittsburgh, PA, USA, 2012.

Michael Dinitz: Work supported in part by an Israel Science Foundation Grant #452/08, a US-Israel BSF Grant #2010418, and by a Minerva grant.
Gopal Pandurangan: Work done when the author was at the Division of Mathematical Sciences, Nanyang Technological University, Singapore 637371, and Department of Computer Science, Brown University, Providence, RI, 02912, USA. Research supported in part by the following grants: Nanyang Technological University Grant M58110000, Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 Grant MOE2010-T2-2-082, and the United States-Israel Binational Science Foundation Grant 2008348.

✉ Gopal Pandurangan
gopalpandurangan@gmail.com

Atish Das Sarma
atish.dassarma@gmail.com

Michael Dinitz
mdinitz@cs.jhu.edu

1 eBay Research Labs, San Jose, CA, USA

2 Johns Hopkins University, Baltimore, MD, USA

3 Department of Computer Science, University of Houston, Houston, TX 77204, USA

1 Introduction

A fundamental operation on large networks is finding shortest paths between pairs of nodes, or at least finding the lengths of these shortest paths. This problem is not only a common building block in many algorithms, but is also a meaningful operation in its own right. In a social network one may be interested in finding the shortest sequence of friends that connects one to a celebrity. In the web graph, a short sequence of links between two URLs may indicate a certain degree of relatedness between the two pages [14,39]. In a distributed network such as a large peer-to-peer network, this may be useful in search, topology discovery, overlay creation, and basic node to node communication. However, given the large size of these networks, computing shortest path distances can, if done naively, require a significant amount of both time and network resources. Moreover, running Dijkstra's well known shortest-path algorithm [9] on a large network (e.g., Web graph) containing tens of billions of nodes and trillions of edges would take several hours, if not days. Even in a distributed setting, if the computation is parallelized, the sequentially dependent nature of Dijkstra's computation would require huge amounts of communication. Furthermore, in a real-time computation, only a small amount of resources—memory access, CPU cycles—are available for a single shortest distance query. As we would like to make distance queries in real time with minimal latency, it becomes important to use small amounts of resources per distance query. Particular to distributed networks, storing some approximate distances or network views from nodes are important as they facilitate understanding the overall topology from a local view of each node. In particular, such *distance "sketches"* (also known as *distance labelings*) of particular importance in the Internet or P2P networks where it is often critical to understand the network topology. Sketches present an ideal way for each node to store its view of the overall global structure of the distributed network. Since distance information can change dynamically, it is important to compute the sketches quickly and also in an online fashion.

One approach to handle online distance requests is to perform a one-time offline or centralized computation. A straightforward brute force solution would be to compute the shortest paths between all pairs of nodes offline and to store the distances locally in the nodes. Once this has been accomplished, answering a shortest-path query online can be done with no communication overhead; however, the local space requirement is quadratic in the number of nodes in the graph (or linear if only shortest paths from the node are stored). For a large network containing millions of nodes, this is simply infeasible. An alternative, more practical approach is to store some auxiliary information with each node that can facilitate a quick distance computation online in real time. This auxiliary information is then used in the online computation that is performed for every request or query. One can view this auxiliary information as a *sketch* of the neighborhood structure of a node that is stored with each node. Simply retrieving the sketches of the two nodes should be sufficient to estimate the distance between them. Three properties are crucial for this purpose: first, these sketches should be reasonably small in size so that they can be stored with each node and accessed for any node at run time. Second, there needs to be a simple algorithm that, given the sketches of two nodes, can estimate the distance between them quickly. And third, even though the computation of the sketches is an offline computation, this cost also needs to be accounted for (as the distance information or network itself changes frequently, and this would require altering the sketches periodically).

Sketches for the specific purpose of distance computation in communication networks have been referred to as *distance labelings* (by the more theoretical literature) and as *network coordinate systems* (by the more applied literature). There has been a significant amount of work from a more theoretical point of view on the fundamental tradeoff between the size of the sketches and the accuracy of the distance estimates they give (see e.g. Thorup and Zwick [37], Gavoille et al. [16], Katz et al. [21], and Cohen et al. [6]). However, all of these papers assumed a *centralized* computation of sketches, so are of limited utility in real distributed systems. In the networking community there has also been much work on constructing good network coordinate systems, including seminal work such as the Vivaldi system [10] and the Meridian system [38]. While this line of work has resulted in almost fully functioning systems with efficient distributed algorithms, the theoretical underpinning of such systems is lacking; most of them can easily be shown to exhibit poor behavior in pathological instances. The main exception to this is Meridian, which has a significant theoretical component. However, it assumes that the underlying metric space is "low-dimensional", and it is easy to construct high-dimensional instances on which Meridian does poorly.

We attempt to move the theoretical line of research closer to practice by designing efficient distributed algorithms for computing accurate distance sketches. We give algorithms with bounded round and message complexity in a standard model of distributed computation (the CONGEST model [33]) that compute sketches that give distances estimates provably close to the actual distances. In particular, we engineer a distributed version of the seminal centralized algorithm of Thorup and Zwick [37]. Their algorithm computes sketches that allow us to approximate distances within a factor $2k - 1$ (this value is known as the *stretch*) by using sketches of size $\tilde{O}(k \cdot n^{1/k})$, for any integer $k \geq 1$. Up to the factor of k in the size, this is known to be a tight tradeoff in the worst case assuming a famous conjecture by Erdős [37]. Note that this achieves its minimum size at $k = \log n$, giving an $O(\log n)$-factor approximation to the distances using sketches of size $O(\log^2 n)$. We further extend these results to give a distributed algorithm based on the centralized algorithms of Chan et al. [5] that computes sketches with the same worst-case stretch and almost the same size, but with provably better *average* stretch. To the best of our knowledge, this is the first theoretical analysis of distributed algorithms for computing distance sketches. Our work can also be viewed as an efficient computation of local node-centric views of the global topology, which may be of independent interest for numerous different applications (cf. Sect. 2.1).

1.1 Our contributions

Our main contributions are new distributed algorithms for various types of distance sketches i.e., distance labeling schemes (which is more commonly used terminology in the distributed computing literature.[1] Most of the actual sketches have been described in previous work (in particular [37] and [5]), but we are the first to show that they can be efficiently constructed in a distributed network. While we formally define the model and problem in Sect. 2.2, at a high level we assume a synchronous distributed network in which in a single communication "round" every node can send a message of up to $O(\log n)$ bits (or 1 word) to each of its neighbors. Every edge has a nonnegative weight associated with it, and the distance between two nodes is the total weight of the shortest path (with respect to weights) between them. An important parameter (defined formally in Sect. 2.2) is S, the *shortest-path diameter* [24], which is (informally) the maximum over all $\binom{n}{2}$ pairs of the minimum number of hops in any shortest path (where "short" is determined by weight). In an unweighted network, S is the same as the network diameter D, hence S can be thought of as a generalization of D in a weighted network. Note that D is a time lower bound on any exact distance computation.

We begin in Sect. 3 by giving a distributed algorithm that constructs the distance sketches of Thorup and Zwick [37] efficiently:

Theorem 1 *For any integer $k \geq 1$, there is a distributed algorithm that takes $O(kn^{1/k} S \log n)$ rounds after which with high probability every node has a sketch of size at most $O(kn^{1/k} \log n)$ words that provides approximate distances up to a factor of $2k-1$.*

In Sect. 4 we show how to extend the techniques of Chan et al. [5] and combine them with the techniques of Sect. 3 to give sketches with "slack". Informally, a sketch has ε-slack if the stretch factor (i.e. the distance approximation guarantee) only holds for a $(1-\varepsilon)$-fraction of the pairs, rather than all pairs. While this is a weaker guarantee, since some pairs have no bound on the accuracy of the distance estimate at all, both the size of the sketches and the time needed to construct them become much smaller. For example, when ε is a constant (even a small constant) we can construct constant stretch sketches in only $O(S \log^2 n)$ rounds.

[1] We simply use the terminology "sketch" throughout this paper, if only for the sake of brevity).

Theorem 2 *For any value $\varepsilon > 0$ and integer $1 \leq k \leq O(\log \frac{1}{\varepsilon})$, there is a distributed sketching algorithm that with high probability gives sketches with stretch $8k-1$ with ε-slack that have size at most $O(k\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n)$ words and completes in at most $O\left(kS\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n\right)$ rounds.*

Finally, in Sect. 4.1 we extend the slack techniques further by combining a hierarchy of sketches with different slack parameters. This allows us to efficiently construct sketches with the same worst-case stretch as in Theorem 1 (with $k = O(\log n)$) but with average stretch only $O(1)$:

Theorem 3 *There is a distributed sketching algorithm that with high probability gives sketches of size at most $O(\log^4 n)$ with $O(\log n)$-stretch and O(1) average stretch and completes in at most $O(S \log^4 n)$ rounds.*

2 Related work and model

2.1 Applications and related work

Applications of approximate distance computations in distributed networks include token management [8,19], load balancing [20], small-world routing [26], and search [7, 17,29]. Several other areas of distributed computing also use distance computations in a crucial way; some examples are information propagation and gathering [4,22], network topology construction [17,27,28], monitoring overlays [30], group communication in ad-hoc network [12], gathering and dissemination of information over a network [2], and peer-to-peer membership management [15,40].

The most concrete application of our algorithms is to quickly computing approximate shortest path distances in networks, i.e. the normal application of network coordinate systems. In particular, in weighted networks, after using our algorithms to preprocess the network and create distance sketches we can compute the approximate distance between any two nodes in at most $O(D)$ times the size of the sketch rounds (where D, the hop-diameter, is the maximum over all pairs of nodes of the minimum number of hops between the nodes) by simply exchanging the sketches of the two nodes. On the other hand, note that any distance computation without using preprocessing (say, Dijkstra's algorithm, Bellman-Ford, or even a simple network ping to obtain the round-trip time) will take at least $\Omega(S)$ rounds, where S is the shortest path diameter. This is less than ideal since S can be as large as n even when D (the hop-diameter) is much smaller. Therefore our sketches yield improved algorithms for pairwise weighted distance computations. Moreover, in networks such as P2P networks and overlay networks, using our algorithms a node can compute distances (number of hops in the overlay) in *constant* times size of sketch rounds if it simply knows the IP address of the other node: it can directly contact the other node using its IP address and ask for its sketch. Thus these sketch techniques can be very relevant and applicable even for unweighted distance computations.

Probably the closest results to ours are from the theory behind Meridian [38], which is based on a modification of the "ring-of-neighbors" theoretical framework developed by Slivkins [35,36] to prove theoretical bounds. However, there are some substantial differences. For one, the bounds given by these papers (including [38]) are limited to special types of metric spaces: those with either bounded doubling dimension or bounded growth. Our bounds hold for all weighted graphs (with weaker guarantees on the stretch, obviously). Furthermore, the distributed framework used by Slivkins is significantly different from the standard CONGEST model of distributed computation that we use, and is based on being able to work in the metric completion of the graph. This means, for example, that the algorithms in [36] have the ability to send a unit-size packet between *any* two nodes in $O(1)$ time (and the algorithms do in fact make strong use of this ability).

Several studies, for example the ones by Goldberg et al. [11,18], have focused on answering exact shortest path queries on road networks. These algorithms make use of a small set of precomputed landmarks and shortcuts and use them at query time to connect a source, destination pair; these landmarks and shortcuts are chosen very carefully using algorithms that are specialized to the structure of road networks. This approach has also proven to be useful in the context of network distances, though, and has been used heavily in several constructions of network location services (e.g. the Meridian system [38] and the theory that it is based on [25]). Our algorithms can be viewed in this framework as using a randomly sampled set of landmarks.

Another line of work in estimating distances is the study of *spanner* construction. A spanner is a sparse subgraph of the given graph, such that the distance on this sparse graph approximates the actual distance, for any pair of points. Although spanners take small space, they do not exactly provide a sketch for each node; thus the online algorithm for estimating distance may take a long time. Some theoretically efficient algorithms for spanners are presented by Feigenbaum et al. [13], and Baswana [3] and the references therein.

2.2 Model and notation

We model a communication network as a weighted, undirected, connected n-node graph $G = (V, E)$. Every node has limited initial knowledge. Specifically, assume that each node is associated with a distinct identity number (e.g., its IP address). At the beginning of the computation, each node v accepts as input its own identity number and the identity numbers of its neighbors in G. The node may also accept some additional inputs as specified by the problem at hand. The nodes are allowed to communicate through the edges of the graph G. We assume that the communication occurs in synchronous *rounds*. We will use only small-sized messages. In particular, in each round, each node v is allowed to send a message of size $O(\log n)$ through each edge $e = (v, u)$ that is adjacent to v. The message will arrive at u at the end of the current round. We also assume that all edge weights are at most polynomial in n, and thus in a single round a distance or node ID can be sent through each edge. A *word* is a block of $O(\log n)$ bits that is sufficient to store either a node ID or a network distance.

This is a widely used standard model to study distributed algorithms (called the *CONGEST model*, e.g., see [32,33]) and captures the bandwidth constraints inherent in real-world computer networks. Many classical network algorithms have been studied in this model, including algorithms for shortest paths (Bellman-Ford, Dijsktra), minimum spanning trees, etc. Our algorithms can be easily generalized if B bits are allowed (for any pre-specified parameter B) to be sent through each edge in a round. Typically, as assumed here, $B = O(\log n)$, which is number of bits needed to send a node id in an n-node network. We assume that n (or some constant factor estimate of n) is common knowledge among nodes in the network.

Every edge in the network has some nonnegative weight associated with it, and the distance between two nodes is the minimum, over all paths between the nodes, of the sum of the weights of the edges on the path. In other words, the normal shortest-path distance with edge weights. We let $d(u, v)$ denote this distance for all $u, v \in V$. For a set $A \subseteq V$ and a node $u \in V$, we define the distance from the node to the set to be $d(u, A) = \min\{d(u, a) : a \in A\}$. For a node $u \in V$ and real number $r \in \mathbb{R}^{\geq 0}$, the *ball* around u of radius r is defined to be the nodes within distance r of u, namely $B(u, r) = \{v \in V : d(u, v) \leq r\}$.

The *hop-diameter* D of G is defined to be the maximum over all pairs u,v in V of the number of hops between u and v. In other words, its the maximum over all pairs of the distance between u and v but where distance is computed assuming that all edge weights are 1, rather than their actual weights. The *shortest-path diameter* S of G is slightly more complicated to define. For $u, v \in V$, let $\mathcal{P}_{u,v}$ be the set of simple paths between u and v with total weight equal to $d(u, v)$ (by the definition of $d(u, v)$ there is at least one such path). Let $h(u, v)$ be the minimum, over all paths in $P \in \mathcal{P}_{u,v}$, of the number of hops in P (i.e. the number of edges). Then $S = \max_{u,v \in V} h(u, v)$. It is easy to see that $D \leq S$ and in general, any method of computing the distance from u to v must use at least S rounds (or else the shortest path will not be discovered).

3 Distributed sketches

We are concerned with the problem of constructing a distance labeling scheme in a distributed manner. Given an input (weighted) graph $G = (V, E)$, we want a distributed algorithm so that at termination every node $u \in V$ knows a small label (or sketch) $L(u)$ with the property that we can (quickly) compute an approximation to the distance between u and v just from $L(u)$ and $L(v)$. Since the requirement of sketch sizes and latency in distance computation may vary from application to application, typically one would like a trade-off between the distance approximation and these parameters.

3.1 Thorup–Zwick construction

The famous algorithm for constructing distance sketches in a centralized manner is Thorup-Zwick [37], which works as follows. They first create a hierarchy of node sets: $A_0 = V$, and for $1 \le i \le k-1$, we get A_i by randomly sampling every vertex in A_{i-1} with probability $n^{-1/k}$, i.e. every vertex in A_{i-1} is included in A_i with probability $n^{-1/k}$. We set $A_k = \emptyset$ and $d(u, A_k) = \infty$ by definition.

Let $B_i(u) = \{w \in A_i : d(u, w) < d(u, A_{i+1})\}$, and let $B(u) = \cup_{i=0}^{k-1} B_i(u)$ (for now we will assume that all distances are distinct; this can be made without loss of generality by breaking ties consistently through processor IDs or some other method). $B(u)$ is called the *bunch* of u. Let $p_i(u)$ be the vertex in A_i with minimum distance from u. The label $L(u)$ of u consists of all nodes $\{p_i(u)\}_{i=0}^{k-1}$ and $B(u)$, as well as the distances to all of these nodes. Thorup and Zwick showed that these labels are enough to approximately compute the distance, and that these labels are small. We give sketches of these proofs for completeness (the reader can safely skip the proofs of these without affecting the continuity).

Lemma 1 ([37]) *With high probability the size of $B_i(u)$ is at most $O(n^{1/k} \log n)$ for all $u \in V$ and $i \in \{0, \ldots, k-1\}$. This means that the size of $L(u)$ is at most $O(kn^{1/k} \log n)$ words with high probability.*

Proof We first prove that the expected size of $B_i(u)$ is at most $n^{1/k}$ for every $0 \le i \le k-1$. Suppose that we have already made the random decisions that define levels $A_0, \ldots, A_i$, and now for each $v \in A_i$ we flip the coin to see if it is also in A_{i+1}. If we flip these coins in order of distance from u (this is just in the analysis; the algorithm can flip the coins simultaneously or in arbitrary order) then the size of $B_i(u)$ is just the number of coins we flip before we see a heads, where the probability of flipping a heads is $n^{-1/k}$. In expectation this is $n^{1/k}$. Now a Chernoff bound guarantees that $|B_i(u)|$ is at most $O(n^{1/k} \log n)$ with high probability, and we can take a union bound over all u and all i. □

Lemma 2 ([37]) *Given $L(u)$ and $L(v)$ for some $u, v \in V$, we can compute a distance estimate $d'(u, v)$ with $d(u, v) \le d'(u, v) \le (2k-1)d(u, v)$ in time $O(k)$.*

Proof For each $0 \le i \le k-1$, we check whether $p_i(u) \in B_i(v)$ or $p_i(v) \in B_i(u)$. Let i^* be the first level at which at least one of these events occurs. Note that i^* is well-defined and is at most $k-1$, since $p_{k-1}(u) \in B_{k-1}(v)$ and $p_{k-1}(v) \in B_{k-1}(u)$. If the first condition is true then we return the distance estimate $d'(u, v) = d(u, p_{i^*}(u)) + d(v, p_{i^*}(u))$, and if the second condition is true we return $d'(u, v) = d(u, p_{i^*}(v)) + d(v, p_{i^*}(v))$. Note that the necessary distances are in the labels as part of $B_{i^*}(u)$ and $B_{i^*}(v)$, so we can indeed compute this from $L(u)$ and $L(v)$.

We first prove by induction that $d(u, p_i(u)) \le i \cdot d(u, v)$ and $d(v, p_i(v)) \le i \cdot d(u, v)$ for all $i \le i^*$. In the base case, when $i = 0$, both inequalities are true by definition. For the inductive step, let $1 \le i \le i^*$. Since $i \le i^*$ we know that $i - 1 < i^*$, so $p_{i-1}(u) \notin B_{i-1}(v)$ and $p_{i-1}(v) \notin B_{i-1}(u)$. This implies that $d(v, p_i(v)) \le d(v, p_{i-1}(u)) \le d(v, u) + d(u, p_{i-1}(u) \le d(u, v) + (i-1)d(u, v) = i \cdot d(u, v)$, where the first inequality is from $i - 1 < i^*$, the second is from the triangle inequality, and the third is from the inductive hypothesis. Similarly, we get that and $d(u, p_i(u)) \le i \cdot d(u, v)$.

Now suppose without loss of generality that $p_{i^*}(v) \in B_{i^*}(u)$ (if the roles are reversed we can just switch the names of u and v). Then our distance estimate is

$$\begin{aligned} d'(u, v) &= d(u, p_{i^*}(v)) + d(v, p_{i^*}(v)) \\ &\le d(u, v) + 2d(v, p_{i^*}(v)) \\ &\le d(u, v) + 2i^* d(u, v) = (2i^* + 1)d(u, v), \end{aligned}$$

where the first inequality is from the triangle inequality and the second is from our previous inductive proof. Since $i^* \le k-1$, this gives a stretch bound of $2k-1$ as claimed. □

3.2 Distributed algorithm

The natural question is whether we can construct these labels in a distributed manner. For a vertex $v \in A_i \backslash A_{i+1}$, let $C(v) = \{w \in V : d(w, v) < d(w, A_{i+1})\}$. This is called the *cluster* of v. Note that the clusters are the inverse of the bunches: $u \in C(v)$ if and only if $v \in B(u)$. So we will construct a distributed algorithm in which every vertex u knows exactly which clusters it is in and its distance from the centers of those clusters, and thus is able to construct its label. Also, it's easy to see that the clusters are connected: if $u \in C(v)$ then obviously any vertex w on the shortest path from u to v is also in $C(v)$.

The distributed protocol is as follows: we first divide into k phases, where in phase i we deal with clusters from vertices in $A_i \backslash A_{i+1}$. However, we run the phases from top to bottom – we first do phase $k-1$, then phase $k-2$, down to phase 0.

In phase i the goal is for every node $u \in V$ to know for every node $v \in V$ whether $u \in C(v)$, and if so, its distance to v. Thus every node u will know $B_i(u)$ at the end of phase i. We will give an upper bound on the length of each phase with respect to n and S, so if every node knows both n and S they can all start each phase together by waiting until the upper bound is met. For now we will make the assumption that every node knows S (the shortest path diameter), thus solving the issue of synchronizing the beginning of the phases, but we will show in Sect. 3.3 how to remove this assumption.

Let us first consider phase $k-1$, i.e. the first phase that is run. This phase is especially simple, since by definition $B_{k-1}(u) = A_{k-1}$ for every node u. So at the end of this phase we simply want every node to know all of the nodes in A_{k-1} and its distances to all of them. This is known as the *k-Source Shortest Paths Problem*, and can be done in $O(|A_{k-1}|S)$ rounds by running distributed Bellman-Ford from each node in A_{k-1} simultaneously [32]. In particular, for a fixed source $v \in A_{k-1}$ every node $u \in V$ runs the following protocol: initially, u guesses that its distance to v is $d'(u, v) = \infty$. If it hears a message from a neighbor w that contains a distance $a(w)$, then it checks if $d(u, w) + a(w) < d'(u, v)$. If so, then it updates $d'(u, v)$ to $d(u, w) + a(w)$ and sends to all its neighbors a message that contains the new $d'(u, v)$. This algorithm is given in detail as Algorithm 1.

Algorithm 1: Basic Bellman-Ford for u

```
  Initialization: $d' = \infty$
1 For each neighbor $w$ of $u$, get message $a(w)$
2 $z \leftarrow \min_{w \in N(u)}\{a(w) + d(u, w)\}$
3 if $z < d'$ then
4     $d' \leftarrow z$
5     Send message $d'$ to all neighbors
```

In this description we assume that there is one source v that is known to all nodes, but this clearly is not necessary. With multiple unknown sources each message could also contain the ID of the source and each node u could keep track of its guesses $d'(u, \cdot)$ for every source that it has seen at least one message from. The standard analysis of Bellman-Ford (see e.g. [32]) gives the following lemmas:

Lemma 3 *At the end of phase $k-1$, every node $u \in V$ knows which vertices are in A_{k-1} as well as $d(u, v)$ for all $v \in A_{k-1}$.*

Lemma 4 *Phase $k-1$ uses at most $O(|A_{k-1}|S)$ rounds.*

To handle phase i, we will assume inductively that $B_{i+1}(u)$ is known to u at the start of phase i, as well as the distance from u to every node in $B_{i+1}(u)$. In particular, we will assume that u knows its distance to the *closest* node in A_{i+1}, i.e. $d(u, A_{i+1})$. In phase i we will simply use a modified version of Bellman-Ford in which the sources are $A_i \backslash A_{i+1}$, but node u only "participates" in the algorithm for sources $v \in A_i \backslash A_{i+1}$ when it gets a message that implies that $d(u, v) < d(u, A_{i+1})$, i.e. that $v \in B_i(u)$. To handle the multiple sources, each node u will maintain for every possible source $v \in V$ an outgoing message queue, which will only ever have a 0 or 1 message in it. u just does round-robin scheduling among the nonempty queues, sending the current message to all neighbors and removing it from the queue. To simplify the code, we will assume without loss of generality that $V = \{0, 1, \ldots, n-1\}$ (this assumption is only used to simplify the round-robin scheduler, and can easily be removed). This algorithm is given as Algorithm 2.

Algorithm 2: Modified Bellman-Ford for node u in phase i

```
1  Initialization:
2  foreach $v \in V \backslash \{u\}$ do
3      $d'(v) \leftarrow \infty$
4      $q(v) \leftarrow 0$
5      $i \leftarrow 0$

6  In the first round:
7  if $u \in A_i \backslash A_{i+1}$ then
8      Send message $\langle u, 0 \rangle$ to all neighbors

9  In each round:
   //Receive and process new messages
10 foreach $w \in N(u)$ do
11     Get message $m(w) = \langle v_w, a_w \rangle$
12     if $a_w + d(u, w) < d(u, A_{i+1}) \wedge a_w + d(u, w) < d'(v_w)$ then
13         $d'(v_w) \leftarrow a_w + d(u, w)$
14         $q(v_w) \leftarrow 1$
   //Send message from next nonempty queue
15 $i' \leftarrow i$
16 $i \leftarrow (i+1)\%n$
17 while $q(i) = 0 \wedge i \neq i'$ do $i \leftarrow (i+1) \bmod n$ if $q(i) = 1$ then
18     Send message $\langle i, d'(i) \rangle$ to all neighbors
19     $q(i) \leftarrow 0$
```

Lemma 5 *At the end of phase i, every node $u \in V$ knows $B_i(u)$ and its distance to all nodes in $B_i(u)$.*

Proof We prove this by induction on the phase. The base case is phase $k-1$, which is satisfied by Lemma 3. Now consider some phase $i \geq 0$. Let $v \in A_i \backslash A_{i+1}$—we will show by induction on the hop count of the shortest path that all nodes $u \in C(v)$ find out their distance to v. If $u \in C(v)$ is adjacent to v via a shortest path, then obviously after the first round it will know its distance to v. If $u \in C(v)$ is not adjacent to v via a shortest path, then by induction the next hop on the shortest path from u to v finds out its correct distance to v, and thus will forward the announcement to u. Thus at the end of phase i every node $u \in C(v)$ knows its distance from v, and since this holds for every $v \in A_i \backslash A_{i+1}$ we have that every $u \in V$ knows its distance to all nodes in $B_i(u)$. □

We can now bound the time and message complexity of each phase:

Lemma 6 *Each phase takes $O(n^{1/k} S \log n)$ rounds (with high probability).*

Proof Let $v \in A_i \backslash A_{i+1}$. Intuitively, if v were the only vertex in $A_i \backslash A_{i+1}$, then in phase i the algorithm devolves into distributed Bellman-Ford in which the only vertices that ever forward messages are vertices in $C(v)$. This would clearly take $O(S)$ rounds. In the general case, each vertex u only participates in $O(|B_i(u)|) = O(n^{1/k} \log n)$ of these shortest path algorithms, so each "round" of the original algorithm can be split up into $O(n^{1/k} \log n)$ rounds to accommodate all of the different sources. Thus the total time taken is $O(n^{1/k} S \log n)$ as claimed.

To prove this formally, let $u \in V$ and $v \in A_i \backslash A_{i+1}$, with $v \in B_i(u)$. Let $v = v_0, v_1, \ldots, v_{\ell-1} = u$ be a shortest path from v to u with the fewest number of hops. Thus $\ell \leq S$. We prove by induction that v_j receives a message $\langle v, d(v_{j-1}, v)\rangle$ at time at most $O(n^{1/k} j \log n)$, which clearly implies the lemma. For the base case, in the first round v sends out the message $\langle v, 0\rangle$ to its neighbors, so v_1 receives the correct message at time $1 \leq n^{1/k} \log n$. For the inductive step, consider node v_j. We know by induction that v_{j-1} received a message $\langle v, d(v_{j-2}, v)\rangle$ at time at most $t = O(n^{1/k}(j-1) \log n)$. If v_{j-1} already knew this distance from v, then it also already sent a message (or put one in the queue) informing its neighbors (v_j in particular) about this. Otherwise, v_{j-1} puts a message in its outgoing queue at time t. Since the nonempty queues are processed in a round-robin manner, and by Lemma 1 we know that at most $O(n^{1/k} \log n)$ queues are ever nonempty throughout the phase, v_{j-1} sends a message $\langle v, d(v_{j-1} v)\rangle$ at time at most $t + O(n^{1/k} \log n) = O(n^{1/k} j \log n)$. This implies the lemma. □

Lemmas 2, 5, 1 and 6 obviously imply the following theorem:

Theorem 4 *For any $k \geq 1$, there is a distributed sketching algorithm that takes $O(kn^{1/k} S \log n)$ rounds, after which with high probability every node has a sketch of size at most $O(kn^{1/k} \log n)$ words (and expected size $O(kn^{1/k})$ words) that provides approximate distances with stretch $2k-1$.*

Note that the probabilistic guarantee is on the time and the size: the approximation of $2k-1$ is in fact deterministic. This means that in the (extremely unlikely) case that the algorithm fails, the algorithm is still *correct* in that it computes valid distance sketches. But with very low probability these sketches might be large and take a long time to compute. Thus our algorithm is a *Las Vegas* algorithm. With knowledge of S, though, each node knows the value of the upper bound of $O(n^{1/k} S \log n)$ from Lemma 6 on the number of rounds, so if a phase takes more rounds we can detect failure and raise an error and restart with new randomness. Without knowing S this process is more difficult: as we show in the next section we can still detect when a phase ends, but it is no longer clear how to detect failure. We leave this as an interesting open problem.

3.3 Termination detection

We now show how to remove the assumption that every node knows S. Note that we do not show how to satisfy this assumption, i.e. we do not give an algorithm that computes S and distributes it to all nodes. Rather, we show how to detect when a phase has terminated, and thus when a new phase should start. We use basically the same termination detection algorithm as the one used by Khan et al. [23], just adapted to our context.

At the very beginning of the algorithm, even before phase $k-1$, we run a leader election algorithm to designate some arbitrary vertex r as the *leader*, and then build a breadth-first search (BFS) tree T out of r so that every node knows its parent in the tree as well as its children. This can be done in $O(D) \leq O(S)$ rounds and $O(|E| \log n)$ messages [23].

At the beginning of phase i, the leader r sends a message to all nodes (along T) telling them when they should start phase i, so they all begin together. We say that a node u is *complete* if either $u \notin A_i \backslash A_{i+1}$ or every vertex in $C(u)$ knows its distance to u (we will see later how to use echo messages to know when this is the case). So initially the only complete nodes are the ones not in $A_i \backslash A_{i+1}$. Any such node that is also a leaf in T immediately sends a COMPLETE message to its parent in the tree. Throughout the phase, when any node has heard COMPLETE messages from all of its children in T and is itself complete, it sends a COMPLETE message to its parent.

Now suppose that when running phase i, some node u gets a message $m(w) = \langle v_w, a_w\rangle$ from a neighbor w. There are two reasons that this message might not result in a new message added to the send queue: if $a_w + d(u, w) \geq d(u, A_{i+1})$ (v_w has not yet been shown to be in $B_i(u)$), or if $a_w + d(u, w) \geq d'(v_w)$ (u already knows a shorter path to v_w). Furthermore, even if $m(w)$ does result in a new message added to the send queue, it might get superseded by a new message added to the queue with an updated value of $d'(v_w)$ before the value based on $m(w)$ can be sent. All three of these conditions can be tracked by u, so for each message m that u receives (say from neighbor w) it keeps track of whether or not it sends out a new message based on m. If it does not (one of the two conditions failed, or it was superseded), then it sends an ECHO message back to w, together with a copy of the message. If u does send out a new message based on m, then when it has received ECHO messages for m from all of its neighbors (except for w) it sends an ECHO message to w together with a copy of m.

It is easy to see inductively that when a node u sends a message m, it will also know via ECHO messages when m has ceased to propagate in the network since all of its neighbors will have ECHO'd it back to u. So if $u \in A_i \backslash A_{i+1}$, and thus only sends out one message that has first coordinate u, it will know when this message has stopped propagating, which clearly implies that every node $v \in V$ that is in $C(u)$ knows its correct distance to u (as well as the fact that $u \in B_i(v)$). At this point u is complete, so once it has received COMPLETE messages from all of its children in T it will send a COMPLETE message to its parent.

Once r has received COMPLETE messages from all of its children (and is itself complete) the phase is over. So r starts the next phase by sending a START message to all nodes using T, and the next phase begins.

It is easy to see that the ECHOs only double the number of messages and rounds, since any message sent along an edge corresponds to exactly one ECHO sent back the other way. Electing a leader and building a BFS tree take only a negligible number of messages and rounds compared to the bounds of Theorem 4. Each node sends only one COMPLETE message, so there are at most $O(n)$ COMPLETE messages which is tiny compared to the bound in Theorem 4, and the number of extra rounds due to COMPLETE messages is clearly only $O(D)$. Thus even with the extra termination detection, the bounds of Theorem 4 still hold.

4 Sketches with slack

Let $u, v \in V$. We say that v is *ε-far* from u if $|\{w : d(u, w) < d(u, v)\}| \geq \varepsilon n$, i.e. if v is not one of the εn closest nodes to u. Given a labeling $L(u)$ for each $u \in V$, we say that it has stretch t with ε-slack if the distance that we compute for u and v given $L(u)$ and $L(v)$ is at least $d(u,v)$ and at most $t \cdot d(u, v)$ for all $u, v \in V$ where v is ε-far from u. Labelings with slack were previously studied by Chan, Dinitz, and Gupta [5] and Abraham, Bartal, Chan, Dhamdhere, Gupta, Kleinberg, Neiman, and Slivkins [1]. The main technique of Chan et al. was the use of a new type of net they called a *density net*. For each $u \in V$, let $R(u, \varepsilon) = \inf\{r : |B(u, r)| \geq \varepsilon n\}$ be the minimum distance necessary for the ball around u to contain at least εn points, and let $B^{\varepsilon}(u) = B(u, R(u, \varepsilon))$ be this ball. We give a definition of density net that is slightly modified from [5] in order to make it easier to work with in a distributed context.

Definition 1 A set of vertices $N \subseteq V$ is an ε-density net if:

1. For all $u \in V$, there is a vertex $v \in N$ such that $d(u, v) \leq R(u, \varepsilon)$, and
2. $|N| \leq \frac{10}{\varepsilon} \ln n$.

Chan et al. give a centralized algorithm that computes an ε-density net in polynomial time for any ε. Their density nets are somewhat different, in that they contain only $1/\varepsilon$ nodes but the closest net node to u is only guaranteed to be within $2R(u, \varepsilon)$ instead of $R(u, \varepsilon)$. We modify these values in order to give a distributed construction, and in fact with these modifications it is trivial to build density nets via random sampling.

Lemma 7 *There is a distributed algorithm that, with high probability, constructs an ε-density net in constant time.*

Proof The algorithm is simple: every vertex independently chooses to be in N with probability $\frac{5 \ln n}{\varepsilon n}$. The expected size of N is clearly $\frac{5 \ln n}{\varepsilon}$, and by a simple Chernoff bound (see e.g. [31]) we have that the probability that $|N| > \frac{10 \ln n}{\varepsilon}$ is at most $e^{-(20 \ln n)/(3\varepsilon)} \leq 1/n^{6/\varepsilon}$, so the second constraint is satisfied with high probability.

For the first constraint, that for every vertex u there is some vertex $v \in B^{\varepsilon}(u) \cap N$, we split into two cases depending on ε. If $\varepsilon \leq \frac{5 \ln n}{n}$, then every node has probability 1 of being in N, so the condition is trivially satisfied. Otherwise we have $\varepsilon > \frac{5 \ln n}{n}$, so for every u we have $|B^{\varepsilon}(u)| \geq 5 \ln n$ and the expected size of $B^{\varepsilon}(u) \cap N$ is exactly $5 \ln n$. Using a similar Chernoff bound (but from the other direction) gives us that the probability that $|B^{\varepsilon}(u) \cap N|$ is less than 1 is at most $e^{-(25 \ln n)/8} \leq 1/n^3$. Now we can just take a union bound over all u to get that the first constraint is satisfied with high probability. □

Unfortunately, even if we assume knowledge of S it is not clear how to test whether the random set N we construct is in fact a density net. This is because doing so would require knowing $R(u, \varepsilon)$ for every u, which might take a long time to calculate if (for example) ε is large. However, we only fail with very low probability (i.e. $1/n^2$), and by increasing the constant in the bound on $|N|$ from 10 to larger values we can drop this failure probability to any inverse polynomial that we want.

Using this construction, we can efficiently construct short sketches with ε-slack:

Theorem 5 *There is a distributed algorithm that with high probability uses at most $O(S\frac{1}{\varepsilon} \log n)$ rounds and at completion, every node has a sketch that with high probability has size at most $O(\frac{1}{\varepsilon} \log n)$ words and stretch at most 3 with ε-slack.*

Proof The algorithm first uses Lemma 7 to construct an ε-density net N. It is easy to see that the closest net points to u and v are a good approximation to the distance between u and v. In particular, suppose that v is ε-far from u, let u' be the closest node in N to u, and let v' be the closest node in N to v. Then $d(u,u') \le R(u,\varepsilon) \le d(u,v)$ by the definition of ε-far and ε-density nets, $d(v,u') \le d(v,u) + d(u,u') \le 2d(u,v)$, and thus $d(u,u')+d(v,u') \le 3d(u,v)$. This means that if every vertex keeps as its sketch its distance from all nodes in N, we will have sketches with ε-uniform slack and stretch 3 (to compute an approximation to $d(u,v)$ we can just consider use $\min_{w\in N}\{d(u,w)+d(w,v)\}$, which we can compute from the two sketches). The size of these sketches is clearly $O(|N|) = O(\frac{1}{\varepsilon}\log n)$, since for every node in N we just need to store its ID and its distance.

It just remains to show how to compute these sketches efficiently. But this is simple, since it is exactly the k-Source Shortest Paths problem where the sources are the nodes in N. So we simply run the k-source version of Distributed Bellman-Ford, which gives the claimed time and message complexity bounds. □

We can get a different tradeoff by applying Thorup and Zwick to the density net itself, instead of simply having every node remember its distance to all net nodes. This is essentially what is done in the slack labeling schemes of Chan et al. [5], just with slightly different parameters and constructions (since they were able to use centralized constructions). In particular, suppose that we manage to use Thorup-Zwick on the net, so the distances between net points are preserved up to stretch $2k-1$. Then these sketches would have size at most $O(k|N|^{1/k}\log n) = O(k\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n)$. For each $u \in V$, let $u' \in N$ be the closest node in the density net to u. We let the sketch of u be the identity of u', the distance between u and u', and the Thorup-Zwick label of u'. We call this the (ε,k)-*CDG sketch*. Clearly this sketch has size $O(k\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n)$. Let $u,v \in V$ such that v is ε-far from u. Our estimate of the distance will be $d(u,u') + d''(u',v') + d(v',v)$, where $d''(u',u') \le (2k-1)d(u',v')$ is the approximate distance given by the Thorup-Zwick labels. This can obviously be computed given the sketches for u and v. To bound the stretch, we simply use the definition of density nets, the triangle inequality, and the fact that v is ε-far from u. This gives us a distance estimate $d'(u,v)$ with

$$\begin{aligned} d'(u,v) &= d(u,u') + d''(u',v') + d(v',v) \\ &\le d(u,u') + (2k-1)d(u',v') + d(v',v) \\ &\le d(u,v) + (2k-1)(d(u',u) + d(u,v) \\ &\quad + d(v,v')) + 2d(u,v) \\ &\le 3d(u,v) + (2k-1)(4d(u,v)) \\ &= (8k-1)d(u,v) \end{aligned}$$

This gives the basic lemma about these sketches, which was proved by [5] (modulo our modifications to density nets):

Lemma 8 ([5]) *For any $\varepsilon > 0$ and $1 \le k \le O(\log \frac{1}{\varepsilon})$, with high probability the (ε,k)-CDG sketch has size at most $O(k\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n)$ words and $(8k-1)$-stretch with ε-slack.*

It remains to show how to construct (ε,k)-CDG sketches in a distributed manner, which boils down to modifying the algorithm of Theorem 4 to work with the density net rather than with the full point set (this is trivial in a centralized setting since we can just consider the metric completion).

Lemma 9 *For any $\varepsilon > 0$ and $1 \le k \le O(\frac{1}{\varepsilon})$, there is a distributed algorithm so that after $O\left(kS\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n\right)$ rounds, with high probability every node knows its (ε,k)-CDG sketch.*

Proof We first apply Lemma 7 to construct the ε-density net N. We now want every node u to know its closest net node u' and its distance from u'. This can be done via a single use of Distributed Bellman-Ford, where we just imagine a "super node" consisting of all of N. This takes $O(S)$ rounds.

Now we need to run Thorup-Zwick on N. But this is easy to do, since we just modify the A_i sets to be subsets of N instead of V and change the sampling probability from $n^{-1/k}$ to $\left(\frac{10}{\varepsilon}\ln n\right)^{-1/k}$. Note that for every $u \in V$ the bunch $B_i(u)$ is still well defined, and with high probability has size at most $O\left(\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n\right)$ (via an argument analogous to Lemma 1). This means that we can run Algorithm 2 using these new A_i sets and every node will know their Thorup-Zwick sketch for these A_i sets. In particular, the nodes in N will have a sketch that is exactly equal to the sketch they would have if we ran Algorithm 2 on the metric completion of N, rather than on G. It is easy to see that Lemma 6 still applies but with $n^{1/k}\log n$ (the upper bound on the size of each $B_i(u)$) changed to $O\left(\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n\right)$, so each phase takes $O\left(S\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n\right)$ rounds. Since there are k phases, this gives the desired complexity bound.

As before, this assumes that every node knows S in order to synchronize the phases. However, we can remove this assumption by using the termination detection algorithm of Sect. 3.3. This at most doubles the number of rounds and adds an extra $O(D)$ rounds, which is negligible. □

Theorem 6 *For any $\varepsilon > 0$ and integer value $1 \le k \le O(\log \frac{1}{\varepsilon})$, there is a distributed sketching algorithm that completes in at most $O\left(kS\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n\right)$ rounds, after which with high probability every node has a sketch of size at most $O(k\left(\frac{1}{\varepsilon}\log n\right)^{1/k}\log n)$ words that provides approximate distances with stretch $8k-1$ and ε-slack.*

Proof Implied by Lemmas 8 and 9. □

4.1 Gracefully degrading sketches and average stretch

We now show how to use Theorem 6 to construct sketches with bounded *average* stretch, as well as bounded worst-case stretch. Formally, suppose that we have a weighted graph $G = (V, E)$ that induces the metric d and a sketching algorithm that allows us to compute distance estimates d' with the property that $d'(u, v) \geq d(u, v)$ for all $u, v \in V$. The *average stretch* of the sketching algorithm is defined to be $(1/\binom{n}{2}) \sum_{\{u,v\}\in\binom{V}{2}} \frac{d'(u,v)}{d(u,v)}$.

In fact, we will prove a stronger statement, that there are good distributed algorithms for computing *gracefully degrading* sketches. A sketching algorithm is gracefully degrading with $f(\varepsilon)$ stretch if for every $\varepsilon \in (0, 1)$ it is a sketch with stretch $f(\varepsilon)$ and ε-slack. In other words, instead of specifying ε ahead of time (as in the slack constructions) we need a single sketch that works simultaneously for every ε. It is easy to see that when f is $O(\log \frac{1}{\varepsilon})$, gracefully degrading sketches provide the desired average and worst-case stretch bounds (this was implicit in Chan et al. [5], but they only formally showed this for their specific gracefully-degrading construction, which is slightly different than ours):

Lemma 10 *Any gracefully degrading sketching algorithm with $O(\log \frac{1}{\varepsilon})$ stretch has stretch at most $O(\log n)$ and average stretch at most O(1).*

Proof The bound on the worst-case stretch is immediate by setting $\varepsilon = \frac{1}{2n}$. With this setting of ε, every two points are ε-far from each other, and thus the stretch bound of $O(\log \frac{1}{\varepsilon}) = O(\log n)$ holds for all pairs.

To bound the average stretch, for each $1 \leq i \leq \log n$ and vertex $u \in V$ let $A(u, i) = B^{1/2^{i-1}}(u) \cap (V \backslash B^{1/2^i}(u))$. In other words, $A(u, i)$ is the set of points that are outside the smallest ball around u containing at least $n/2^i$ points, but inside the smallest ball around u containing at least $n/2^{i-1}$ points. So $|A(u, i)| = n/2^i$. We can bound the stretch between u and any node in $A(u, i)$ by $O(i)$, since when we set $\varepsilon = 1/2^i$ we have a stretch bound of $O(\log \frac{1}{\varepsilon}) = O(i)$ for the nodes in $A(u, i)$. Then the average stretch is at most

$$\begin{aligned}
\frac{1}{\binom{n}{2}} \sum_{\{u,v\}\in\binom{V}{2}} \frac{d'(u,v)}{d(u,v)} &\leq \frac{1}{n(n-1)} \sum_{u\in V} \sum_{v\neq u} \frac{d'(u,v)}{d(u,v)} \\
&\leq \frac{1}{n(n-1)} \sum_{u\in V} \sum_{i=1}^{\log n} \sum_{v\in A(u,i)} \frac{d'(u,v)}{d(u,v)} \\
&\leq \frac{1}{n(n-1)} \sum_{u\in V} \sum_{i=1}^{\log n} O(i \cdot \frac{n}{2^i}) \\
&\leq \frac{1}{n(n-1)} \sum_{u\in V} O(n) \\
&\leq O(1),
\end{aligned}$$

proving the lemma. □

This lemma reduces the problem of constructing sketches with good average stretch to the problem of constructing gracefully degrading sketches. But this turns out to be simple, given Theorem 6. The intuition behind gracefully degrading sketches is that they work simultaneously for every slack parameter ε, so to create them we simply use $O(\log n)$ different sketches with slack, one for each power of 2 between $1/n$ and 1.

Theorem 7 *There is a distributed gracefully degrading sketching algorithm that with high probability gives sketches of size at most $O(\log^4 n)$ words with $O(\log \frac{1}{\varepsilon})$-stretch and completes in at most $O(S \log^4 n)$ rounds.*

Proof Our construction is simple: for every $1 \leq i \leq \log n$ we use Theorem 6 with slack $\varepsilon_i = \frac{1}{2^i}$ and stretch $k = O(\log \frac{1}{\varepsilon_i}) = O(\log 2^i)$. The sketch remembered by a node is just the union of these $O(\log n)$ sketches. Given the sketches for two different vertices u and v where v is ε-far from u, we can compute the $O(\log n)$ different distance estimates and take the minimum of them as our estimate.

To see that this is gracefully degrading with stretch at most $O(\log \frac{1}{\varepsilon})$, first note that all of the $O(\log n)$ estimates are at least as large as $d(u, v)$, so we just need to show that at least one of the estimates is at most $O(\log \frac{1}{\varepsilon})d(u, v)$. Let ε_i be ε rounded down to the nearest power of $1/2$. Then v is obviously ε_i-far from u, so the estimate for the ε_i-sketch will provide an estimate of at most $O(\log \frac{1}{\varepsilon_i})d(u, v) = O(\log \frac{1}{\varepsilon})d(u, v)$.

Theorem 6, when specialized to the case of $k = O(\log \frac{1}{\varepsilon})$, completes in at most $O(S \log \frac{1}{\varepsilon} \log^2 n)$ rounds and provides sketches of size $O(\log \frac{1}{\varepsilon} \log^2 n)$. Since we just run each of the $O(\log n)$ instantiations of the theorem back to back, the total number of rounds is at most $O(S \log^2 n) \sum_{i=1}^{\log n} \log 2^i = O(S \log^4 n)$ and the size is at most $O(\log^4 n)$. Note that we can handle determination detection for each of these as usual, based on Sect. 3.3. □

Together with Lemma 10, this gives the following corollary:

Corollary 1 *There is a distributed sketching algorithm that with high probability gives sketches of size at most $O(\log^4 n)$ with at most $O(\log n)$-stretch and O(1) average stretch and which completes in at most $O(S \log^4 n)$ rounds.*

Note that, when compared to our sketch from Theorem 4 with $O(\log n)$ stretch, we pay only an extra $O(\log^2 n)$ factor in the size of the sketch as well as the number of rounds, and in return we are able to achieve constant average stretch.

5 Conclusions

In this paper we initiated the study from a theoretical point of view of distributed algorithms for computing distance sketches in a network. We showed that the Thorup-Zwick distance sketches [37], which provide an almost optimal tradeoff between the size of the sketches and their accuracy, can be computed efficiently in a distributed setting, where our notion of efficiency is the standard definition of the number of rounds in the CONGEST model. Combining this distributed algorithm with centralized techniques of Chan et al. [5], that we were also able to turn into efficient distributed algorithms, yielded a combined construction with the same worst-case stretch as the smallest version of Thorup-Zwick, but much better average stretch. This required only a polylogarithmic cost in the size of the sketches and the time necessary to construct them. These results are a first step towards making the theoretical work on distance sketches more practical, by moving from a centralized setting to a distributed setting.

Our distributed algorithms are all randomized. An interesting question is whether the same performance can be obtained using *deterministic* distributed algorithms. In the centralized setting, obtaining a deterministic algorithm for distance oracles was left open in Thorup and Zwicks original paper [37] and later Roditty, Thorup and Zwick [34] show that essentially the same results can be obtained deterministically. We leave it as a open problem to show whether one can obtain an efficient deterministic distributed algorithm using the approach of [34] or another approach.

It would be interesting in the future to weaken the distributed model even further, by working in failure-prone and asynchronous settings, in the hope of eventually getting practical distance sketches with provable performance guarantees.

References

1. Abraham, I., Bartal, Y., Chan, T.-H.H., Dhamdhere, K.D., Gupta, A., Kleinberg, J., Neiman, O., Slivkins, A.: Metric embeddings with relaxed guarantees. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS '05, pp. 83–100, Washington, DC, USA. IEEE Computer Society (2005)
2. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: FOCS, pp. 218–223 (1979)
3. Baswana, S.: Streaming algorithm for graph spanners: single pass and constant processing time per edge. Inf. Process. Lett. **106**(3), 110–114 (2008)
4. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM, pp. 353–366 (2004)
5. Chan, T.-H.H., Dinitz, M., Gupta, A.: Spanners with slack. In: Proceedings of the 14th European Symposium on Algorithms, pp. 196–207 (2006)
6. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Labeling schemes for tree representation. Algorithmica **53**(1), 1–15 (2009)
7. Cooper, B.F.: Quickly routing searches without having to move content. In: IPTPS, pp. 163–172 (2005)
8. Coppersmith, D., Tetali, P., Winkler, P.: Collisions among random walks on a graph. SIAM J. Discret. Math. **6**(3), 363–374 (1993)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, vol. 24, pp. 595–601. MIT Press and McGraw-Hill, Cambridge (2001)
10. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: Proceedings of the ACM SIGCOMM '04 Conference, Portland, Oregon, August (2004)
11. Demetrescu, C., Goldberg, A.V., Johnson, D.S.: Implementation challenge for shortest paths. In: Encyclopedia of Algorithms (2008)
12. Dolev, S., Schiller, E., Welch, J.L.: Random walk for self-stabilizing group communication in ad hoc networks. IEEE Trans. Mob. Comput. **5**(7), 893–905 (2006). also in PODC'02
13. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the streaming model: the value of space. In: SODA, pp. 745–754 (2005)
14. Fouss, F., Pirotte, A., Renders, J.-M., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Trans. Knowl. Data Eng. **19**(3), 355–369 (2007)
15. Ganesh, A.J., Kermarrec, A.-M., Massoulié, L.: Peer-to-peer membership management for gossip-based protocols. IEEE Trans. Comput. **52**(2), 139–149 (2003)
16. Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. J. Algorithms **53**(1), 85–112 (2004)
17. Gkantsidis, C., Mihail, M., Saberi, A.: Hybrid search schemes for unstructured peer-to-peer networks. In: INFOCOM, pp. 1526–1537 (2005)
18. Goldberg, A.V.: Point-to-point shortest path algorithms with preprocessing. In: SOFSEM (1), pp. 88–102 (2007)
19. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizing mutual exclusion. In: PODC, pp. 119–131 (1990)
20. Karger, D.R., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: SPAA, pp. 36–43 (2004)
21. Katz, M., Katz, N.A., Korman, A., Peleg, D.: Labeling schemes for flow and connectivity. SIAM J. Comput. **34**(1), 23–40 (2004)
22. Kempe, D., Kleinberg, J.M., Demers, A.J.: Spatial gossip and resource location protocols. In: STOC, pp. 163–172 (2001)
23. Khan, M., Kuhn, F., Malkhi, D., Pandurangan, G., Talwar, K.: Efficient distributed approximation algorithms via probabilistic tree embeddings. In: Proceedings of 27th ACM Symposium on Principles of Distributed Computing (PODC), 2008. Journal version: Distributed Computing (2012)
24. Khan, M., Pandurangan, G.: A fast distributed approximation algorithm for minimum spanning trees. Distrib. Comput. **20**, 391–402 (2008)
25. Kleinberg, J., Slivkins, A., Wexler, T.: Triangulation and embedding using small sets of beacons. J. ACM **56**, 32:1–32:37 (2009)
26. Kleinberg, J.M.: The small-world phenomenon: an algorithm perspective. In: STOC, pp. 163–170 (2000)
27. Law, C., Siu, K.: Distributed construction of random expander networks. In: Proceedings of IEEE INFOCOM (2003)
28. Loguinov, D., Kumar, A., Rai, V., Ganesh, S.: Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In: SIGCOMM, pp. 395–406 (2003)
29. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: ICS, pp. 84–95 (2002)
30. Morales, R., Gupta, I.: Avmon: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In: ICDCS, p. 55 (2007)
31. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
32. Pandurangan, G., Khan, M.: Theory of communication networks. In: Algorithms and Theory of Computation Handbook, Second Edition. CRC Press (2009)

33. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia, PA (2000)
34. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: ICALP, pp. 261–272 (2005)
35. Slivkins, A.: Distance estimation and object location via rings of neighbors. In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC '05, pp. 41–50. ACM, New York, NY (2005)
36. Slivkins, A.: Towards fast decentralized construction of locality-aware overlay networks. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC '07, pp. 89–98. ACM, New York, NY (2007)
37. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM **52**(1), 1–24 (2005)
38. Wong, B., Slivkins, A., Sirer, E.G.: Meridian: a lightweight network location service without virtual coordinates. In: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '05, pp. 85–96. ACM, New York, NY (2005)
39. Yen, L., Saerens, M., Mantrach, A., Shimbo, M.: A family of dissimilarity measures between nodes generalizing both the shortest-path and the commute-time distances. In: KDD, pp. 785–793 (2008)
40. Zhong, M., Shen, K., Seiferas, J.I.: Non-uniform random membership management in peer-to-peer networks. In: INFOCOM, pp. 1151–1161 (2005)