Distrib. Comput. (2018) 31:119–137
https://doi.org/10.1007/s00446-017-0304-4

On efficient distributed construction of near optimal routing schemes

Michael Elkin[1] · Ofer Neiman[1]

Received: 15 November 2016 / Accepted: 25 May 2017 / Published online: 1 June 2017

Abstract Given a distributed network represented by a weighted undirected graph $G = (V, E)$ on n vertices, and a parameter k, we devise a randomized distributed algorithm that whp computes a routing scheme in $O(n^{1/2+1/k} + D) \cdot n^{o(1)}$ rounds, where D is the hop-diameter of the network. Moreover, for odd k, the running time of our algorithm is $O(n^{1/2+1/(2k)} + D) \cdot n^{o(1)}$. Our running time nearly matches the lower bound of $\tilde{\Omega}(n^{1/2} + D)$ rounds (which holds for any scheme with polynomial stretch). The routing tables are of size $\tilde{O}(n^{1/k})$, the labels are of size $O(k \log^2 n)$, and every packet is routed on a path suffering stretch at most $4k - 5 + o(1)$. Our construction nearly matches the state-of-the-art for routing schemes built in a centralized sequential manner. The previous best algorithms for building routing tables in a distributed small messages model were by Lenzen and Patt-Shamir (In: Symposium on theory of computing conference, STOC'13, Palo Alto, CA, USA, 2013) and Lenzen and Patt-Shamir (In: Proceedings of the 2015 ACM symposium on principles of distributed computing, PODC 2015, Donostia-San Sebastián, Spain, 2015). The former has similar properties but suffers from substantially larger routing tables of size $O(n^{1/2+1/k})$, while the latter has sub-optimal running time of $\tilde{O}(\min\{(nD)^{1/2} \cdot n^{1/k}, n^{2/3+2/(3k)} + D\})$.

A preliminary version [11] of this paper was published in PODC'16.

Michael Elkin: This research was supported by the ISF Grant No. (724/15). Ofer Neiman: Supported in part by ISF Grant No. (523/12) and by BSF Grant No. 2015813.

✉ Ofer Neiman
neimano@cs.bgu.ac.il

Michael Elkin
elkinm@cs.bgu.ac.il

[1] Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

1 Introduction

A routing scheme in a distributed network is a mechanism that allows packets to be delivered from any node to any other node. The network is represented as a weighted undirected graph, and each node should be able to forward incoming data by using local information stored at the node, and the (short) packet's header. The local routing information is often referred to as a routing table. The routing scheme has two main phases: in the preprocessing phase, each node is assigned a routing table and a short label. In the routing phase, each node receiving a packet should make a local decision, based on its own routing table and the packet's header (which contains the label of the destination), to which neighbor to forward the packet to. The *stretch* of a routing scheme is the worst ratio between the length of a path on which a packet is routed to the shortest possible path.

Designing efficient routing schemes is a central problem in the area of distributed networking, and was studied intensively [1,2,4,6,7,15,30,30,33]. The first general tradeoffs for this problem were given in pioneering works by Peleg and Upfal [30] and Awerbuch and Bar-Noy [1]. In a seminal paper [33], Thorup and Zwick presented the following compact routing scheme: Given a weighted graph G on n vertices and a parameter $k \geq 1$, the scheme has routing tables of size $\tilde{O}(n^{1/k})$,[1] labels of size $O(k \log n)$ and stretch $4k - 5$. (Assuming that port numbers may be assigned by the routing process, otherwise the label size increases by a factor

[1] The $\tilde{O}$ hides $\log^{O(1)} n$ factors.

of $\log n$.)[2] The state-of-the-art is a scheme of [4], which is based on [33], and improves the stretch to $3.68k$.

All the results above assume that the preprocessing phase can be executed in a sequential centralized manner. However, as the problem of designing a compact routing scheme is inherently concerned with a distributed network, constructing the scheme efficiently in a distributed manner is a very natural direction. We focus on the standard CONGEST model [27]. In this model, every vertex initially knows only the edges touching it, and communication between vertices occurs in synchronous *rounds*. On every round, each vertex may send a small message to each of its neighbors. Every message takes a unit time to reach the neighbor, regardless of the edge weight. The time complexity is measured by the number of rounds it takes to complete a task (we assume local computation does not cost anything). Often the time depends on n, the number of vertices, and D, the *hop-diameter* of the graph. The hop-diameter is the maximum hop-distance between two vertices, where the hop-distance is the minimal number of edges on a path between the vertices (regardless of the weights). The hop-diameter is not to be confused with the *shortest path diameter* S, which is the maximal number of hops a shortest path uses (assuming shortest paths are unique). We always have $D \leq S$, and typically D is small while S could be as large as $\Omega(n)$. We also assume, as common in the literature [13,17,19,20,25], that edge weights are integers and at most polynomial in n (so that they can be sent in a single message).

A rich research thread is concerned with finding efficient distributed (approximation) algorithms for classical graph problems (e.g., minimum spanning tree, minimum cut, shortest paths), in sub-linear time [8,14,17,29,32]. There are several results obtaining running times of the form $\tilde{O}(\sqrt{n} + D)$, e.g. for MST, connectivity, minimum cut, approximate shortest path tree, etc. These results are often accompanied by (nearly) matching lower bounds. The lower bound of [32], based on [9,29], implies that devising a routing scheme with any polynomial stretch requires $\tilde{\Omega}(\sqrt{n}+D)$ rounds.

The first result on computing a routing scheme in a distributed manner within $o(n)$ rounds (for general graphs with $D = o(n)$), was shown by Lenzen and Patt-Shamir [20].[3] Their algorithm, given a graph on n vertices and a parameter k, provides routing tables of size $\tilde{O}(n^{1/2+1/k})$, labels of size $O(\log n \cdot \log k)$, stretch at most $O(k \log k)$, and has a nearly optimal running time of $\tilde{O}(n^{1/2+1/k} + D)$ rounds. Note that the routing tables are of size $\Omega(\sqrt{n})$ for any value of k, which could be prohibitively large (the routing scheme of [33] supports stretch 3 with $\tilde{O}(\sqrt{n})$ table size). They also show implications for related problems, such as approximate diameter, generalized Steiner forest, and distance estimation. In a follow-up paper, [22] showed how to improve the stretch of the above scheme to roughly $3k/2$ (for any k divisible by 4). They also exhibited a different tradeoff, that overcame the issue of large routing tables. They devised an algorithm that produced routing tables of size $\tilde{O}(n^{1/k})$, labels of size $O(k \log^2 n)$ and stretch $4k - 3 + o(1)$,[4] but the number of rounds increases to $\tilde{O}(\min\{(nD)^{1/2} \cdot n^{1/k}, n^{2/3+2/(3k)} + D\})$. Note that for moderately large hop-diameter $D \approx n^{1/3}$, the number of rounds is bounded by only $\approx n^{2/3}$ for any value of k. (They also show a variant where the number of rounds is $\tilde{O}(S + n^{1/k})$, but as was mentioned above, S might be much larger than D.) All these and our results are not in the setting of *name-independent* routing, in which the label of a vertex is its ID. Lenzen and Patt-Shamir [20] showed a strong lower bound for the latter setting: any such scheme with stretch ρ (even average stretch ρ) must take $\tilde{\Omega}(n/\rho^2)$ rounds to compute in this model.

In the *distance estimation* problem (also known as sketching, or distance labeling), we wish to compute a small *sketch* for each vertex, so that given any two sketches, one can efficiently compute the (approximate) distance between the vertices. This problem was introduced in [28], who provided initial existential results. In [31], a distributed (randomized) algorithm running in $\tilde{O}(S \cdot n^{1/k})$ rounds was shown, that computes sketches of size $O(kn^{1/k} \log n)$ with stretch at most $2k - 1$. While this essentially matches the best sequential algorithm of [34], the number of rounds could be $\Omega(n)$, even when D is small. In [20], a running time of $\tilde{O}(n^{1/2+1/k} + D)$ rounds was presented, at the cost of significantly increasing the stretch to $O(k^2)$.[5] Izumi and Wattenhofer [18] showed a lower bound of $n^{1/2+\Omega(1/k)}$ rounds for this problem. In the Conclusion part of their paper [18], Izumi and Wattenhofer posed an open problem:

> An open problem related to our results is to find algorithms whose running time gets close to our lower bounds.

Our contribution We devise a randomized distributed algorithm running in $(n^{1/2+1/k}+D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds, that with high probability (whp),[6] computes a compact routing scheme with routing tables of size $O(n^{1/k} \log^2 n)$, labels of size $O(k \log^2 n)$, and stretch at most $4k - 5 +$

[2] They also presented stretch $2k-1$, assuming "handshaking": allowing the source and destination to communicate before the routing phase begins, but it is often desirable to avoid handshaking. Henceforth, we discuss only routing schemes that do not allow handshaking.

[3] We remark that for the class of k-chordal graphs, [26] showed a construction of a routing scheme that could be computed efficiently in a distributed manner.

[4] The paper [22] claimed label size $O(k \log n)$, but in [23] it was communicated to us that the actual size is $O(k \log^2 n)$.

[5] In fact, they showed a scheme in which it suffices to have a sketch of one vertex, and a $O(k \log n)$ size label of the other vertex, to derive the distance estimation. Our result has a similar property.

[6] By "high probability" we mean with probability at least $1 - n^{-c}$, for any desired constant c.

Table 1 Comparison of compact routing schemes for graphs with n vertices, m edges, hop-diameter D, and shortest path diameter S

	Number of rounds	Table size	Label size	Stretch
[4,33]	$O(m)$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log n)$	$3.68k$
[22]	$\tilde{O}(S + n^{\frac{1}{k}})$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log n)$	$4k - 3$
[20,22]	$\tilde{O}(n^{\frac{1}{2}+\frac{1}{4k}} + D)$	$\tilde{O}(n^{\frac{1}{2}+\frac{1}{4k}})$	$O(\log n)$	$6k - 1 + o(1)$
[22]	$\tilde{O}(\min\{(nD)^{\frac{1}{2}} \cdot n^{\frac{1}{k}}, n^{\frac{2}{3}+\frac{2}{3k}} + D\})$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log^2 n)$	$4k - 3 + o(1)$
[24]	$(n^{\frac{1}{2}+\frac{1}{k}} + D) \cdot 2^{\tilde{O}(\sqrt{\log n})}$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log^2 n)$	$4k - 3 + o(1)$
This paper, even k	$(n^{\frac{1}{2}+\frac{1}{k}} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log^2 n)$	$4k - 5 + o(1)$
This paper, odd k	$(n^{\frac{1}{2}+\frac{1}{2k}} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log^2 n)$	$4k - 5 + o(1)$

$o(1)$. Moreover, for odd k, the running time of our algorithm is $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$. Note that our result nearly matches the construction of [33], up to logarithmic terms in the size and $o(1)$ additive term in the stretch. This is even though the latter is computed in a sequential centralized manner. Observe that our running time nearly matches the lower bound of [32], and is substantially better than that of [22] whenever $D \geq n^{\Omega(1)}$ (which achieved similar size-stretch tradeoff). The previous result obtaining near optimal running time [20], suffers from excessive routing table size.

As a corollary, we show a distance estimation scheme, that can be computed in a distributed manner in $(n^{1/2+1/k} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds for even k, and for odd k in $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds, providing sketches of size $O(n^{1/k} \log n)$ with stretch $2k-1+o(1)$. Each distance estimation takes only $O(k)$ time. Our result combines the improved running time of [20] (up to lower order terms), with the near optimal size-stretch tradeoff of [31]. Moreover, our bound for the running time of distance estimation scheme nearly matches the lower bound $n^{1/2+\Omega(1/k)}$ of Izumi and Wattenhofer [18], addressing their open problem. See Table 1 for a concise summary of previous and our results.

We note that to the best of our knowledge, all existing routing schemes [1,2,4,23,30,34], as well as the routing scheme that we present in this paper, enable distance estimation, i.e., given routing tables and labels of a pair u, v of vertices, one can compute (without communication) a distance estimate $\hat{d}(u, v)$, which approximates the actual distance $d_G(u, v)$ between u and v up to the stretch factor of the routing scheme. All routing schemes of this type require, by the lower bound of [18], at least $n^{1/2+\Omega(1/k)}$ rounds to compute.

When preparing this submission, we learnt that concurrently and independently of us [24] came up with a distributed algorithm running in $(n^{1/2+1/k} + D) \cdot 2^{\tilde{O}(\sqrt{\log n})}$ rounds, that with high probability, computes a routing scheme with routing tables of size $\tilde{O}(n^{1/k})$, labels of size $O(k \log^2 n)$, and stretch at most $4k - 3 + o(1)$. Their result has slightly worse stretch, and a larger number of rounds whenever $k < \sqrt{\log n / \log \log n}$, or if k is odd.

1.1 Overview of techniques

Let us first briefly sketch the Thorup–Zwick construction of a routing scheme. First they designed a routing scheme for trees, with routing tables of constant size and logarithmic label size. (Throughout the paper, the size is measured in RAM words, i.e., each word is of size $O(\log n)$.) For a general graph $G = (V, E)$ on n vertices, they randomly sample a collection of sets $V = A_0 \supseteq A_1 \cdots \supseteq A_k = \emptyset$, where for each $0 < i < k$, each vertex in A_{i-1} is chosen independently to be in A_i with probability $n^{-1/k}$. The *cluster* of a vertex $u \in A_i \backslash A_{i+1}$ is defined as

$$C(u) = \{v \in V : d_G(u, v) < d_G(v, A_{i+1})\}. \tag{1}$$

They proved that each cluster $C(x)$ can be viewed as a tree rooted at x, and showed an efficient procedure that given a pair $u, v \in V$, finds a vertex x so that routing in the tree $C(x)$ has small stretch. So each vertex u maintains in its routing table the routing information for all trees $C(x)$ containing it, while the label of u consists of the tree-labels for a few special trees. They also show that (with high probability) every vertex is contained in at most $\tilde{O}(n^{1/k})$ trees.

The first difficulty we must deal with is that the routing scheme of Thorup–Zwick for a (single) tree could take a linear number of rounds to construct. We thus develop a variation on that scheme, that can be implemented efficiently in a distributed network. The basic idea is inspired by Kutten and Peleg [19] (and also used in [25]), which is to select $\approx\sqrt{n}$ vertices that partition the tree into bounded depth subtrees. We then apply the TZ-scheme locally in every subtree. The subtler part is to design a global routing scheme for the virtual tree[7] induced on the sampled vertices, which must incorporate the local routing information.

[7] By a virtual tree we mean a tree whose edges are not present in the network.

Approximate clusters Once we have a distributed algorithm for routing in trees, we set off to apply the TZ-scheme for general graphs. Unfortunately, it is not known how to compute the exact clusters efficiently in a distributed manner. In order to circumvent this barrier, we introduce the notion of *approximate clusters*. An approximate cluster is a subset of a cluster, that may exclude vertices that are "near" the boundary. [Slightly more formally, we may omit vertices for which the inequality (1) becomes false if we multiply the left hand side by a $1+\epsilon$ factor, for a small $\epsilon > 0$.] Our main technical contributions are: exhibiting a procedure that computes these approximate clusters, and showing that these approximate clusters are sufficient for constructing a routing scheme, with nearly matching size and stretch as in [33].

The construction of clusters $C(u)$ for $u \in A_i \backslash A_{i+1}$, where $i < k/2$, can be done in a straightforward manner (within the allotted number of rounds), since the depth of the corresponding tree is $\tilde{O}(\sqrt{n})$ with high probability, and since the *overlap* (the number of clusters containing a fixed vertex) is only $\tilde{O}(n^{1/k})$. The main challenge is computing the approximate clusters in the large scales, for $i \geq k/2$. To this end, we employ several tools. The first is *approximate multi-source hop-bounded distance computation*, which appeared recently in [25] (a certain variant of it appeared also in [21]). This enables us to compute approximations for B-hops shortest paths (paths that use at most B edges), from a given m sources to every vertex, in $\tilde{O}(B+m+D)$ rounds. The second tool we use is *hopsets*. The notion of hopsets was introduced by Cohen [5] in the context of parallel approximate shortest path algorithms, and it has found applications in dynamic, streaming and distributed settings as well [3,16,17]. A (β,ϵ)-hopset is a (small) set of edges F, so that every shortest path has a corresponding β-hops path, whose weight is at most $1+\epsilon$ larger.

We compute the approximate clusters in the large scales as follows. First we sample $\approx\sqrt{n}$ vertices (those in $A_{k/2}$), and compute approximate $\sqrt{n}$-hops shortest paths from all the sampled vertices. Next we apply a (β,ϵ)-hopset on the graph induced by these sampled vertices, where $\beta \leq 2^{\tilde{O}(\sqrt{\log n})}$ and $\epsilon \approx 1/k^4$. (A pair of sampled vertices is connected in this graph by an edge if and only if one is reachable from the other via an approximate $\sqrt{n}$-hop-bounded shortest path.) An efficient distributed algorithm to construct such hopsets is given by Henzinger et al. [17] and Elkin and Neiman [10]. We shall use the construction of [10], since it facilitates much smaller β whenever k is small. (There are also some additional properties of hopsets from [10], that make them more convenient in the context of routing. See Sect. 2.) This enables us to compute the approximate clusters on the sampled vertices, since we need only β steps of exploration from each source u, using again that the overlap is small. Finally, we extend each approximate cluster to the other vertices, by initiating an exploration from each sampled vertex to hop-distance $\approx\sqrt{n}$ in the original graph (in fact, one can use the multi-source hop-bounded distance computation of [25]). The correctness follows since with high probability, every vertex that should be included in some approximate cluster $\tilde{C}(u)$, has either u or a sampled vertex within $\approx\sqrt{n}$ hops on the shortest path to it. The thresholds for entering an approximate cluster must be set carefully, so that every vertex on that shortest path will also join $\tilde{C}(u)$, in order to guarantee that the trees will indeed be connected (which is clearly crucial for routing), and on the other hand, to make sure that no vertex participates in too many trees. Unlike the exact TZ clusters, approximate clusters generally do not have to be connected.

The fact that our clusters are only approximate induces increased stretch. The analysis is similar to that of [34], which consists of k iterations of searching for the "right" tree. We must pay a factor of $1+O(\epsilon)$ in every one of these iterations, but fortunately, the hopset construction allows us to take sufficiently small ϵ, so that all the additional stretch accumulates to an additive $o(1)$.

From a high level, our approach is similar to those of [20, 22]. In [22], they also use a variant of the TZ-routing scheme, which allows small errors in the distance estimations. The main difference is in handling the large scales. In [20], the idea was to build a spanner on a sample of $\approx\sqrt{n}$ vertices, which reduces the number of edges. So a routing scheme can be efficiently computed on the spanner, and then extended to the entire graph. This approach inherently suffers from large storage requirement, since every vertex needs to know all the spanner edges. In [22] the idea was to "delay" the start of large scales from $k/2$ to roughly $l_0 = (k/2)\cdot(1+\log D/\log n)$. Then they apply a distance estimation on the sampled vertices at scale l_0 (those in A_{l_0}) to construct the routing tables for all higher scales, and extend these to the remainder of the graph. However, the exploration in the graph on A_{l_0} may need to be of $\approx n^{1-l_0/k}$ hops, which induces a factor of $D \cdot n^{1-l_0/k} = (nD)^{1/2}$ to the number of rounds. The use of hopsets allows us to avoid the large memory requirement, since the routing is oblivious to the hopset, while significantly shortening the exploration range. Since the exploration range is proportional to the running time, the latter also decreases.

1.2 Organization

After stating in Sect. 2 some of the tools we shall apply, in Sect. 3 we describe the notion of approximate clusters, and show how to compute these efficiently in a distributed manner. In Sect. 4 we show our distributed tree routing. Then in Sect. 5, we demonstrate how these approximate clusters could be used for a routing scheme in general graphs. Finally, in Sect. 6 we show the distance estimation scheme.

2 Preliminaries

Let $G = (V, E, w)$ be a weighted graph on n vertices. We assume that $w : E \to \{1, \dots, \text{poly}(n)\}$ (without this assumption, there will be a logarithmic dependence on the aspect ratio in the data structures' size and running times). Let D be the *hop-diameter* of G, that is, the diameter of G if all weights were 1. Denote by d_G the shortest path metric on G. Assume, as we may, that all shortest paths are unique. Let $d_G^{(t)}$ be the *t-hops* shortest path distance (abusing notation, since this is not a metric). That is, $d_G^{(t)}(u, v)$ is the shortest length of a path from u to v, that has at most t edges (set $d_G^{(t)}(u, v) = \infty$ if every path from u to v has more than t edges). For each $u, v \in V$, define $h_G(u, v)$ as the number of hops on the shortest path in G between u and v. We shall always use this notation with respect to the input graph G, and thus will omit the subscript. A (dominating) *virtual graph* on G is a graph $G' = (V', E', w')$ with $V' \subseteq V$, and for every $u, v \in V'$ we have that $d_{G'}(u, v) \ge d_G(u, v)$. Every vertex in V' should know all the edges of E' touching it. The following lemma formalizes the broadcast ability of a distributed network (see, e.g., [27]).

Lemma 1 *Suppose every $v \in V$ holds m_v messages, each of $O(1)$ words, for a total of $M = \sum_{v \in V} m_v$. Then all vertices can receive all the messages within $O(M + D)$ rounds.*

Bellman–Ford algorithm The classical Bellman–Ford is an algorithm to compute shortest paths in a graph from a certain root vertex $u \in V$ (or a set of vertices). Every vertex $v \in V$ holds a vector b_v of distances to other vertices. Initially, $b_u(u) = 0$ and all other entries are ∞. The algorithm is executed in iterations. In every iteration v communicates b_v to its neighbors, and updates b_v according to the messages it received from its neighbors and the edge weights to those neighbors. In order to provide efficient implementation in distributed models, we often will use bounded-depth explorations, where v will send $b_v(u)$ to its neighbors iff some condition is met. Usually the condition is that $b_v(u)$ is sufficiently small, and the exact bound might change in different settings.

2.1 Tools

We will make use of the following theorem due to [25, Theorem 3.6], which shows how to compute hop-bounded distances from a given set of sources efficiently in a distributed manner.

Theorem 1 [25] *Given a weighted graph $G = (V, E, w)$ of hop-diameter D, a set $V' \subseteq V$, and parameters $B \ge 1$ and $0 < \epsilon < 1$, there is a (randomized) distributed algorithm that whp runs in $\tilde{O}(|V'| + B + D)/\epsilon$ rounds, so that every $u \in V$ will know values $\{d_{uv}\}_{v \in V'}$ satisfying*[8]

$$d_G^{(B)}(u, v) \le d_{uv} \le (1 + \epsilon) d_G^{(B)}(u, v), \tag{2}$$

Remark 1 While not explicitly stated in [25], the proof also provides that each $u \in V$ knows, for every $v \in V'$, a vertex $p = p_v(u)$ which is a neighbor of u satisfying

$$d_{uv} \ge w(u, p) + d_{pv}. \tag{3}$$

Hopsets The following notion of hopsets was introduced by Cohen [5].

Definition 1 (*Hopsets*) A set of (weighted) edges F is a (β, ϵ)-hopset for a weighted graph $G = (V, E, w)$, if in the graph $H = (V, E \cup F)$, for every $u, v \in V$,

$$d_G(u, v) \le d_H(u, v) \le d_H^{(\beta)}(u, v) \le (1 + \epsilon) d_G(u, v). \tag{4}$$

We will need the following *path-reporting* property from our hopset. This property will be crucial for the connectivity of the trees corresponding to the approximate clusters.

Property 1 *A hopset F for a graph G is called* path-reporting, *if for every hopset edge $(u, v) \in F$ of weight b, there exists a corresponding path P in G between u and v of length b. Furthermore, every vertex x on P knows $d_P(x, u)$ and $d_P(x, v)$, and its neighbors on P.*

The following result is from [10], which provides a path-reporting hopset. We remark that the original hopset construction of [5] could be made path-reporting. Also, in [17, Theorem 4.10], a distributed algorithm constructing a hopset is provided, which possibly could be made path-reporting, however, it inherently cannot provide a better hopbound than $2^{\tilde{O}(\sqrt{\log n})}$.

Theorem 2 [10] *Let G be a weighted graph on n vertices with hop-diameter D, let $0 < \epsilon < 1$, and let G' be a virtual graph on G with m vertices. Let $0 < \rho < 1/2$ be a parameter, and write $\beta = \left(\frac{\log m}{\epsilon \cdot \rho}\right)^{O(1/\rho)}$. Then there is a randomized distributed algorithm that whp computes in $\tilde{O}(m^{1+\rho} + D) \cdot \beta^2$ rounds, a path-reporting (β, ϵ)-hopset F for G'.*

We remark that in many applications (see, e.g., applications in [5,10]) the size of the hopset is important. However, here we care about the number of rounds required to compute the hopset, and not its size.
Approximate shortest path tree (SPT) Recently, [17] obtained an efficient distributed algorithm for computing an approximate SPT, which we shall use. Let us first define the problem

[8] The computed values are symmetric, that is, $d_{uv} = d_{vu}$ whenever $u, v \in V'$.

formally. Let $G = (V, E, w)$ be a weighted graph. Given a set of vertices $A \subseteq V$, computing an $(1 + \epsilon)$-approximate SPT rooted at A, means that every vertex $u \in V$ will know a value $\hat{d}(u)$ satisfying

$$d_G(u, A) \le \hat{d}(u) \le (1 + \epsilon) d_G(u, A), \quad (5)$$

and that u will know a vertex $\hat{z}(u) \in A$ so that $d_G(u, \hat{z}(u)) \le \hat{d}(u)$. The following theorem is a slight variation on a theorem shown in [17]. Here we use the hopsets of [10] for an improved running time.

Theorem 3 *Let $G = (V, E, w)$ be a weighted graph on n vertices with hop-diameter D. Given a set $A \subseteq V$ of size $|A| \le 2\sqrt{n} \ln n$, integer $k \ge 1$ and $\frac{1}{\text{polylog } n} < \epsilon < 1$, there is a distributed algorithm that computes an $(1 + \epsilon)$-approximate SPT rooted at A in $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds.*

We defer the proof to Appendix 1.

3 Distributed routing scheme

In this section we define the notions of approximate pivots and approximate clusters, and describe an efficient distributed algorithm that computes these. Let us first recall the basic definitions from [34].

Let $G = (V, E, w)$ be a weighted graph and fix $k \ge 1$. Sample a collection of sets $V = A_0 \supseteq A_1 \cdots \supseteq A_k = \emptyset$, where for each $0 < i < k$, each vertex in A_{i-1} is chosen independently to be in A_i with probability $n^{-1/k}$. A point $z \in A_i$ is called an i-pivot of v, if $d_G(v, z) = d_G(v, A_i)$. The cluster of a vertex $u \in A_i \backslash A_{i+1}$ is defined as

$$C(u) = \{v \in V : d_G(u, v) < d_G(v, A_{i+1})\}. \quad (6)$$

We quote a claim from [34], which provides a bound on the overlap of clusters.

Claim 2 *With high probability, each vertex is contained in at most $4n^{1/k} \log n$ clusters.*

The following claim shows that (with high probability) the sets A_i have favorable properties.

Claim 3 *With high probability the following holds for every $0 \le i \le k - 1$: (1) $|A_i| \le 4n^{1-i/k} \ln n$, and (2) For every $u, v \in V$ such that $h(u, v) > 4n^{i/k} \ln n$, there exists a vertex of A_i on the shortest path between u and v.*

Proof Fix i. The first assertion holds by a simple Chernoff bound, since every vertex is chosen to be in A_i independently with probability $n^{-i/k}$, and the expected size of A_i is $n^{1-i/k}$. For the second assertion, let u, v be such that $h(u, v) > 4n^{i/k} \ln n$ (recall that $h(u, v)$ is the number of hops on the shortest path from u to v in G). The probability that none of the vertices on the unique u to v shortest path is included in A_i is at most

$$\left(1 - n^{-i/k}\right)^{4n^{i/k} \ln n} \le n^{-4}.$$

Taking a union bound on the k possible values of i and $\binom{n}{2}$ pairs completes the proof. □

From now on assume that all the events in the claims above hold, which yields the following corollary.

Corollary 4 *For any $0 \le i < k - 1$, $u \in A_i \backslash A_{i+1}$ and $v \in C(u)$, it holds that $h(u, v) \le 4n^{(i+1)/k} \ln n$.*

Proof If it were the case that $h(u, v) > 4n^{(i+1)/k} \ln n$, then Claim 3 would imply that there exists a vertex of A_{i+1} on the shortest path from v to u. In particular, $d_G(v, u) > d_G(v, A_{i+1})$, which contradicts (6). □

3.1 Approximate clusters and pivots

Since we do not know how to compute efficiently in a distributed manner the pivots and clusters, we settle for an approximate version, which is formally defined in this section. Fix the parameter $\epsilon = \frac{1}{48k^4}$. For each $v \in V$ and $0 \le i \le k - 1$, a point $\hat{z} \in A_i$ is called an *approximate i-pivot* of v if

$$d_G(v, \hat{z}) \le (1 + \epsilon) d_G(v, A_i). \quad (7)$$

Now we define for each $0 \le i \le k - 1$ and each vertex $u \in A_i \backslash A_{i+1}$, a set of vertices which we call an *approximate cluster*. The approximate cluster is a subset of the cluster $C(u)$, and it is allowed to exclude vertices of $C(u)$ which are "close" to the boundary. First define the vertices that are far from the boundary (with respect to ϵ), as

$$C_\epsilon(u) = \left\{v \in V : d_G(u, v) < \frac{d_G(v, A_{i+1})}{1 + \epsilon}\right\}. \quad (8)$$

The approximate cluster $\tilde{C}(u)$ will be a set that satisfies the following:

$$C_{6\epsilon}(u) \subseteq \tilde{C}(u) \subseteq C(u). \quad (9)$$

Each approximate cluster $\tilde{C}(u)$ we compute, will be stored as a tree rooted at u; that is, each vertex $v \in \tilde{C}(u)$ will store a pointer to its parent in the tree. This tree (abusing notation, we call this tree $\tilde{C}(u)$ as well) has the property that distances to the root u are approximately preserved; that is, for any $v \in \tilde{C}(u)$ we have that

$$d_G(u, v) \le d_{\tilde{C}(u)}(u, v) \le (1 + \epsilon)^4 d_G(u, v). \quad (10)$$

Remark 2 Since $\tilde{C}(u) \subseteq C(u)$, Claim 2 implies that with high probability, each vertex is contained in at most $4n^{1/k}\log n$ approximate clusters.

In the remainder of this section we devise an efficient distributed algorithm for computing the approximate pivots and the trees built from approximate clusters, and show the following.

Theorem 4 *Let $G = (V, E)$ be a weighted graph with n vertices and hop-diameter D, and let $k \geq 1$ be an integer. Set $\epsilon = 1/(48k^4)$. Then there is a randomized distributed algorithm that whp computes all approximate pivots and approximate clusters (with respect to ϵ) within $(n^{1/2+1/k} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds.*[9]

Computing pivots We first compute the pivots for $0 \leq i \leq \lceil k/2 \rceil$. For these values of i we can compute the exact pivots. We conduct $4n^{i/k} \cdot \ln n$ iterations of Bellman–Ford rooted in the vertex set A_i. As a result, every $v \in V$ learns the exact value $\hat{d}_i(v) = d_G(v, A_i)$ and a pivot $\hat{z}_i(v) \in A_i$. Indeed, for any $v \in V$, if $u \in A_i$ is a vertex such that $d_G(v, u) = d_G(v, A_i)$, then Claim 3 implies that $h(v, u) \leq 4n^{i/k} \cdot \ln n$, so the exploration will detect this shortest path. As every message consists of $O(1)$ words (every vertex sends to its neighbors the name of the vertex in A_i and the current distance to it), the total number of rounds is $\sum_{i=0}^{\lceil k/2 \rceil} O(n^{i/k} \cdot \ln n) \leq \tilde{O}(n^{1/2+1/(2k)})$.

For $\lceil k/2 \rceil < i \leq k - 1$ we can only compute *approximate* pivots $\hat{z}_i(v)$ for each $v \in V$. For each such i, apply Theorem 3 with root set A_i and the parameter ϵ (indeed by Claim 3, $|A_i| \leq 4n^{1-(\lceil k/2 \rceil+1)/k} \ln n \leq 2\sqrt{n} \ln n$, and $\epsilon = \Omega(1/k^4) \geq \Omega(1/\log^4 n)$). This will take $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds. At the end, every vertex $v \in V$ will know its approximate pivot $\hat{z}_i(v)$, and the (approximate) distance $\hat{d}_i(v)$, as returned by the algorithm. By (5), $\hat{z}_i(v)$ satisfies the requirement from an approximate pivot [see (7)].

3.2 Building the small trees

For $0 \leq i < \lceil k/2 \rceil$, we can compute the trees $C(u)$ corresponding to the actual clusters. We need to find such a tree for every $u \in A_i \backslash A_{i+1}$, and it is done in the following manner. For each such u in parallel, we initiate a bounded-depth Bellman–Ford exploration for $4n^{(i+1)/k} \ln n$ iterations. This means that each $v \in V$ that receives a message originated at u, and computes that its (current) distance to u is $b_v(u)$, will join $C(u)$ and broadcast the message to its neighbors in G iff

$$b_v(u) < d_G(v, A_{i+1}). \tag{11}$$

(Recall that for $i \leq \lceil k/2 \rceil$, each vertex stores the distance to the exact i-th pivot $\hat{d}_i(v) = d_G(v, A_i)$.) The vertex v will also store the name of its parent in $C(u)$, the neighbor $p \in V$ that sent v the message which last updated $b_v(u)$.

We now argue that if $v \in C(u)$, then v will surely receive a message from u and will have $b_v(u) = d_G(u, v)$. Let P be the shortest path in G between u and v. Note that every vertex y on P has $y \in C(u)$, because

$$\begin{aligned} d_G(y, u) = d_G(v, u) - d_G(v, y) &\overset{(6)}{<} d_G(v, A_{i+1}) - d_G(v, y) \\ &\leq d_G(y, A_{i+1}). \end{aligned}$$

It follows by a simple induction that every such y will receive a message with the exact distance $b_y(u) = d_G(y, u)$ and thus will send it onwards, after at most $h(u, y)$ steps of the algorithm. In particular, distances to the root u in $C(u)$ are preserved exactly. Corollary 4 asserts that for all $v \in C(u)$ we have that $h(u, v) \leq 4n^{(i+1)/k} \ln n$. So there are enough Bellman–Ford iterations to reach all vertices of $C(u)$.

The middle level When k is odd, the level $i = (k - 1)/2$ induces a relatively large running time $\tilde{O}(n^{1/2+3/(2k)})$ (see the upcoming paragraph on running-time analysis), if one uses the algorithm that was described above. To overcome this, we use a different method for this level. We apply Theorem 1 on the set of sources $S = A_i \backslash A_{i+1}$, with $B = 4n^{(i+1)/k} \cdot \ln n$ and ϵ, so that each vertex $v \in V$ will get a distance estimate $b_v(u)$ for each $u \in S$. Indeed, if $v \in C(u)$ then by Corollary 4, $h(u, v) \leq B$, so that the distance estimate returned by the theorem is a $1 + \epsilon$ approximation to $d_G(u, v) = d_G^{(B)}(u, v)$.

We say that v joins the (approximate) cluster $\tilde{C}(u)$ of $u \in S$ if the following holds

$$b_v(u) < d_G(v, A_{i+1}),$$

(recall that v knows the exact distance to its $i+1 = (k+1)/2$-pivot). The parent p of v in the tree induced by $\tilde{C}(u)$ will be the parent given by Remark 1. We show that this p will join $\tilde{C}(u)$ as well. This holds because

$$\begin{aligned} b_p(u) &\overset{(3)}{\leq} b_v(u) - w(v, p) < d_G(v, A_{i+1}) - d_G(v, p) \\ &\leq d_G(p, A_{i+1}). \end{aligned}$$

Finally, we note that this is an approximate cluster; since $d_G(u, v) \leq b_v(u)$ it follows that $\tilde{C}(u) \subseteq C(u)$, while if $v \in C_\epsilon(u)$ then

$$b_v(u) \overset{(2)}{\leq} (1 + \epsilon) d_G(u, v) \overset{(8)}{<} d_G(v, A_{i+1}),$$

so $\tilde{C}(u) \supseteq C_\epsilon(u)$, satisfying (9). (We remark that the middle level is the only one in which one may use Theorem 1. In all

[9] For odd k the number of rounds becomes $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$.

other levels, either the number of sources $|A_i| \approx n^{1-i/k}$ or the required depth $B \approx n^{(i+1)/k}$ will be larger than $n^{1/2+1/k}$.)

Running time By Claim 2, every vertex can belong to at most $\tilde{O}(n^{1/k})$ clusters. Hence, the congestion at every Bellman–Ford iteration is at most $\tilde{O}(n^{1/k})$. Thus the number of rounds required to implement each of the $4n^{(i+1)/k} \ln n$ iterations of Bellman–Ford is $\tilde{O}(n^{1/k})$. When k is even, the total running time is $\sum_{i=0}^{k/2-1} \tilde{O}(n^{(i+2)/k}) = \tilde{O}(n^{1/2+1/k})$. When k is odd, the middle level $(k-1)/2$ will take time $\tilde{O}(|S| + B + D) = \tilde{O}(n^{1/2+1/(2k)} + D)$, while the lower levels will take $\sum_{i=0}^{(k-3)/2} \tilde{O}(n^{(i+2)/k}) = \tilde{O}(n^{1/2+1/(2k)})$. So for odd k, the total running time is $\tilde{O}(n^{1/2+1/(2k)} + D)$.

3.3 Building the large trees

Building the trees $\tilde{C}(u)$ for $u \in A_i \backslash A_{i+1}$ when $i \geq \lceil k/2 \rceil$ is more involved, since the number of iterations for the simple Bellman–Ford style approach grows like $\approx n^{(i+2)/k}$. We will use the fact that there are only few vertices in A_i, and divide the computation into two phases. In the first phase we compute virtual trees only on $\approx\sqrt{n}$ vertices, and in the second phase we extend the trees to the entire graph. Before we turn to the two-phase construction, we describe the preprocessing stage, in which we build structures that are later used in both phases.

3.3.1 Preprocessing

Let $V' = A_{\lceil k/2 \rceil}$, and set $B = 4n/\mathbb{E}[|V'|] \cdot \ln n$. That is, for even k we set $B = 4n^{1/2} \cdot \ln n$, while for odd k, $B = 4n^{1/2+1/(2k)} \cdot \ln n$. Apply Theorem 1 to G with the set V' and parameters B and $\epsilon/2$. By Claim 3 we may assume $|V'| \leq 4n^{1/2} \ln n$, and since $1/\epsilon \leq 48 \log^4 n$, the number of rounds required is whp $\tilde{O}(n^{1/2+1/(2k)} + D)$. From now on assume that (2) indeed holds (with ϵ replaced by $\epsilon/2$). This happens whp. Let $G' = (V', E', w')$ be a (virtual) graph on G, and for each $u, v \in V'$ with $d_{uv} < \infty$, set the weight of the edge connecting them to be $w'(u, v) = d_{uv}$ (where d_{uv} is the value computed in Theorem 1). Following [25], it can be shown that

Claim 5 *For any $u, v \in V'$,*

$$d_G(u, v) \leq d_{G'}(u, v) \leq (1 + \epsilon/2) d_G(u, v). \tag{12}$$

Proof The left hand side of (12) holds since if $u = v_0, v_1, \ldots, v_l = v$ is the shortest-path in G' from u to v, then

$$d_{G'}(u, v) = \sum_{i=0}^{l-1} d_{v_i v_{i+1}} \overset{(2)}{\geq} \sum_{i=0}^{l-1} d_G^B(v_i, v_{i+1})$$
$$\geq \sum_{i=0}^{l-1} d_G(v_i, v_{i+1}) \geq d_G(u, v).$$

To see the right hand side of (12): if $h(u, v) \leq B$ then (2) implies that $d_{G'}(u, v) \leq d_{uv} \leq (1 + \epsilon/2) d_G^B(u, v) = (1 + \epsilon/2) d_G(u, v)$. Otherwise, consider the shortest-path from u to v in G. By Claim 3, there exist vertices $u = u_0, u_1, \ldots, u_l = v$ on this path such that $u_i \in V'$ and $h(u_i, u_{i+1}) \leq B$ for all $0 \leq i < l$.[10] Now, using the triangle inequality and (2) again,

$$\begin{aligned} d_{G'}(u, v) &\leq \sum_{i=0}^{l-1} d_{u_i u_{i+1}} \overset{(2)}{\leq} (1 + \epsilon/2) \sum_{i=0}^{l-1} d_G^B(u_i, u_{i+1}) \\ &= (1 + \epsilon/2) \sum_{i=0}^{l-1} d_G(u_i, u_{i+1}) \\ &= (1 + \epsilon/2) d_G(u, v). \end{aligned}$$ □

Apply Theorem 2 on G' with parameters $\epsilon/3$ and $\rho = \max\{1/k, \log\log n/\sqrt{\log n}\}$. We obtain a $(\beta, \epsilon/3)$-hopset F with $\beta = \min\{2^{\tilde{O}(\sqrt{\log n})}, (\log n)^{O(k)}\}$. The number of rounds required is $\tilde{O}(|V'|^{1+\rho} + D) \cdot \beta^2 = (n^{1/2(1+1/k)} + D) \cdot \min\{2^{\tilde{O}(\sqrt{\log n})}, (\log n)^{O(k)}\}$.

Let $G'' = (V', E' \cup F, w'')$ be the graph obtained from G' by adding all the hopset edges. (Note that some edges may have their weight replaced. In the case of conflict, the weights w'' agree with the weights of the hopset F.) By (4) and (12) we have that G'' is indeed a virtual graph since $d_{G''}(u, v) \geq d_{G'}(u, v) \geq d_G(u, v)$. On the other hand,

$$\begin{aligned} d_{G''}^{(\beta)}(u, v) &\leq (1 + \epsilon/3) d_{G'}(u, v) \\ &\leq (1 + \epsilon/2)(1 + \epsilon/3) d_G(u, v) \\ &\leq (1 + \epsilon) d_G(u, v). \end{aligned}$$

We conclude that the graph G'' satisfies the following property: for every $u, v \in V'$,

$$d_G(u, v) \leq d_{G''}^{(\beta)}(u, v) \leq (1 + \epsilon) d_G(u, v). \tag{13}$$

3.3.2 Construction

Fix $\lceil k/2 \rceil \leq i \leq k - 1$. Recall that every $v \in V'$ knows of the hopset edges touching it, and every $u \in V$ knows the approximate B-limited distances to the vertices of V'. We build the trees $\tilde{C}(u)$ for all $u \in A_i \backslash A_{i+1}$ in parallel, in two main phases.

[10] The claim guarantees the existence of u_1, but we may apply it on the pair u_1, v as well (since the shortest-path between them is a subset of the u to v shortest-path) to obtain u_2, and so on.

Phase 1 For each such u, conduct β iterations of depth-bounded Bellman–Ford in the graph G''.[11] (Since this is a virtual graph, all the messages will be collected at the root of some BFS tree of G via pipelined convergecast, and then broadcasted to the entire graph G via pipelined broadcast. See Lemma 1.) If $v \in V'$ receives a message that originated at u with (current) distance to u which is $b_v(u)$, it will join the approximate cluster of u and forward the message to its neighbors in G'' iff

$$b_v(u) < \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^3}. \tag{14}$$

[Recall that $\hat{d}_{i+1}(v)$ is the approximate distance from v to the its (approximate) level $i+1$ pivot.] The vertex v will also store its *virtual* parent, the neighbor $p \in V'$ that sent v the message which last updated $b_v(u)$. For each $u \in A_i \backslash A_{i+1}$, we have a (virtual) tree $\tilde{C}'(u)$ on the vertices of V' that received a message originated at u and satisfy (14).

Phase 1.5 The purpose of this step is to guarantee that every vertex which was added to the (virtual) tree being built for some $u \in A_i \backslash A_{i+1}$, will have an appropriate parent in G (through which it will route later on). The issue is that hopset edges are not equipped with parents in G, unlike the edges of G', for which Remark 1 provides parents. We deal with this by using the path-reporting property of hopset edges—each such edge is realized by a path in G', so we ensure the vertices of this path join the tree as well, and set parents accordingly. We now describe this formally.

When the first phase ends after β iterations, for every hopset edge $(x, y) \in F$ such that x is the virtual parent of y we do the following. Let P be the path in G' realizing this edge. Each $v \in V'(P)\backslash\{x\}$ that has $b_v(u)$ value (for some $u \in A_i \backslash A_{i+1}$) at least $b_x(u) + d_P(x, v)$, updates its distance estimate to be $b_v(u) = b_x(u) + d_P(x, v)$, joins $\tilde{C}'(u)$ (if it hasn't already), and sets its virtual parent as v', where v' is the neighbor of v on P closer to x (recall Property 1, which guarantees that v knows the relevant information).

Finally, set the *real* parents: for each vertex $v \in \tilde{C}'(u)$ with a virtual parent v', set $p(v) = p_{v'}(v)$ (see Remark 1 for the definition and computation of $p_{v'}(v)$). Recall that (v, v') is a virtual edge (of the graph G'), while $(v, p(v))$ is a "real" edge from G.

Phase 2 Here we extend each virtual tree $\tilde{C}'(u)$ to the vertices of V. For all $u \in A_i \backslash A_{i+1}$, every vertex $v \in \tilde{C}'(u)$ broadcasts to the entire graph its value $b_v(u)$ (and the name of u). A vertex $y \in V$ will add itself to $\tilde{C}(u)$ if

$$d_{yv} + b_v(u) < \frac{\hat{d}_{i+1}(y)}{1+\epsilon}, \tag{15}$$

where d_{yv} is the value computed in Theorem 1. Also, y will set $p(y) = p_v(y)$ as its (real) parent in $\tilde{C}(u)$ for the v minimizing $b_y(u) = d_{yv} + b_v(u)$ (breaking ties arbitrarily). We remark that the condition of (15) is less stringent than that of (14). That is, vertices of V' that did not join $\tilde{C}'(u)$, may now be included in $\tilde{C}(u)$.

Claim 6 *For any $u \in A_i \backslash A_{i+1}$, the vertices $v \in V'$ added to $\tilde{C}'(u)$ in phase 1.5 with distance estimate $b_v(u)$ satisfy the following:*

$$b_v(u) < \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^2}. \tag{16}$$

Proof To see this, let $(x, y) \in F$ be the hop-set edge which triggered the addition of v to $\tilde{C}'(u)$ at phase 1.5, and let P be the path in G' realizing this edge, then

$$\begin{aligned} b_v(u) &\le d_P(x, v) + b_x(u) = d_P(x, y) - d_P(v, y) + b_x(u) \\ &= b_y(u) - d_P(v, y). \end{aligned}$$

It follows that

$$\begin{aligned} b_v(u) &\le b_y(u) - d_P(v, y) \overset{(14)}{<} \frac{\hat{d}_{i+1}(y)}{(1+\epsilon)^3} - d_G(v, y) \\ &\overset{(5)}{\le} \frac{d_G(y, A_{i+1}) - d_G(v, y)}{(1+\epsilon)^2} \\ &\le \frac{d_G(v, A_{i+1})}{(1+\epsilon)^2} \overset{(5)}{\le} \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^2}. \end{aligned}$$

□

The next lemma asserts that the values $b_v(u)$ approximate well the distances to the root u of the virtual tree.

Lemma 7 *For any $u \in A_i \backslash A_{i+1}$ and $v \in \tilde{C}(u)$ with the corresponding value $b_v(u)$, we have that*

$$d_G(u, v) \le b_v(u) \le (1+\epsilon)^4 d_G(u, v). \tag{17}$$

Proof We consider 3 cases according to the phase in which v joins $\tilde{C}(u)$.

Case 1 First we prove for $v \in \tilde{C}'(u)$ added at phase 1. Note that the left hand side of (17) can be verified by induction on the iteration in which $b_v(u)$ was last updated. The base case $u = v$ clearly holds. Assume it holds for v' (the virtual parent of v). Recall that w'' is the weight function in G''. We have

$$\begin{aligned} b_v(u) &= w''(v, v') + b_{v'}(u) \\ &\ge d_{G''}(v, v') + d_G(u, v') \overset{(13)}{\ge} d_G(u, v). \end{aligned}$$

We now turn to the right hand side of (17). Seeking contradiction, assume

$$b_v(u) > (1+\epsilon)^4 d_G(u, v). \tag{18}$$

[11] See (14) below for the required condition on depth.

Let P be the shortest β-hops path in G'' from u to v, and we will show (by induction) that every vertex z on P, which lies h hops from u, must join $\tilde{C}'(u)$ with value $b_z(u) \le d_P(u,z)$ by the iteration h of the Bellman–Ford exploration of phase 1. The base case for $z=u$ clearly holds. Fix any other $z \in P$ with h hops to u on P, and assume it holds for p, the neighbor of z on P (the one closer to u), so we have that $b_p(u) \le d_P(u,p)$ by iteration $h-1$. At iteration h, p will broadcast its value $b_p(u)$, and thus z could have updated its value to be $b_p(u) + w''(p,z)$. In particular,

$$\begin{aligned} b_z(u) &\le b_p(u) + w''(p,z) \\ &\le d_P(u,p) + w''(p,z) = d_P(u,z). \end{aligned} \tag{19}$$

We now argue $b_z(u)$ satisfies (14), which would cause z to join $\tilde{C}'(u)$,

$$\begin{aligned} b_z(u) &\overset{(19)}{\le} d_P(u,z) \\ &= d_P(u,v) - d_P(v,z) \\ &\le d^{(\beta)}_{G''}(u,v) - d_G(v,z) && (20) \\ &\overset{(13)}{\le} (1+\epsilon)d_G(u,v) - d_G(v,z) \\ &\overset{(18)}{\le} \frac{b_v(u)}{(1+\epsilon)^2} - d_G(v,z) && (21) \\ &\overset{(14)}{<} \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^4} - d_G(v,z) \\ &\overset{(5)}{\le} \frac{d_G(v,A_{i+1}) - d_G(v,z)}{(1+\epsilon)^3} \\ &\le \frac{d_G(z,A_{i+1})}{(1+\epsilon)^3} \\ &\overset{(5)}{\le} \frac{\hat{d}_{i+1}(z)}{(1+\epsilon)^3}, && (22) \end{aligned}$$

where (20) uses that P is the shortest β-hops path in G'', and (21) uses the contradiction assumption (18) (note that it was used with the term $(1+\epsilon)^3$ rather than $(1+\epsilon)^4$). Hence z joins $\tilde{C}'(u)$, and so $b_v(u) \le d_P(u,v)$. Hence

$$b_v(u) \le d_P(u,v) = d^{(\beta)}_{G''}(u,v) \overset{(13)}{\le} (1+\epsilon)d_G(u,v),$$

which contradicts our assumption that (17) does not hold.

Case 1.5 We now turn to vertices $v \in \tilde{C}'(u)$ who joined in phase 1.5. The left hand side holds since if $(x,y) \in F$ is the hop-set edge that triggered the addition of v, and P' is the path in G' realizing this edge, we have that $b_v(u) = d_{P'}(v,x) + b_x(u) \ge d_G(v,x) + d_G(x,u) \ge d_G(v,u)$. For the right hand side, note that we only used the fact that v joined in phase 1 at (22), so we can repeat the argument, replacing the use of (14) by (16). We indeed lose a factor of $1+\epsilon$, but the inequality is still valid, yielding the same contradiction.

Case 2 Finally, we turn to $v \in \tilde{C}(u)$ joining at phase 2. Note that for each such v, there exists some $x \in V'$ for which v sets its value to be $b_v(u) = d_{vx} + b_x(u) \ge d_G(v,x) + d_G(x,u) \ge d_G(v,u)$, which proves the left hand side of (17). For the right hand side, we consider two subcases.

Subcase a Consider first the case that $h(v,u) \le B$. Since v could update $b_v(u)$ directly from the broadcast of u itself, we have

$$b_v(u) \le 0 + d_{vu} \overset{(2)}{\le} (1+\epsilon)d^{(B)}_G(v,u) = (1+\epsilon)d_G(v,u),$$

as required.

Subcase b The other case is when $h(v,u) > B$, but then Claim 3 (with $i = \lceil k/2 \rceil$) implies that there exists $x \in V'$ on the shortest path in G from v to u, with $h(v,x) \le B$. In particular, $d^{(B)}_G(x,v) = d_G(x,v)$. Again seeking contradiction, assume (17) does not hold for v. Let P be the shortest (at most) β-hops path from u to x in G''. We claim that every $z \in P$ must have joined $\tilde{C}'(u)$ at phase 1. To see this by induction, fix $z \in P$ with h hops from u on P, and assume p (the neighbor of z closer to u) did join by the $h-1$ iteration of Bellman–Ford, with $b_p(u) \le d_P(u,p)$. When p broadcasts $b_p(u)$ at step h, then indeed $b_z(u) \le b_p(u) + w''(p,z) = d_P(u,z)$. Now,

$$\begin{aligned} b_z(u) &\le d_P(u,z) \\ &\le d^{(\beta)}_{G''}(u,x) - d_P(z,x) \\ &\overset{(13)}{\le} (1+\epsilon)d_G(u,x) - d_G(z,x) \\ &= (1+\epsilon)[d_G(u,v) - d_G(x,v)] - d_G(z,x) \\ &\le \frac{b_v(u)}{(1+\epsilon)^3} - d_G(x,v) - d_G(z,x) \\ &\overset{(15)}{<} \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^4} - d_G(x,v) - d_G(z,x) \\ &\overset{(7)}{\le} \frac{d_G(v,A_{i+1}) - d_G(x,v) - d_G(z,x)}{(1+\epsilon)^3} \\ &\le \frac{d_G(z,A_{i+1})}{(1+\epsilon)^3} \\ &\le \frac{\hat{d}_{i+1}(z)}{(1+\epsilon)^3}. \end{aligned} \tag{23}$$

[(23) is because x lies on the shortest $u-v$ path in G.]

This implies $b_z(u)$ satisfies (14) and thus z indeed joins $\tilde{C}'(u)$ by iteration h of phase 1. In particular, x joins by the end of phase 1, and broadcasts $b_x(u)$ at phase 2. We already proved that x satisfies (17), so we have that

$$\begin{aligned} b_v(u) &\le b_x(u) + d_{xv} \le (1+\epsilon)^4 d_G(u,x) + (1+\epsilon)d^B_G(x,v) \\ &\le (1+\epsilon)^4 d_G(u,v), \end{aligned}$$

(Recall that d_{xv} is the value computed by the algorithm of Theorem 1.) This yields a contradiction to (18) and concludes the proof. □

The following lemma shows that the sets $\tilde{C}(u)$ satisfy the requirement from approximate clusters. The proof is similar to that of Lemma 7, though it uses the definition of $C_\epsilon(u)$, rather than the (contradiction) assumption that $b_v(u)$ is large.

Lemma 8 *For any $u \in A_i \backslash A_{i+1}$, the set $\tilde{C}(u)$ satisfies (9).*

Proof For the right hand side of (9), note that if $v \in \tilde{C}'(u)$, then

$$d_G(u,v) \overset{(17)}{\leq} b_v(u) \overset{(14)\wedge(16)}{<} \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^2} \overset{(5)}{\leq} d_G(v, A_{i+1}),$$

so $v \in C(u)$ as well. For the left hand side of (9) (at this point we only show that $\tilde{C}'(u) \supseteq C_{6\epsilon}(u) \cap V'$), consider $v \in C_{6\epsilon}(u) \cap V'$, and let P be the (at most) β-hops shortest path from v to u in G''. It suffices to show that every vertex y along this path which is h hops from u will join $\tilde{C}'(u)$ and have $b_y(u) \leq d_P(y,u)$ by the iteration h of Bellman–Ford in phase 1. Assume (by induction) that p, the predecessor of y on P, joins $\tilde{C}'(u)$ and satisfies $b_p(u) \leq d_P(p,u)$ by iteration $h-1$. Thus, p sends at iteration h the value $b_p(u)$. Since $b_y(u) \leq w''(y,p) + b_p(u) \leq w''(y,p) + d_P(u,p) = d_P(u,y)$, it remains to show that this value of $b_y(u)$ satisfies (14), and thus y joins $\tilde{C}'(u)$. To this end,

$$\begin{aligned}
b_y(u) &\leq d_P(u,y)\\
&\leq d_{G''}^{(\beta)}(u,v) - d_P(y,v)\\
&\overset{(13)}{\leq} (1+\epsilon)d_G(u,v) - d_G(y,v)\\
&\leq \frac{(1+\epsilon)d_G(v,A_{i+1})}{1+6\epsilon} - d_G(y,v)\\
&< \frac{d_G(v,A_{i+1}) - d_G(y,v)}{(1+\epsilon)^3}\\
&\leq \frac{d_G(y,A_{i+1})}{(1+\epsilon)^3}\\
&\overset{(5)}{\leq} \frac{\hat{d}_{i+1}(y)}{(1+\epsilon)^3}.
\end{aligned}$$

where the fourth inequality uses that $v \in C_{6\epsilon}(u)$ (recall (8)). This implies v will join $\tilde{C}'(u)$ in phase 1.

We now prove that (9) holds for $\tilde{C}(u)$. For the right hand side, let $y \in \tilde{C}(u) \backslash \tilde{C}'(u)$, then there exists $v \in V'$ for which y satisfies (15). So we obtain

$$\begin{aligned}
d_G(y,u) &\leq d_G(y,v) + d_G(v,u) \overset{(2)\wedge(17)}{\leq} d_{yv} + b_v(u)\\
&\overset{(15)}{<} \frac{\hat{d}_{i+1}(y)}{1+\epsilon} \overset{(5)}{\leq} d_G(y, A_{i+1}).
\end{aligned}$$

This implies that $y \in C(u)$. For the left hand side of (9), assume $y \in C_{6\epsilon}(u)$. Consider first the case that $h(u,y) \leq B$. Then when u broadcasts $b_u(u) = 0$ at phase 2, y will add itself to $\tilde{C}(u)$ because

$$\begin{aligned}
d_{yu} + 0 &\overset{(2)}{\leq} (1+\epsilon)d_G^{(B)}(y,u) = (1+\epsilon)d_G(y,u)\\
&\overset{(8)}{\leq} \frac{1+\epsilon}{1+6\epsilon} \cdot d_G(y, A_{i+1}) \overset{(5)}{<} \frac{\hat{d}_{i+1}(y)}{1+\epsilon}. \qquad (24)
\end{aligned}$$

The other case is that $h(y,u) > B$. Then by Claim 3 there is a vertex $v \in V'$ on the shortest path from y to u so that $h(y,v) \leq B$. We now argue that $v \in \tilde{C}'(u)$, by a similar (though slightly more involved) argument as above. To see this, consider the shortest path P with (at most) β-hops in G'' from u to v, and we claim that each vertex z on this path with h hops from u, will join $\tilde{C}'(u)$ with $b_z(u) \leq d_P(u,z)$ by iteration h of the Bellman–Ford of phase 1. Again by induction, at step h the vertex z heard $b_p(u) \leq d_P(u,p)$ from its predecessor p on P. Then indeed $b_z(u) \leq b_p(u) + w''(p,z) \leq d_P(u,z)$. Now we show that z joins $\tilde{C}'(u)$.

$$\begin{aligned}
b_z(u) &\leq d_P(u,z)\\
&= d_{G''}^{(\beta)}(u,v) - d_P(z,v)\\
&\overset{(13)}{\leq} (1+\epsilon)d_G(u,v) - d_G(z,v)\\
&= (1+\epsilon)[d_G(u,y) - d_G(y,v)] - d_G(z,v) \qquad (25)\\
&\leq \frac{(1+\epsilon)d_G(y,A_{i+1})}{1+6\epsilon} - d_G(y,v) - d_G(z,v)\\
&\leq \frac{d_G(y,A_{i+1}) - d_G(y,v) - d_G(z,v)}{(1+\epsilon)^3}\\
&\leq \frac{d_G(z,A_{i+1})}{(1+\epsilon)^3}\\
&\overset{(5)}{\leq} \frac{\hat{d}_{i+1}(z)}{(1+\epsilon)^3}, \qquad (26)
\end{aligned}$$

where (25) uses that v is on the shortest path in G from u to y, and (26) uses that $y \in C_{6\epsilon}(u)$. In particular, we have shown $v \in \tilde{C}'(u)$ by the end of phase 1. It follows that v will broadcast the value $b_v(u) \leq d_{G''}^{(\beta)}(u,v)$ in the second phase. Since $h(y,v) \leq B$,

$$\begin{aligned}
b_y(u) &\leq d_{yv} + b_v(u)\\
&\overset{(2)}{\leq} (1+\epsilon)d_G^B(y,v) + d_{G''}^{(\beta)}(u,v)\\
&\overset{(13)}{\leq} (1+\epsilon)[d_G(y,v) + d_G(u,v)]\\
&= (1+\epsilon)d_G(y,u)
\end{aligned}$$

$$\begin{aligned}&\overset{(8)}{\le} \frac{1+\epsilon}{1+6\epsilon}\cdot d_G(y, A_{i+1})\\ &< \frac{\hat{d}_{i+1}(y)}{1+\epsilon}.\end{aligned}$$

So y will be added to $\tilde{C}(u)$. This concludes the proof of the lemma. □

Our next goal is to argue that the parent setting ensures that root-vertex distances in each cluster tree satisfy (10), i.e., are approximated up to a factor $(1+\epsilon)^4$. It suffices to prove the following claim.

Claim 9 *For any $u \in A_i \backslash A_{i+1}$, and any $v \in \tilde{C}(u)$, if $p = p(v)$ is the (real) parent of v with corresponding value $b_p(u)$, then $p \in \tilde{C}(u)$ and*

$$b_v(u) \ge w(v, p) + b_p(u). \tag{27}$$

Once this claim is established, we get by induction on the depth of the tree that $d_{\tilde{C}(u)}(u, v) \le b_v(u)$. The base case when $u = v$ clearly holds. Assume for $p = p(v)$ that $d_{\tilde{C}(u)}(u, p) \le b_p(u)$, and now

$$\begin{aligned}d_{\tilde{C}(u)}(u, v) &= w(v, p) + d_{\tilde{C}(u)}(u, p)\\ &\le w(v, p) + b_p(u) \overset{(27)}{\le} b_v(u).\end{aligned}$$

Combining this with Lemma 7 establishes (10).

Proof of Claim 9 Consider first the case that $v \in \tilde{C}'(u)$, and there are two sub-cases to consider. In the first sub-case, v updated $b_v(u)$ in phase 1 from some $x \in \tilde{C}'(u)$, who sent $b_x(u)$ over the (virtual) edge $(x, v) \in E'$ (which is not a hop-set edge). Then by the definition of G', $b_v(u) = w'(x, v) + b_x(u) = d_{xv} + b_x(u)$, the virtual parent of v is set to x, and the real parent is thus $p = p_x(v)$. Since p receives a message from x in the second phase, it sets $b_p(u)$ to at most $d_{px} + b_x(u)$. It follows that

$$\begin{aligned}b_p(u) &\le d_{px} + b_x(u) \overset{(3)}{\le} d_{vx} - w(v, p) + b_x(u)\\ &= b_v(u) - w(v, p),\end{aligned} \tag{28}$$

which satisfies (27). But we must also argue that p indeed joins the tree $\tilde{C}(u)$. Here we use the relaxed condition of (15) [compared to (14)], and obtain that

$$b_p(u) \overset{(28)}{\le} b_v(u) - w(v, p) \tag{29}$$

$$\overset{(14)}{<} \frac{\hat{d}_{i+1}(v)}{(1+\epsilon)^3} - d_G(v, p) \tag{30}$$

$$\begin{aligned}&\overset{(5)}{\le} \frac{d_G(v, A_{i+1}) - d_G(v, p)}{1+\epsilon}\\ &\le \frac{d_G(p, A_{i+1})}{1+\epsilon}\\ &\le \frac{\hat{d}_{i+1}(p)}{1+\epsilon},\end{aligned} \tag{31}$$

which satisfies (15).

The second sub-case is that v updated $b_v(u)$ in phase 1 or 1.5 due to some hop-set edge $(x, y) \in F$, so that v lies on the path P in G' realizing this edge (it could be that $y = v$, if it happened in phase 1). We set $b_v(u) = b_x(u) + d_P(x, v)$, and the virtual parent of v is $v' \in V'$, its neighbor on P which is closer to x. Recall that in G', the weight $w'(v, v') = d_{vv'}$, so that

$$d_P(x, v) = d_P(x, v') + d_{vv'}. \tag{32}$$

The real parent of v is set as $p = p_{v'}(v)$. Since v' broadcasts in phase 2 its estimate $b_{v'}(u) \le b_x(u) + d_P(x, v')$, it follows that

$$\begin{aligned}b_p(u) &\le d_{pv'} + b_{v'}(u)\\ &\overset{(3)}{\le} (d_{vv'} - w(v, p)) + (b_x(u) + d_P(x, v'))\\ &\overset{(32)}{=} b_x(u) + d_P(v, x) - w(v, p)\\ &= b_v(u) - w(v, p),\end{aligned}$$

as required in (27). Again, to see that $p \in \tilde{C}(u)$, we repeat the calculation of (29) with one change: in (30), replace the use of (14) by (16), which will have the factor of $(1+\epsilon)^3$ replaced by $(1+\epsilon)^2$, but this suffices to satisfy (31).

We turn to the case that $v \in \tilde{C}(u) \backslash \tilde{C}'(u)$. Let $x \in \tilde{C}'(u)$ be the vertex which broadcasts in phase 2 a value $b_x(u)$ minimizing $b_v(u) = d_{vx} + b_x(u)$. The parent of v is thus set to be $p = p_x(v)$, and now

$$\begin{aligned}b_p(u) &\le d_{px} + b_x(u) \overset{(3)}{\le} d_{vx} - w(v, p) + b_x(u)\\ &= b_v(u) - w(v, p),\end{aligned}$$

The proof that $p \in \tilde{C}(u)$ is again similar to (29). □

Running time We noted that the number of rounds required for the preprocessing is $\tilde{O}(n^{1/2+1/(2k)} + D)\cdot \min\{2^{\tilde{O}(\sqrt{\log n})}, (\log n)^{O(k)}\}$. Since by (9) we have $\tilde{C}'(u) \subseteq C(u)$, then Remark 2 implies that $v \in V'$ sends at most $\tilde{O}(n^{1/k})$ distance estimates $b_v(\cdot)$. As $|V'| \le \tilde{O}(n^{1/2})$, by Lemma 1, implementing a single Bellman–Ford iteration will take $\tilde{O}(n^{1/2+1/k} + D)$ rounds. As there are β iterations in phase 1 (and a single one in phases 1.5 and 2), the total number of rounds is $\tilde{O}(n^{1/2+1/k} + D)\cdot \min\{2^{\tilde{O}(\sqrt{\log n})}, (\log n)^{O(k)}\}$. (For odd k, both $|V'|\cdot n^{1/k}, B \le \tilde{O}(n^{1/2+1/(2k)})$), so we get $\tilde{O}(n^{1/2+1/(2k)} + D)\cdot \min\{2^{\tilde{O}(\sqrt{\log n})}, (\log n)^{O(k)}\}$ rounds.)

4 Distributed tree routing

In this section we present a modification of the (exact) routing scheme of Thorup–Zwick for rooted trees, that can be implemented efficiently in a distributed manner. The price is that the size of the labels and tables increases by a factor of $\log n$, compared to what [33] achieved.

Theorem 5 *Fix a graph $G = (V, E)$ on n vertices with hop-diameter D. For any tree T which is a subgraph of G, there is a routing scheme with stretch 1, routing tables of size $O(\log n)$ and labels of size $O(\log^2 n)$, that can be computed in a distributed manner within $\tilde{O}(\sqrt{n} + D)$ rounds.*

Lemma 10 *If we are given n trees, each a sub-graph of $G = (V, E)$, so that each vertex $v \in V$ participates in at most s trees, then routing schemes for all the trees can be computed in $\tilde{O}(\sqrt{n \cdot s} + D)$ rounds.*

Let us first recall briefly how (a simplified version of) the TZ scheme works. For every non-leaf vertex, define a *heavy child* as the child with the largest subtree. Run a depth first search (DFS) on the tree, each vertex u receives an entry time a_u and exit time b_u. The routing table stored at each vertex u consists of the name and port number of its parent $p(u)$ in the tree, the name (and port) of its heavy child $h(u)$, and the numbers a_u, b_u. The label of a vertex u contains the number a_u and additional $\lceil \log n \rceil$ words: consider the path P from the root to u, for every vertex w on this path such that its heavy child is not on P, we append to the label of u the name of w and the port number leading from w to its child on P. The observation is that whenever the path does not use the heavy child, the size of the subtree shrinks by a factor of at least 2, so this can happen only $\lceil \log n \rceil$ times. In order to route from u to v, every intermediate vertex x does as follows: if $a_x = a_v$ we are done, if $a_v \notin (a_x, b_x)$, we know the DFS did not find v in the subtree rooted at x, so x sends the message to its parent, and if $a_v \in (a_x, b_x)$ then v lies in the subtree of x. In the latter case, x examines the label of v for an entry of the form (x, x'), if it exists it sends to its child x', if not, x sends the message to its heavy child.

In order to obtain a scheme that runs efficiently in a distributed manner, we cannot compute heavy children and run DFS on the entire tree. Instead, we shall apply certain variants of the TZ-scheme in two levels. Let T be a tree on the vertices $V(T) \subseteq V$, rooted at z. For $u \in V(T)$, denote by $p(u)$ the parent of u in T. We assume that every vertex knows the names of its parent and its children. The basic idea is to randomly sample $\gamma \geq c \cdot \ln n$, for a sufficiently large constant c, vertices $U \subseteq V$. (γ here is a parameter.) Each vertex in V chooses itself to U independently with probability $\frac{\gamma}{n}$. Partition the tree T into subtrees according to the vertices of $U(T) = (U \cap V(T)) \cup \{z\}$, by removing each edge from a vertex of $U(T)$ to its parent. Note that this partitions T into a forest F of $|U(T)|$ subtrees, each of these subtrees is rooted at a vertex of $U(T)$. For $w \in U(T)$, denote by T_w the subtree in F rooted in w. Let T' denote the virtual tree on the vertices of $U(T)$, where w is a parent of u in T', if $p(u)$ lies in T_w. We shall devise a routing scheme for each T_w, and a global scheme that routes in T'. We begin by bounding the depth of each subtree; let $B = \frac{4n}{\gamma} \cdot \ln n$.

Claim 11 *With high probability, $|U| = O(\gamma)$, and for each $w \in U(T)$, the tree T_w has depth at most B.*

Proof The first event holds with high probability by a simple Chernoff bound. For the second: by independence, the probability that a path P in T of length B has $P \cap U = \emptyset$, is

$$\left(1 - \frac{\gamma}{n}\right)^{4n/\gamma \ln n} \leq \frac{1}{n^4}.$$

Taking a union bound on the $O(n^2)$ possible paths (in a tree, choosing the path's endpoints determines it) completes the proof. □

Remark Observe that we still have high probability that the events of Claim 11 hold over n different trees of the Thorup–Zwick cover.

From now on assume the events of Claim 11 hold. The assignment has two phases.

Phase 1 In the first phase we compute a routing scheme for each T_w in the forest F, in parallel. In each round, every vertex u that received messages from all its children, sends to its parent in F the size of its subtree (by summing up the sizes of the subtrees of the children of u). By Claim 11, the depth of each tree in F is at most B, and in each round we send one word per vertex. Hence after B rounds every vertex knows the size of its subtree (in F), and in particular, can infer who is its heavy child. Now each $w \in U(T)$ can start a parallel DFS of T_w—that is, every vertex assigns entry and exit times to all if its children in parallel (it is possible since it knows the sizes of every child's subtree). Each vertex in T_w adds to its routing table $(p(x), h(x), a_x, b_x, w)$, which are the name of the parent of x, the heavy child of x, the entry and exit times, and the name w. This computation (parallel DFS) will also require $O(B)$ rounds, since all subtrees work in parallel.

The (local) label assignment for vertices in T_w is done in the following manner. Starting from w (which has empty label), every vertex x that receives a label ℓ from its parent, and has children $x_1, \ldots, x_l$, sends ℓ to its heavy child, and $\ell \circ (x, x_i)$ to x_i for each non-heavy child x_i. The label $\ell(x)$ will consist of a_x and the list ℓ of edges that was given to x.

Phase 2 In the second phase we compute a routing scheme on T'. Every $u \in U(T)$ sends a message to its parent x in T, and receives from x the following message: $\ell(x)$, the name w such that $x \in T_w$ (so that the edge (w, u) should be in T'), and also the port number $e(x, u)$ of x leading to u. Then every such u broadcasts $((w, u), x, \ell(x), e(x, u))$ to the entire graph. Once the root vertex z has full information on T', it may locally compute the TZ routing scheme for T'. The routing table given to $u \in U(T)$ is slightly different than in the usual scheme, as it will contain local routing information for the vertex leading to the heavy child. More formally, the table will be $(h'(u), \ell(y), e(y, h'(u)), a'_u, b'_u)$. Here $h'(u)$ is the name of heavy child of u in T', $y \in T_u$ is the *portal* vertex which is the parent of $h'(u)$ in T, and $e(y, h'(u))$ is the port of y leading to $h'(u)$. Note that z has the name, label and the appropriate port of y when $h'(u)$ reported the edge $(u, h'(u))$. Finally a'_u, b'_u are the entry and exit times of the DFS run by z on T'. Observe that $\ell(y)$ has size $O(\log n)$, and this term dominates the size of a routing table. There are at most $O(\gamma)$ such tables. Hence Lemma 1 implies that we can broadcast to the entire graph all these messages within $O(\gamma \log n + D)$ rounds. In addition, every vertex $u \in U(T)$ sends the routing table given to it to all the vertices in T_u. Since we can send the information inside each subtree in parallel, it will take only $O(B \log n)$ rounds.

The label assignment to the vertices of T' is also modified, since for every possible edge taken in T' which is not leading to a heavy child, we must add the local routing information. Fix $u \in U(T)$. Assume $((v_1, w_1), \ldots, (v_l, w_l))$ is the list of all edges in the path of T' from z to u, so that each w_i is a non-heavy child of v_i. Ordinarily, this list would have been the label of u (along with a'_u). However, in order to be able to route in T', we replace each such edge with $(v_i, w_i, \ell(x_i), e(x_i, w_i))$, where x_i is the parent of w_i in T, $\ell(x_i)$ is the label x_i received in the first phase (for local routing within T_{v_i}), and $e(x_i, w_i)$ is the port leading from x_i to w_i. Recall that z knows the label and appropriate port of every such x_i. Since each $\ell(x_i)$ has size at most $O(\log n)$ words, and $l \le \log n$, we have that the label size is $O(\log^2 n)$. As before, each $u \in U(T)$ propagates this label $\ell'(u)$ to every vertex in T_u. The number of rounds is therefore $O(\gamma \log^2 n + D)$.

Protocol The routing from u to v will be done as follows. Assume we have arrived to an intermediate vertex x that lies in T_w. First x checks if routing in T' is required, by comparing a'_v with a'_x, b'_x (recall that a'_v is part of the label of v, and the routing table of x contains $a'_x = a'_w$ and $b'_x = b'_w$). If $a'_v = a'_x$ then $v \in T_w$, and we proceed to route inside T_w. If $a'_v \notin (a'_x, b'_x)$, we need to route to the subtree rooted at the parent of w in T', and if $a'_v \in (a'_x, b'_x)$ then we need to route to the appropriate child of w in T',

Routing inside T_w This is done exactly as in the TZ scheme, while considering the local routing tables of vertices in T_w and $\ell(v)$. If $a_x = a_v$ we are done. If $a_v \notin (a_x, b_x)$ we route to the parent of x (stored in the local routing table of x), and when $a_v \in (a_x, b_x)$, we inspect $\ell(v)$: if it contains an edge of the form (x, x'), for some x', we route to x'. Otherwise to the heavy child of x (the heavy child's name is also in the local routing table of x).

Routing to the parent of w in T' This is simple, x just routes to its parent, its name is stored in the local routing table of x. Eventually we will reach w (since all vertices in T_w have the same ℓ' label), and route from it to vertex in the tree of w's parent in T'.

Routing to a child of w in T' Here we inspect $\ell'(v)$, if it contains an entry of the form $(w, w', \ell(y), e(y, w'))$ then we know we have to route in T' from w to its child w' in T'. Fortunately, the label $\ell(y)$ provides us the required routing information to route in T_w to the portal vertex y (that has w' as a child in T). From y we go to its child w' using the port $e(y, w')$. If the label $\ell'(v)$ contains no such entry, then we know we need to route to the heavy child of w in T'. Here the label of v is useless, but we stored the label of $y' \in T_w$, the portal vertex which is the parent of this heavy child, in the routing table of each vertex of T_w. Using the label of y' we can route locally in T_w, and from y' route to $h'(w)$ (using the port number for the heavy child stored in the routing table).

When constructing routing tables and labels for one single tree, the overall running time is $O(\gamma \cdot \log^2 n + D) + O(B \cdot \log n) = O(\gamma \cdot \log^2 n + \frac{n}{\gamma} \cdot \log^2 n + D)$, i.e., $O(D + \sqrt{n} \cdot \log^2 n)$, by setting $\gamma = \sqrt{n}$.

Proof of Lemma 10 To avoid high running time, we shall perform the routing tables and labels computations in parallel in all cluster trees, while appending to each message the name of the relevant tree. In the first phase, which can be implemented in $\tilde{O}(\sqrt{n})$ rounds for each tree, we send information on the graph edges (every vertex notifies all its neighbors in each round), so the overhead due to participation in up to s trees is only a factor of s. In the second phase, however, we broadcast messages to the entire graph. So we need a bound on the number of these messages. For each tree T' (which consists of the vertices of U alone) we broadcast two messages per vertex: the first informing the root of its existence, its parent, and the local routing information. In the second message, the root broadcasts routing information and a label for the vertex. Each message is of size $O(\log^2 n)$. By charging these messages to the vertices of U, each such vertex pays for 2 messages per tree containing it. But the number of these trees is at most s, so we need to broadcast at most $\tilde{O}(\sqrt{n} \cdot s)$ words. By Lemma 1, these can be broadcast to the entire graph in $\tilde{O}(\sqrt{n} \cdot s + D)$ rounds.

We next argue that this bound can be further improved to $\tilde{O}(\sqrt{n \cdot s} + D)$, using random start times (see e.g. [12]).

Every root w of a tree T_w in one of the forests F (each cluster tree gives rise to one such a forest) tosses a starting time *start*(w) uniformly at random from the interval $[1, c \cdot \ln n \cdot \sqrt{ns}]$, for a sufficiently large constant c. It then starts broadcasting to vertices of T_w at time $20 \cdot$ *start*(w). [It broadcasts to them the value *start*(w).] Each round of this broadcast is replaced by stages consisting of 20 rounds each. Specifically, a vertex x in T_w that already received the message from its parent tries to deliver it to its children for 20 consecutive rounds. We will show that, whp, for every edge, on one of these rounds no congestion will be experienced. Only when these 20 rounds are over, the children of x will start broadcasting.

Consider a specific edge $e = (x, y)$ in a tree T_w. Let $w_1, w_2, \ldots, w_s$ be the roots of trees T_{w_i} that contain this edge. [Recall that, by Claim 2, whp, $s = O(n^{1/k} \cdot \log n)$.] Let $t_1, t_2, \ldots, t_s$ be the respective hop-distances between w_i and the closer endpoint of e_i to w_i. In other words, for every $i \in [s]$, if w_i broadcasted a message over T_{w_i}, and no other messages would have interfered with its broadcast, then the broadcast of w_i would traverse e_i on step t_i. (For convenience, we number the steps starting from 0.)

For any index R, the probability that the broadcast of w_i will want to traverse e on stage R, conditioned on the assumption that it experienced no congestion whatsoever before that, is the probability that w_i starts broadcasting at stage $R - t_i$, i.e., this is equal to $\mathbb{P}(start(w_i) = R - t_i)$. The latter probability is at most $\frac{1}{c\sqrt{ns}\ln n}$. For a positive integer $\alpha \leq s$, the probability that α cluster trees wish to employ e on stage R, conditioned on the assumption that no congestion was experienced by any of them so far, is at most

$$\left(\frac{1}{c\ln n \cdot \sqrt{ns}}\right)^{\alpha} \cdot \binom{s}{\alpha} \leq \left(\frac{s}{c\ln n \cdot \sqrt{ns}}\right)^{\alpha}$$
$$\leq \left(\frac{1}{n^{1/2-1/(2k)}}\right)^{\alpha}.$$

For $\alpha = 20$, this probability is at most $\frac{1}{n^{10-10/k}} \leq \frac{1}{n^5}$. By union-bound over all stage indices $R \leq n$, and all the $|E| \leq n^2$ edges, we still have an only negligible probability that a congestion was ever experienced throughout the algorithm. (Here we say that a congestion is experienced if a vertex v wishes to broadcast a message m on a stage R of the algorithm through an edge (v, u) incident on v, and v cannot do it for the entire $\alpha = 20$ rounds of this stage, because of other transmissions that employ the same edge.)

Hence, whp, in $O(B \cdot \alpha) + O(\sqrt{ns}\ln n) = \tilde{O}(B + n^{1/2+1/(2k)}\ln n)$ rounds, all broadcasts of the values of starting times will be completed. (Recall that B is an upper bound on the depth of trees T_{w_i}.) This completes Phase 0 of the algorithm.

Now the algorithm proceeds to Phase 1, on which convergecasts are conducted in all these trees. As a result of these convergecasts, every vertex $x \in T_{w_i}$ knows the size of its subtree in T_{w_i}. These convergecasts are conducted by a similar procedure to the one that was described above, i.e., all leaves of T_{w_i} start broadcasting at stage *start*(w_i), and each stage lasts for $\alpha = 20$ rounds. Hence these convergecasts are also completed in $O(B + \sqrt{ns} \cdot \ln n)$ rounds. Then the "parallel DFSs" are conducted in all the trees in parallel by the same procedure of tree broadcast. As a result, all vertices x in these trees T_{w_i} learn their routing tables within T_{w_i}. They also learn their routing labels within additional $O(B\log n + \sqrt{ns}\log^2 n)$ time. [Note that for labels one may need to send messages of size $O(\log n)$ words, and so stages of length $O(\alpha \cdot \log n) = O(\log n)$ are needed.]

Phase 2 is performed in the same way as was already described. Specifically, the algorithm conducts convergecasts of messages $(\ell(x), w, e(x, u))$, where $u \in U(T)$ and x is its parent in T, for some cluster tree T, over the BFS tree τ of the entire graph G. Since every selected vertex u may participate in up to s trees, and there are $O(\gamma)$ selected vertices, this convergecast requires $O(\gamma \cdot s + D)$ time. Analogously, the broadcast of the computed routing tables requires $O(\gamma \cdot s\log n + D)$ time.

Then each $u \in U(T)$ sends its routing table to all vertices of T_u. This is done using the tossed starting times and with stages of α rounds each, as in Phase 1. Hence this step requires $O(B\log n + \sqrt{ns}\log^2 n)$ time. Finally, the labels of selected nodes in T' are broadcasted over the BFS tree τ within additional $O(\gamma \cdot s \cdot \log^2 n + D)$ time.

To summarize, the overall running time of the algorithm is $\tilde{O}(B + D + \sqrt{ns} + \gamma \cdot s) = \tilde{O}(\frac{n}{\gamma} + D + n^{1/2+1/(2k)} + \gamma \cdot s)$. By setting $\gamma = \sqrt{n/s} = \frac{n^{1/2-1/(2k)}}{\sqrt{\log n}}$, we get the running time of $\tilde{O}(\sqrt{ns} + D) = \tilde{O}(n^{1/2+1/(2k)} + D)$. □

5 Routing based on approximate clusters

In this section we show that approximate pivots and approximate clusters suffice for a compact routing scheme, and prove our main result.

Theorem 6 *Let $G = (V, E)$ be a weighted graph with n vertices and hop-diameter D, and let $k \geq 1$ be a parameter. Then there exists a routing scheme with stretch at most $4k - 5 + o(1)$, labels of size $O(k\log^2 n)$ and routing tables of size $O(n^{1/k}\log^2 n)$, that can be computed in a distributed manner within $(n^{1/2+1/k} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds, and for odd k only $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds.*

Construction Apply Theorem 4 on G to obtain approximate pivots and approximate clusters for all vertices. For each $0 \leq i \leq k-1$ and each $u \in A_i \backslash A_{i+1}$, construct the routing scheme for trees given by Theorem 5 on $\tilde{C}(u)$. (We postpone the proof of Theorem 5, i.e., the description of the algorithm that constructs routing tables and labels for each tree, to Sect. 4.) Specifically, in each tree, every vertex stores a table of size $O(\log n)$ and has a label of size $O(\log^2 n)$.

The routing table of each $v \in V$ consists of all the tree-routing tables, for every $u \in V$ such that $v \in \tilde{C}(u)$. The label of v consists of the tree-labels for the (at most) k trees $\tilde{C}(\hat{z}_0(v)), \ldots, \tilde{C}(\hat{z}_{k-1}(v))$, where $\hat{z}_i(v)$ is the approximate i-pivot of v (note that it could be that v does not belong to some of these trees, the label of v will mark these as missing). By Remark 2 there are at most $O(n^{1/k} \log n)$ trees containing v, and as each tree-table is of size $O(\log n)$, the routing table size is as promised. Since each tree-label is of size $O(\log^2 n)$, the label size also obeys the given bound.

Finding a tree Assume we would like to route from vertex u to vertex v. The routing protocol will find a vertex $w = \hat{z}_i(v)$ for some $0 \le i \le k-1$, such that the stretch of the (unique) path from u to v in the tree $\tilde{C}(w)$ is at most $4k - 5 + o(1)$. The algorithm to find such a vertex appears in Algorithm 1.

Algorithm 1 `Find-tree`(u, v)

```
$i \leftarrow 0$;
while $|\{u, v\} \cap \tilde{C}(\hat{z}_i(v))| < 2$ do
    $i \leftarrow i + 1$;
end while
return $\hat{z}_i(v)$;
```

We note that our algorithm differs slightly from that of [33], since it could be the case that v does not belong to the cluster centered at the pivot of v at level i. For this reason we keep searching until we find a cluster containing both u, v.

First we claim that the algorithm is correct. Note that the definition of approximate cluster (9) implies that $\tilde{C}(x) = V$ for every $x \in A_{k-1}$ (this holds since the distance to A_k is defined as ∞). Therefore when $i = k - 1$ it must be that both $u, v \in \tilde{C}(\hat{z}_{k-1}(v))$, and the algorithm indeed halts. The tree $\tilde{C}(w)$ contains both u, v (where $w = \hat{z}_i(v)$ is the vertex returned by the algorithm), by definition. Finally, the information from the label of v indicates which of these trees contain it, and the routing table of u also lists the names of all trees containing it. So we can run the algorithm from u knowing the label of v.

Once u computes the root w, it appends w to the message header along with the label of v. From this point on the header does not change, and we route in the tree $\tilde{C}(w)$. Since this routing is exact, it remains to bound the stretch incurred by using the tree.

Bounding stretch We distinguish between two types of iterations i that the algorithm did not stop at. Let $I_u = \{0 \le i \le k-1 : u \notin \tilde{C}(\hat{z}_i(v))\}$, and let $I_v = \{0 \le i \le k-1 : \{u, v\} \cap \tilde{C}(\hat{z}_i(v)) = \{u\}\}$ be the remaining iterations in which the algorithm did not halt. For any $i \in I_u$, by (9) it holds that $C_{6\epsilon}(\hat{z}_i(v)) \subseteq \tilde{C}(\hat{z}_i(v))$. Hence, we have $u \notin C_{6\epsilon}(\hat{z}_i(v))$, which implies that

$$\begin{aligned} d_G(u, \hat{z}_{i+1}(u)) &\overset{(7)}{\le} (1+\epsilon) d_G(u, A_{i+1}) \\ &\overset{(8)}{\le} (1+\epsilon)(1+6\epsilon) d_G(u, \hat{z}_i(v)) \\ &\le (1+8\epsilon) d_G(u, \hat{z}_i(v)). \end{aligned} \tag{33}$$

Similarly for $i \in I_v$,

$$\begin{aligned} d_G(v, \hat{z}_{i+1}(v)) &\le (1+\epsilon) d_G(v, A_{i+1}) \\ &\le (1+\epsilon)(1+6\epsilon) d_G(v, \hat{z}_i(v)) \\ &\le (1+8\epsilon) d_G(v, \hat{z}_i(v)). \end{aligned} \tag{34}$$

Define the following values $y_0 = d_G(u, v)$, $x_0 = 0$, and for $0 < i \le k-1$ define recursively $y_i = (1+10\epsilon) \cdot (y_0 + x_{i-1})$, and $x_i = (1+\epsilon) \cdot (y_0 + y_i)$. Assume that the algorithm halted at iteration i'. Then for each $0 \le i \le i'$ we claim that

$$d_G(v, \hat{z}_i(v)) \le x_i. \tag{35}$$

We verify the validity of (35) by induction. The base case trivially holds since $\hat{z}_0(v) = v$ and $x_0 = 0$. Fix $0 < i \le i'$. The algorithm did not halt at iteration $i - 1$. If it is the case that $i - 1 \in I_u$, then we have that

$$\begin{aligned} d_G(u, \hat{z}_i(u)) &\overset{(33)}{\le} (1+8\epsilon) d_G(u, \hat{z}_{i-1}(v)) \\ &\le (1+8\epsilon) \cdot (d_G(u, v) + d_G(v, \hat{z}_{i-1}(v))) \\ &\overset{(35)}{\le} (1+8\epsilon) \cdot (y_0 + x_{i-1}) \\ &\le y_i. \end{aligned} \tag{36}$$

The other case is that $i - 1 \in I_v$. Since $\hat{z}_i(u) \in A_i$ we obtain

$$\begin{aligned} d_G(u, \hat{z}_i(u)) &\overset{(7)}{\le} (1+\epsilon) d_G(u, A_i) \\ &\le (1+\epsilon) d_G(u, \hat{z}_i(v)) \\ &\le (1+\epsilon) \cdot (d_G(u, v) + d_G(v, \hat{z}_i(v))) \\ &\overset{(34)}{\le} (1+\epsilon) \cdot (d_G(u, v) \\ &\quad + (1+8\epsilon) d_G(v, \hat{z}_{i-1}(v))) \\ &\le (1+10\epsilon) \cdot (y_0 + x_{i-1}) \\ &= y_i \end{aligned} \tag{37}$$

We conclude that in both cases,

$$\begin{aligned} d_G(v, \hat{z}_i(v)) &\le (1+\epsilon) d_G(v, A_i) \\ &\le (1+\epsilon) d_G(v, \hat{z}_i(u)) \\ &\le (1+\epsilon) \cdot (d_G(u, v) + d_G(u, \hat{z}_i(u))) \\ &\overset{(36) \wedge (37)}{\le} (1+\epsilon) \cdot (y_0 + y_i) \\ &= x_i. \end{aligned} \tag{38}$$

We now have a recurrence $x_i = (1+\epsilon)(2+10\epsilon)y_0 + (1+\epsilon)(1+10\epsilon)x_{i-1}$. Solving it yields

$$x_i = (1+\epsilon)(2+10\epsilon)y_0 \sum_{j=0}^{i-1}((1+\epsilon)(1+10\epsilon))^j.$$

We use the fact that for any real $x \geq 0$ and positive integer r such that $xr \leq 1/2$, the following holds $(1+x)^r \leq 1+2xr$. Now we may bound x_i by

$$\begin{aligned} x_i &\leq (2+13\epsilon)y_0 \sum_{j=0}^{i-1}(1+12\epsilon)^j \\ &\leq (2+13\epsilon)y_0 \sum_{j=0}^{i-1}(1+24\epsilon j) \\ &\leq (2+13\epsilon)y_0(i+12\epsilon i^2) \\ &\leq (2+13\epsilon)y_0(i+1/(4k^2)), \end{aligned} \tag{39}$$

where in the last inequality we use that $\epsilon = \frac{1}{48k^4} \leq \frac{1}{48k^2i^2}$. Finally, using that $i' \leq k-1$ and that $w = \hat{z}_{i'}(v)$, the stretch is given by

$$\begin{aligned} & d_{\tilde{C}(w)}(u,w) + d_{\tilde{C}(w)}(w,v) \\ & \quad \overset{(10)}{\leq} (1+\epsilon)^4(d_G(u,w)+d_G(v,w)) \\ & \quad \overset{(35)}{\leq} (1+5\epsilon)\cdot(d_G(u,v)+2x_{i'}) \\ & \quad \overset{(39)}{\leq} (1+5\epsilon)\cdot(1+(4+26\epsilon)(k-1+1/(4k^2))) \\ & \qquad \cdot d_G(u,v) \\ & \quad \leq (4k-3+o(1))\cdot d_G(u,v). \end{aligned}$$

In order to improve the stretch to the promised $4k-5+o(1)$, we use same trick as in [33]. They used recursive sampling to guarantee that each vertex $u \in A_0 \backslash A_1$ has $|C(u)| \leq 4n^{1/k}$. (In the recursive sampling, centers of clusters which are larger than $4n^{1/k}$ may be sampled again to A_1. As shown in [33], this may increase by $O(\log n)$ factor the size of A_1. We note that this resampling can be easily done in the distributed setting, since for the first level we compute the exact clusters and know their respective sizes.) Now, each vertex $u \in A_0 \backslash A_1$ will add to its routing table all the vertices in $C(u)$. The algorithm will change slightly in the first iteration: we try to find u in $C(v)$ as before, but also check if v is in $C(u)$. If v is found in $C(u)$ then we are done (with stretch 1). Otherwise, it follows that $d_G(v, A_1) \leq d_G(u,v)$, that is, we can bound $x_1 \leq (1+\epsilon)\cdot y_0$. This saves an additive term of $d_G(u,v)$ in both x_i and y_i for all $i \geq 1$, and thus reduces the final stretch by $2d_G(u,v)$. We refer the reader to [33] for more details.

Running time By Theorem 4, the time required to compute the approximate pivots and the trees $\tilde{C}(u)$ for every $u \in A_i \backslash A_{i+1}$ is $(n^{1/2+1/k}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$, when k is even, and $(n^{1/2+1/(2k)}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$, when k is odd. By Claim 2, each vertex participates in at most $\tilde{O}(n^{1/k})$ trees. Hence, by Lemma 10, which will be stated and proven in Sect. 4, it will take only $\tilde{O}(n^{1/2+1/(2k)}+D)$ rounds to compute the routing tables for all trees in parallel. We conclude that the total number of rounds is $(n^{1/2+1/k}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$, for even k, and $(n^{1/2+1/(2k)}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$, for odd.

6 Distance estimation

In this section we sketch how the routing tables can be used for distance estimation, and prove the following.

Theorem 7 *Let $G=(V,E)$ be a weighted graph with n vertices and hop-diameter D, and let $k \geq 1$ be a parameter. Then there exists a distance estimation scheme, that assigns a sketch of size $O(n^{1/k}\log n)$ for each node, and has stretch $2k-1+o(1)$, that can be computed by a randomized distributed algorithm within $(n^{1/2+1/k}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds (whp). In the case of odd k, the running time can be decreased to $(n^{1/2+1/(2k)}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$. Furthermore, the distance computation can be done in time $O(k)$.*

Apply Theorem 4, which computes all the approximate pivots and approximate clusters. Each vertex $v \in V$ includes in its sketch the pair $(u, b_v(u))$, for every $u \in V$ so that $v \in \tilde{C}(u)$. (Here $b_v(u)$ is the approximate distance to u computed in Sect. 3). Also for every $0 \leq i \leq k-1$, add $(\hat{z}_i(v), \hat{d}_i(v))$, which is the approximate i-pivot and distance to it. By Remark 2, every sketch is of size $O(n^{1/k}\log n)$. The algorithm that computes a distance estimate given two sketches is similar to that of [34]. We state it formally in Algorithm 2.

Algorithm 2 `Dist`(u,v)

```
i ← 0;
w ← u;
while $v \notin \tilde{C}(w)$ do
    i ← i + 1;
    $(u, v) \leftarrow (v, u)$;
    $w \leftarrow \hat{z}_i(u)$;
end while
return $\hat{d}_i(u) + b_v(w)$;
```

Observe that the sketch contains all the relevant information for executing Algorithm 2. When the while loop terminates, it must be that $v \in \tilde{C}(w)$. This implies that v has the estimate $b_v(w)$, while u stores the approximate distance $\hat{d}_i(u)$ to every one of its approximate pivots. The stretch analysis is a variant of the analysis of [34], similar in spirit to that of Sect. 5. Roughly speaking, on the stretch $2k-1$ achieved by Thorup and Zwick [34], we pay a multiplicative factor of $(1+O(\epsilon))^k$ due to the fact that distances are approximated. However, this boils down to an $o(1)$ additive term, since $\epsilon=\frac{1}{48k^4}$, and so $(1+O(\epsilon))^k=1+O(1/k^3)$.

Appendix: Proof of Theorem 3

Let $X\subseteq V$ be a set of vertices so that each $v\in V$ is sampled to X independently with probability $1/\sqrt{n}$. Define $V'=A\cup X$, and note that with high probability $B=4\sqrt{n}\ln n\geq|V'|$ (since it is given that $|A|\leq 2\sqrt{n}\ln n$). Apply the same preprocessing steps as in Sect. 3.3.1 with V' as defined here, to obtain a graph G'' on V' satisfying (13).

Computing approximate SPT for V' The first step is to compute the values $(\hat{d}(v),\hat{z}(v))$ for vertices $v\in V'$. Every vertex in $v\in A$ initializes its values as $(0,v)$, while $v\notin A$ sets $(\infty,\bot)$. Conduct $\beta=\min\{2^{\tilde{O}(\sqrt{\log n})},(\log n)^{O(k)}\}$ iterations of Bellman–Ford rooted at A: at every iteration, every vertex $v\in V'$ broadcasts its pair $(\hat{d}(v),\hat{z}(v))$ to the entire graph, and if $u\in V'$ has $w''(u,v)+\hat{d}(v)<\hat{d}(u)$, then u updates its pair to be $(w''(u,v)+\hat{d}(v),\hat{z}(v))$. (Recall that w'' is the edge weight function of G'', where the latter is the virtual graph given by Theorem 1 augmented with the hopset edges of Theorem 2.)

The number of rounds required to construct G'' is $(n^{1/2+1/(2k)}+D)\cdot\min\{2^{\tilde{O}(\sqrt{\log n})},(\log n)^{O(k)}\}$, and by Lemma 1 this term also bounds the number of rounds it takes to broadcast the $O(|V'|\cdot\beta)$ messages for the Bellman–Ford iterations.

Extending the SPT to V At the end of the β iterations of Bellman–Ford, every vertex $u\in V$ knows $(\hat{d}(v),\hat{z}(v))$ for every $v\in V'$. Every vertex $u\in V$ computes

$$\hat{d}(u)=\min_{v\in V'}\{d_{uv}+\hat{d}(v)\},\tag{40}$$

and sets $\hat{z}(u)=\hat{z}(v)$, where $v\in V'$ is the minimizer of (40). (Recall that d_{uv} is the value computed in Theorem 1.)

Analysis We assume all the events of Claim 3 hold (which happens with high probability). For $u\in V$ let $z_u\in A$ be a vertex satisfying $d_G(u,z_u)=d_G(u,A)$. Since we performed β iterations of Bellman–Ford, using (13) with $v\in V'$ and $z_v\in A\subseteq V'$ we have that v' satisfies (5).

Consider now some $u\in V$, and let $v\in V'$ be the minimizer in (40). The left hand side of (5) holds, as the fact that $v\in V'$ satisfies (5) implies

$$\begin{aligned}d_{uv}+\hat{d}(v)&\overset{(2)}{\geq} d_G^{(B)}(u,v)+d_G(v,A)\\&\geq d_G(u,v)+d_G(v,A)\geq d(u,A).\end{aligned}$$

For the right hand side of (5): in the case that $h(u,z_u)\leq B$, by (2) we get that

$$\begin{aligned}\hat{d}(u)&\leq d_{uz_u}+\hat{d}(z_u)\leq(1+\epsilon)d_G^{(B)}(u,z_u)+0\\&=(1+\epsilon)d_G(u,z_u).\end{aligned}$$

Otherwise $h(u,z_u)>B$, and by Claim 3 there exists $v\in X\subseteq V'$ on the shortest path in G from u to z_u with $h(u,v)\leq B$. Since (5) holds for v,

$$\begin{aligned}\hat{d}(u)&\leq d_{uv}+\hat{d}(v)\\&\overset{(2)}{\leq}(1+\epsilon)d_G^{(B)}(u,v)+(1+\epsilon)d_G(v,A)\\&\leq(1+\epsilon)d_G(u,v)+(1+\epsilon)d_G(v,z_u)\\&=(1+\epsilon)d_G(u,z_u).\end{aligned}$$

References

1. Awerbuch, B., Bar-Noy, A., Linial, N., Peleg, D.: Improved routing strategies with succinct tables. J. Algorithms **11**(3), 307–341 (1990)
2. Abraham, I., Gavoille, C., Malkhi, D.: Routing with improved communication-space trade-off. In: Distributed Computing, 18th International Conference, DISC 2004, Amsterdam, The Netherlands, October 4–7, 2004, Proceedings, pp. 305–319 (2004)
3. Bernstein, A.: Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In: 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25–27, 2009, Atlanta, Georgia, USA, pp. 693–702 (2009)
4. Chechik, S.: Compact routing schemes with improved stretch. In: ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22–24, 2013, pp. 33–41 (2013)
5. Cohen, E.: Polylog-time and near-linear work approximation scheme for undirected shortest paths. J. ACM **47**(1), 132–166 (2000)
6. Cowen, L.: Compact routing with minimum stretch. J. Algorithms **38**(1), 170–183 (2001)
7. Eilam, T., Gavoille, C., Peleg, D.: Compact routing schemes with low stretch factor. J. Algorithms **46**(2), 97–114 (2003)
8. Elkin, M.: A faster distributed protocol for constructing a minimum spanning tree. J. Comput. Syst. Sci. **72**(8), 1282–1308 (2006)
9. Elkin, M.: An unconditional lower bound on the time-approximation trade-off for the distributed minimum spanning tree problem. SIAM J. Comput. **36**(2), 433–456 (2006)
10. Elkin, M., Neiman, O.: Hopsets with constant hopbound, and applications to approximate shortest paths. In: 57th IEEE Annual Symposium on Foundations of Computer Science, FOCS (2016)
11. Elkin, M., Neiman, O.: On efficient distributed construction of near optimal routing schemes: Extended abstract. In: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25–28, 2016, pp. 235–244 (2016)
12. Ghaffari, M.: Near-optimal scheduling of distributed algorithms. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC '15, ACM, New York, NY, USA, pp. 3–12 (2015)
13. Ghaffari, M., Kuhn, F.: Distributed minimum cut approximation. In: Distributed Computing—27th International Symposium, DISC

2013, Jerusalem, Israel, October 14–18, 2013. Proceedings, pp. 1–15 (2013)

14. Garay, J.A., Kutten, S., Peleg, D.: A sublinear time distributed algorithm for minimum-weight spanning trees. SIAM J. Comput. **27**(1), 302–316 (1998)
15. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distrib. Comput. **16**(2–3), 111–120 (2003)
16. Henzinger, M., Krinninger, S., Nanongkai, D.: Decremental single-source shortest paths on undirected graphs in near-linear total update time. In: 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18–21, 2014, pp. 146–155 (2014)
17. Henzinger, M., Krinninger, S., Nanongkai, D.: A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In: Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18–21, 2016, pp. 489–498 (2016)
18. Izumi, T., Wattenhofer, R.: Time lower bounds for distributed distance oracles. In: Principles of Distributed Systems—18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16–19, 2014. Proceedings, pp. 60–75 (2014)
19. Kutten, S., Peleg, D.: Fast distributed construction of small k-dominating sets and applications. J. Algorithms **28**(1), 40–66 (1998)
20. Lenzen, C., Patt-Shamir, B.: Fast routing table construction using small messages. In: Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1–4, 2013, pp. 381–390 (2013)
21. Lenzen, C., Peleg, D.: Efficient distributed source detection with limited bandwidth. In: ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22–24, 2013, pp. 375–382 (2013)
22. Lenzen, C., Patt-Shamir, B.: Fast partial distance estimation and applications. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015, pp. 153–162 (2015)
23. Lenzen, C., Patt-Shamir, B.: Private communication (2016)
24. Lenzen, C., Patt-Shamir, B., Peleg, D.: Distributed distance computation and routing with small messages (2016) (Unpublished)
25. Nanongkai, D.: Distributed approximation algorithms for weighted shortest paths. In: Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31–June 03, 2014, pp. 565–573 (2014)
26. Nisse, N., Rapaport, I., Suchan, K.: Distributed computing of efficient routing schemes in generalized chordal graphs. Theor. Comput. Sci. **444**, 17–27 (2012)
27. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia, PA (2000)
28. Peleg, D.: Proximity-preserving labeling schemes. J. Graph Theory **33**(3), 167–176 (2000)
29. Peleg, D., Rubinovich, V.: A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. SIAM J. Comput. **30**(5), 1427–1442 (2000)
30. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. ACM **36**(3), 510–530 (1989)
31. Sarma, A.D., Dinitz, M., Pandurangan, G.: Efficient distributed computation of distance sketches in networks. Distrib. Comput. **28**(5), 309–320 (2015)
32. Sarma, A.D., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., Wattenhofer, R.: Distributed verification and hardness of distributed approximation. SIAM J. Comput. **41**(5), 1235–1265 (2012)
33. Thorup, M., Zwick, U.: Compact routing schemes. In: Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '01, pp. 1–10, New York, NY, USA, ACM (2001)
34. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM **52**(1), 1–24 (2005)