On Efficient Distributed Construction of Near Optimal Routing Schemes

[Extended Abstract] *

Michael Elkin†
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel.
elkinm@cs.bgu.ac.il

Ofer Neiman‡
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel.
neimano@cs.bgu.ac.il

ABSTRACT

Given a distributed network represented by a weighted undirected graph $G = (V, E)$ on n vertices, and a parameter k, we devise a distributed algorithm that 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. The 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 [LP13a, STOC 2013] and [LP15, PODC 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 $\tilde{O}(\min\{(nD)^{1/2} \cdot n^{1/k}, n^{2/3+2/(3k)} + D\})$.

Keywords

Routing; CONGEST model

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 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 [ABLP90, Cow01, EGP03, GP03, AGM04, PU89, TZ01, Che13]. In [TZ01], 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 of $\log n$.)[2] The state-of-the-art is a scheme of [Che13], which is based on [TZ01], and improves the stretch to $3.68k$.

All the results above assume that the preprocessing phase can be computed 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 [Pel00a]. 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

*A full version of this paper is available at `http://arxiv.org/abs/1602.02293`

†This research was supported by the ISF grant 724/15.

‡Supported in part by ISF grant No. (523/12) and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement $n^{\circ}303809$.

PODC'16, July 25-28, 2016, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/2933057.2933098

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

[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.

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 [LP13a, Nan14, KP98, GK13, HKN16], that edge weights are integers and at most polynomial in n (so that they could be sent in a single message).[3]

A rich research thread concerns with finding efficient distributed (approximation) algorithms for classical graph problems (e.g., minimum spanning tree, minimum cut, shortest paths), in sub-linear time [GKP98, PR00, Elk06a, SHK+12, HKN16]. 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 a (nearly) matching lower bounds. The lower bound of [SHK+12], based on [PR00, Elk06b], 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 [LP13a].[4] 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 [TZ01] 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, [LP15] 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 $4k-3+o(1)$,[5] 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.)

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 [Pel00b], who provided initial existential results. In [SDP15], 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 [TZ05], the number of rounds could be $\Omega(n)$, even when D is small. In [LP13a], a running time of $\tilde{O}(n^{1/2+1/k}+D)$ rounds was presented, at the cost of significantly increasing the stretch to $O(k^2)$.[6]

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[7], that with high probability, 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 $4k-5+o(1)$. Note that our result nearly matches the construction of [TZ01], 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 [SHK+12], and is substantially better than that of [LP15] whenever $D \geq n^{\Omega(1)}$ (which achieved similar size-stretch tradeoff). The previous result obtaining near optimal running time [LP13a], suffers from excessive routing table size.

As a corollary, we show a distance estimation scheme, whose computation can be done in $(n^{1/2+1/k}+D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ 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 [LP13a] (up to lower order terms), with the near optimal size-stretch tradeoff of [SDP15].

When preparing this submission, we learnt that concurrently and independently of us [LPP16] 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 $O(k \log n)$, and stretch at most $4k-3+o(1)$. Full details of [LPP16] algorithm and analysis are not currently available to us.

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 constant routing tables and logarithmic label size. 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 $u \in A_i \setminus A_{i+1}$ is defined as

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

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

[3] We shall not consider *name-independent* routing, in which the label of a vertex is its ID, because [LP13a] showed a strong lower bound: any such scheme with stretch ρ (even average stretch ρ) must take $\tilde{\Omega}(n/\rho^2)$ rounds to compute in this model.

[4] We remark that for the class of k-chordal graphs, [NRS12] showed a construction of a routing scheme that could be computed efficiently in a distributed manner.

[5] The paper [LP15] claimed label size $O(k \log n)$, but in [LP16] it was communicated to us that the actual size is $O(k \log^2 n)$.

[6] 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 flavor.

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

	Number of Rounds	Table size	Label size	Stretch
[TZ01, Che13]	$O(m)$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log n)$	$3.68k$
[LP15]	$\tilde{O}(S + n^{\frac{1}{k}})$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log n)$	$4k - 3$
[LP13a, LP15]	$\tilde{O}(n^{\frac{1}{2}+\frac{1}{4k}} + D)$	$\tilde{O}(n^{\frac{1}{2}+\frac{1}{4k}})$	$O(\log n)$	$6k - 1 + o(1)$
[LP15]	$\tilde{O}(\min\{(nD)^{\frac{1}{2}} \cdot n^{\frac{1}{k}}, n^{\frac{2}{3}+\frac{2}{3k}} + D\})$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log^2 n)$	$4k - 3 + o(1)$
This paper	$(n^{1/2+1/k} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$	$\tilde{O}(n^{\frac{1}{k}})$	$O(k \log^2 n)$	$4k - 5 + o(1)$

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

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 tree could take a linear number of rounds. We thus develop a variation on that scheme, that can be implemented efficiently in a distributed network. The basic idea is inspired by [KP98] (and also used in [Nan14]), 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[8] induced on the sampled vertices, which must incorporate the local routing information.

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. Even for a single shortest path tree (SPT), no algorithm running in $o(n)$ rounds is known. 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 are sufficient for constructing a routing scheme, with nearly matching size and stretch as in [TZ01].

The construction of clusters $C(u)$ for $u \in A_i \setminus A_{i+1}$ 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 [Nan14] (a certain variation appeared also in [LP13b]). 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 [Coh00] in the context of parallel approximate shortest path algorithms, and it has found applications in dynamic and distributed settings as well [Ber09, HKN14, HKN16]. 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 $\epsilon \approx 1/k^4$. An efficient distributed algorithm to construct such hopsets is given by [HKN16, EN16]. We shall use the construction of [EN16], since it facilitates much smaller β, whenever k is small. 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 [Nan14]). 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 [TZ05], 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 [LP13a, LP15]. In [LP15], 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 [LP13a], the idea was to build a spanner on a sample of $\approx \sqrt{n}$ vertices, which helps by reducing the number of edges, so a routing scheme could 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 [LP15] 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 ver-

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

tices 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, enabling fast running time.

1.2 Organization

After stating in Section 2 some of the tools we shall apply, in Section 3 we describe the notion of approximate clusters, and show how to compute these efficiently in a distributed manner. The proof of correctness is deferred to the full version. Then in Section 4, we demonstrate how these approximate clusters could be used for a routing scheme in general graphs. In Section 5 we show the distance estimation scheme. Our distributed tree routing is deferred to the full version.

2. PRELIMINARIES

Let $G = (V, E, w)$ be a weighted graph on n vertices. We assume that $w : E \to \{1, \ldots, \text{poly}(n)\}$ (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. 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 $d_{G'}(u, v) \geq d_G(u, v)$. Every vertex in V' should know all the edges of E' touching it. Throughout the paper, we measure the size of the labels and routing tables in machine *words*, where a word consists of $\Theta(\log n)$ bits. The following lemma formalizes the broadcast ability of a distributed network (see, e.g., [Pel00a]).

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.*

2.1 Tools

We first state our theorem on distributed tree routing, which is proven in the full version of this paper.

THEOREM 1. *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.*

We also note we can compute efficiently many trees in parallel, if no vertex participates in too many of them.

REMARK 1. *If we are given n trees, each a subgraph 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.*

Computing hop-bounded distances from multiple sources.

We will make use of the following theorem due to [Nan14, Theorem 3.6], which shows how to compute hop-bounded distances from a given set of sources, efficiently in a distributed manner.

THEOREM 2 ([NAN14]). *Given a weighted graph $G = (V, E, w)$ of hop-diameter D, a set $V' \subseteq V$, and parameters $B \geq 1$ and $0 < \epsilon < 1$, there is a (randomized) distributed algorithm that w.h.p runs in $\tilde{O}(|V'| + B + D)/\epsilon$ rounds, so that every $u \in V$ will know values $\{d_{uv}\}_{v \in V'}$ satisfying*[9]

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

REMARK 2. *While not explicitly stated in [Nan14], 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*

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

The following notion of hopsets was introduced by [Coh00].

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

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

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 our companion paper [EN16], which provides a path-reporting hopset. We remark that the original hopset construction of [Coh00] could be made path-reporting. Also, in [HKN16, Theorem 4.10], a distributed algorithm constructing a hopset is provided, which possibly could be made to be path-reporting, however, it inherently cannot provide a better hopbound than $2^{\tilde{O}(\sqrt{\log n})}$.

THEOREM 3 ([EN16]). *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 w.h.p 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 the size of the hopset is important. However, here we only care about the size to the extent that it affects the number of rounds required to compute the hopset.

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

Approximate Shortest Path Tree (SPT).

Very recently, [HKN16] obtained an efficient distributed algorithm for computing an approximate SPT, which we shall use. Let us first define the problem 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

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

and that u will know a vertex $\hat{z}(u) \in A$ so that $d_G(u, \hat{z}(u)) \leq \hat{d}(u)$. The following theorem is shown in [HKN16].[10]

THEOREM 4 ([HKN16]). *Let $G = (V, E, w)$ be a weighted graph on n vertices with hop-diameter D. Given a set $A \subseteq V$ of size $|A| \leq 2\sqrt{n} \ln n$, and $\frac{1}{\text{polylog } n} < \epsilon < 1$, there is a distributed algorithm that computes an $(1+\epsilon)$-approximate SPT rooted at A in $\tilde{O}(\sqrt{n} + D) \cdot 2^{\tilde{O}(\sqrt{\log n})}$ rounds.*

We remark that using the hopsets of [EN16], one can obtain an approximate SPT as in Theorem 4 within $(n^{1/2+1/k} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds. (This holds for any $1 \leq k \leq \log n$. Choosing $k = \sqrt{\log n}$ captures the bound in the theorem.)

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 [TZ05].

Let $G = (V, E, w)$ be a weighted graph, fix $k \geq 1$. 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 $u \in A_i \setminus A_{i+1}$ is defined as

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

We quote a claim from [TZ05], 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 $0 \leq i \leq k-1$: (1) $|A_i| \leq 4n^{1-i/k} \ln n$, 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 u to v shortest path is included in A_i is at most

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

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

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 $0 \leq i \leq k-1$, a point $\hat{z} \in A_i$ is called an *approximate i-pivot* of v if

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

Now we define for each $0 \leq i \leq k-1$ and each vertex $u \in A_i \setminus A_{i+1}$, 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

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

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

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

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

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

REMARK 3. *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 5. *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 w.h.p 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.*[11]

[10] In fact, [HKN16] shows this theorem for a single root and for $\epsilon = 1/\log n$, but an inspection of their proof gives the claimed result.

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

Computing Pivots.

We first compute the pivots for $0 \le i \le \lceil k/2 \rceil$. 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 $h(v,u) \le 4n^{i/k} \cdot \ln n$, 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 $\sum_{i=0}^{\lceil k/2 \rceil} O(n^{i/k} \cdot \ln n) \le \tilde{O}(n^{1/2+1/(2k)})$.

For $\lceil k/2 \rceil < i \le k-1$ we can only compute *approximate* pivots $\hat{z}_i(v)$ for each $v \in V$. For each such i, apply Theorem 4 with root set A_i and the parameter ϵ (indeed by Claim 3, $|A_i| \le 4n^{1-(\lceil k/2 \rceil + 1)/k} \ln n \le 2\sqrt{n} \ln n$, and $\epsilon = \Omega(1/k^4) \ge \Omega(1/\log^4 n)$). This will take $(n^{1/2+1/(2k)} + D) \cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$ rounds (see the remark after Theorem 4). 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 $0 \le i < \lceil k/2 \rceil$, we can compute the trees $C(u)$ corresponding to the actual clusters. We need to find such a tree for every $u \in A_i \setminus A_{i+1}$, 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. By bounded-depth we mean the following: 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 $i \le \lceil k/2 \rceil$, 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

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

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 $h(u,v) \le 4n^{(i+1)/k} \ln n$. So there are enough Bellman-Ford iterations to reach all vertices of $C(u)$.

The middle level.

When k is odd, the level $i = (k-1)/2$ will induce large running time $\tilde{O}(n^{1/2+3/(2k)})$ (see the upcoming paragraph on running-time analysis). To overcome this, we use a different method for this level. We apply Theorem 2 on the set of sources $S = A_{(k-1)/2} \setminus A_{(k+1)/2}$, with $B = 4n^{(i+1)/k} \cdot \ln n$ and ϵ, 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 $h(u,v) \le B$, 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

$$b_v(u) < d_G(v, A_{(k+1)/2})$$

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

$$\begin{aligned} b_p(u) &\overset{(3)}{\le} b_v(u) - w(v,p) < d_G(v, A_{(k+1)/2}) - d_G(v,p) \\ &\le d_G(p, A_{(k+1)/2}) \ . \end{aligned}$$

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

$$b_v(u) \overset{(2)}{\le} (1+\epsilon) d_G(u,v) \overset{(8)}{<} d_G(v, A_{(k+1)/2}) \ ,$$

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 2. In all 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 $u \in A_i \setminus A_{i+1}$ when $i \ge \lceil k/2 \rceil$ is more involved, since the number of iterations for the simple Bellman-Ford style approach grows like $\approx n^{(i+1)/k}$. We will use the fact that there are only few roots, 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 all the 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 $B = 4\mathbb{E}[n/|V'|] \cdot \ln n$. 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 2 to G with the set V' and parameters B and $\epsilon/2$. By Claim 3 we may assume $|V'| \le 4n^{1/2} \ln n$, and since $1/\epsilon \le 48 \log^4 n$, the number of rounds required is w.h.p $\tilde{O}(n^{1/2+1/(2k)} + D)$. From now on assume that (2) indeed holds (which happens w.h.p). 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 2). Following [Nan14], it can be shown that for any $u, v \in V'$,

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

Apply Theorem 3 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 is

$$\begin{aligned} &\tilde{O}(|V'|^{1+\rho} + D) \cdot \beta^2 \\ &= \tilde{O}(n^{1/2+1/(2k)} + D) \cdot \min\{2^{\tilde{O}(\sqrt{\log n})}, (\log n)^{O(k)}\} \ . \end{aligned}$$

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 F.) By (4) and (12) we have that G'' is indeed a virtual graph since $d_{G''}(u,v) \ge d_{G'}(u,v) \ge d_G(u,v)$. On the other hand,

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

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

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

3.3.2 Construction

Fix $\lceil k/2 \rceil \le i \le k-1$. We build the trees $\tilde{C}(u)$ for all $u \in A_i \setminus A_{i+1}$ in parallel, in two main phases.

Phase 1.

For each such u, conduct β iterations of depth-bounded Bellman-Ford in the graph G''.[12] (Since this is a virtual graph, all the messages will be collected at the root of some BFS tree via pipelined convergecast, and then broadcast to the entire graph G via pipelined broadcast. See Lemma 1.) If $v \in V'$ receives a message 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

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

(Recall that $\hat{d}_{i+1}(v)$ is the approximate distance from v to the its (approximate) level i 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 $u \in A_i \setminus A_{i+1}$, 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 additional step is to guarantee that every vertex which was added to the (virtual) tree being built for some $u \in A_i \setminus A_{i+1}$, 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 2 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 $v \in P \setminus \{x\}$ that has $b_v(u)$ value at least $b_x(u) + d_P(x,v)$, updates its distance estimate $b_v(u) = b_x(u) + d_P(x,v)$, 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 $p(v) = p_{v'}(v)$ (recall Remark 2).

Phase 2.

Here we extend each virtual tree $\tilde{C}'(u)$ to the vertices of V. For all $u \in A_i \setminus A_{i+1}$, 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

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

where d_{yv} is the value computed in Theorem 2. Also y will set $p(y) = p_v(y)$ as its 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). Thus vertices of V' who did not join $\tilde{C}'(u)$, may now be included in $\tilde{C}(u)$.

We defer the proof that every $\tilde{C}(u)$ is indeed an approximate cluster, that it corresponds to a tree in G, and the running time analysis, to the full version.

4. 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 $k \ge 1$ 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.*

Construction.

Apply Theorem 5 on G to obtain approximate pivots and approximate clusters for all vertices. For each $0 \le i \le k-1$ and each $u \in A_i \setminus A_{i+1}$, construct the routing scheme for trees given by Theorem 1 on $\tilde{C}(u)$. Recall that 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 3 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

[12] See (14) below for the required condition.

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 $0 \leq i \leq k-1$, 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 ← 0;
while |{u, v} ∩ C̃(ẑ_i(v))| < 2 do
    i ← i + 1;
end while
return ẑ_i(v);
```

We note that our algorithm differs slightly from that of [TZ05], 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 (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 algorithm did not stop at. Let $I_u = \{0 \leq i \leq k-1 \ : \ u \notin \tilde{C}(\hat{z}_i(v))\}$ be the iterations in which $\{u, v\} \cap \tilde{C}(\hat{z}_i(v))$ is empty or contains just v, and let $I_v = \{0 \leq i \leq k-1 \ : \ \{u, v\} \cap \tilde{C}(\hat{z}_i(v)) = \{u\}\}$ be the remaining iterations that 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 suggests that

$$\begin{aligned} d_G(u, \hat{z}_{i+1}(u)) &\overset{(7)}{\leq} (1+\epsilon) d_G(u, A_{i+1}) \\ &\overset{(8)}{\leq} (1+\epsilon)(1+6\epsilon) d_G(u, \hat{z}_i(v)) \\ &\leq (1+8\epsilon) d_G(u, \hat{z}_i(v)) \ . \end{aligned} \tag{16}$$

Similarly for $i \in I_v$,

$$\begin{aligned} d_G(v, \hat{z}_{i+1}(v)) &\leq (1+\epsilon) d_G(v, A_{i+1}) \\ &\leq (1+\epsilon)(1+6\epsilon) d_G(v, \hat{z}_i(v)) \\ &\leq (1+8\epsilon) d_G(v, \hat{z}_i(v)) \ . \end{aligned} \tag{17}$$

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

$$d_G(v, \hat{z}_i(v)) \leq x_i \ . \tag{18}$$

We verify the validity of (18) by induction, the base case trivially holds since $\hat{z}_0(v) = v$ and $x_0 = 0$. Fix $0 < i \leq i'$. The algorithm did not halt at iteration $i-1$. If it is the case that $i-1 \in I_u$, then we have that

$$\begin{aligned} d_G(u, \hat{z}_i(u)) &\overset{(16)}{\leq} (1+8\epsilon) d_G(u, \hat{z}_{i-1}(v)) \\ &\leq (1+8\epsilon)[d_G(u, v) + d_G(v, \hat{z}_{i-1}(v))] \\ &\overset{(18)}{\leq} (1+8\epsilon)[y_0 + x_{i-1}] \\ &\leq y_i \ . \end{aligned} \tag{19}$$

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

$$\begin{aligned} d_G(u, \hat{z}_i(u)) &\overset{(7)}{\leq} (1+\epsilon) d_G(u, A_i) \\ &\leq (1+\epsilon) d_G(u, \hat{z}_i(v)) \\ &\leq (1+\epsilon)[d_G(u, v) + d_G(v, \hat{z}_i(v))] \\ &\overset{(17)}{\leq} (1+\epsilon)[d_G(u, v) + (1+8\epsilon) d_G(v, \hat{z}_{i-1}(v))] \\ &\leq (1+10\epsilon)[y_0 + x_{i-1}] \\ &= y_i \end{aligned} \tag{20}$$

We conclude that in both cases,

$$\begin{aligned} d_G(v, \hat{z}_i(v)) &\leq (1+\epsilon) d_G(v, A_i) \\ &\leq (1+\epsilon) d_G(v, \hat{z}_i(u)) \\ &\leq (1+\epsilon)[d_G(u, v) + d_G(u, \hat{z}_i(u))] \\ &\overset{(19)\wedge(20)}{\leq} (1+\epsilon)[y_0 + y_i] \\ &= x_i \ . \end{aligned} \tag{21}$$

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

$$\begin{aligned} x_i &\leq (2+13\epsilon) y_0 \sum_{j=0}^{i-1} (1+12\epsilon)^j \\ &\leq (2+13\epsilon) y_0 \sum_{j=0}^{i-1} (1+24\epsilon j) \\ &\leq (2+13\epsilon) y_0 (i + 12\epsilon i^2) \\ &\leq (2+13\epsilon) y_0 (i + 1/(4k^2)) \ , \end{aligned} \tag{22}$$

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

is given by

$$\begin{aligned} & d_{\tilde{C}(w)}(u,w) + d_{\tilde{C}(w)}(w,v) \\ & \overset{(10)}{\leq} (1+\epsilon)^4[d_G(u,w) + d_G(v,w)] \\ & \overset{(18)}{\leq} (1+5\epsilon)[d_G(u,v) + 2x_{i'}] \\ & \overset{(22)}{\leq} (1+5\epsilon)[1 + (4+26\epsilon)(k-1+1/(4k^2))] \cdot d_G(u,v) \\ & \leq (4k-3+o(1)) \cdot d_G(u,v) \ . \end{aligned}$$

In order to improve the stretch to the promised $4k-5+o(1)$, we use same trick as in [TZ01]. Each vertex $u \in A_0 \setminus A_1$ will store in its routing table all the labels for vertices in $C(u)$, which enables to save an additive term of $d_G(u,v)$ in both x_i and y_i. We refer the reader to [TZ01] for the details.

Running time.

By Theorem 5, the time required to compute the approximate pivots and the trees $\tilde{C}(u)$ for every $u \in A_i \setminus A_{i+1}$ is $(n^{1/2+1/k}+D)\cdot\min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$. By Claim 2 each vertex participates in at most $\tilde{O}(n^{1/k})$ trees, so Remark 1 implies that it will take only $\tilde{O}(n^{1/2+1/k}+D)$ rounds to compute the routing tables for all trees in parallel. We conclude that the total number of rounds is $(n^{1/2+1/k}+D)\cdot \min\{(\log n)^{O(k)}, 2^{\tilde{O}(\sqrt{\log n})}\}$.

5. 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). Furthermore, the distance computation can be done in time $O(k)$.*

Apply Theorem 5, which computes all the approximate pivots and approximate clusters. Each vertex $v \in V$ include in its sketch for every $u \in V$ so that $v \in \tilde{C}(u)$, the pair $(u, b_v(u))$, where $b_v(u)$ is the approximate distance to u computed in Section 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 3 every sketch is of size $O(n^{1/k}\log n)$. The algorithm that computes the distance estimation given two sketches is similar to that of [TZ05]. We state it formally in Algorithm 2.

Algorithm 2 `Dist`(u,v)

```
1: i ← 0;
2: w ← u;
3: while v ∉ C̃(w) do
4:     i ← i + 1;
5:     (u, v) ← (v, u);
6:     w ← ẑ_i(u);
7: end while
8: return d̂_i(u) + b_v(w);
```

Observe that the sketch contains all the relevant information for executing Algorithm 2. When the while loop terminates $v \in \tilde{C}(w)$, so it 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 variation of the analysis of [TZ05], similar in spirit to that of Section 4. Roughly speaking, on the stretch $2k-1$ achieved by [TZ05], we pay a multiplicative factor of $(1+O(\epsilon))^k$ due to the fact that distances are approximated, but this boils down to an $o(1)$ additive term, since $\epsilon = \frac{1}{48k^4}$. We leave the details to the reader.

6. REFERENCES

[ABLP90] Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *J. Algorithms*, 11(3):307–341, 1990.

[AGM04] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. Routing with improved communication-space trade-off. In *Distributed Computing, 18th International Conference, DISC 2004, Amsterdam, The Netherlands, October 4-7, 2004, Proceedings*, pages 305–319, 2004.

[Ber09] Aaron Bernstein. 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*, pages 693–702, 2009.

[Che13] Shiri Chechik. Compact routing schemes with improved stretch. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 33–41, 2013.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000.

[Cow01] Lenore Cowen. Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183, 2001.

[EGP03] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *J. Algorithms*, 46(2):97–114, 2003.

[Elk06a] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. *J. Comput. Syst. Sci.*, 72(8):1282–1308, 2006.

[Elk06b] Michael Elkin. 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.

[EN16] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *CoRR*, abs/1605.04538, 2016.

[GK13] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, pages 1–15, 2013.

[GKP98] Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for

minimum-weight spanning trees. *SIAM J. Comput.*, 27(1):302–316, 1998.

[GP03] Cyril Gavoille and David Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16(2-3):111–120, 2003.

[HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 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*, pages 146–155, 2014.

[HKN16] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. An almost-tight distributed algorithm for computing single-source shortest paths. 2016. To appear in STOC'16.

[KP98] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

[LP13a] Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 381–390, 2013.

[LP13b] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 375–382, 2013.

[LP15] Christoph Lenzen and Boaz Patt-Shamir. 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*, pages 153–162, 2015.

[LP16] Christoph Lenzen and Boaz Patt-Shamir. Personal communication, 2016.

[LPP16] Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. 2016.

[Nan14] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 565–573, 2014.

[NRS12] Nicolas Nisse, Ivan Rapaport, and Karol Suchan. Distributed computing of efficient routing schemes in generalized chordal graphs. *Theor. Comput. Sci.*, 444:17–27, 2012.

[Pel00a] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Pel00b] David Peleg. Proximity-preserving labeling schemes. *J. Graph Theory*, 33(3):167–176, March 2000.

[PR00] David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[SDP15] Atish Das Sarma, Michael Dinitz, and Gopal Pandurangan. Efficient distributed computation of distance sketches in networks. *Distributed Computing*, 28(5):309–320, 2015.

[SHK+12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 1–10, New York, NY, USA, 2001. ACM.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.