Efficient Distributed Source Detection with Limited Bandwidth

Christoph Lenzen
Massachusetts Institute of Technology
32 Vassar Street
02139 Cambridge, USA
clenzen@csail.mit.edu

David Peleg
Dept. of Computer Science and Applied Math.
Weizmann Institute of Science
76100 Rehovot, Israel
david.peleg@weizmann.ac.il

ABSTRACT

Given a simple graph $G = (V, E)$ and a set of sources $S \subseteq V$, denote for each node $v \in V$ by $\mathcal{L}_v^{(\infty)}$ the lexicographically ordered list of distance/source pairs $(d(s, v), s)$, where $s \in S$. For integers $d, k \in \mathbb{N} \cup \{\infty\}$, we consider the *source detection*, or (S, d, k)*-detection* task, requiring each node v to learn the first k entries of $\mathcal{L}_v^{(\infty)}$ (if for all of them $d(s, v) \leq d$) or all entries $(d(s, v), s) \in \mathcal{L}_v^{(\infty)}$ satisfying that $d(s, v) \leq d$ (otherwise). Solutions to this problem provide natural generalizations of concurrent breadth-first search (BFS) tree constructions. For example, the special case of $k = \infty$ requires each source $s \in S$ to build a complete BFS tree rooted at s, whereas the special case of $d = \infty$ and $S = V$ requires constructing a partial BFS tree comprising at least k nodes from every node in V.

In this work, we give a simple, near-optimal solution for the source detection task in the CONGEST model, where messages contain at most $\mathcal{O}(\log n)$ bits, running in $d + k$ rounds. We demonstrate its utility for various routing problems, exact and approximate diameter computation, and spanner construction. For those problems, we obtain algorithms in the CONGEST model that are faster and in some cases much simpler than previous solutions.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms, Path and circuit problems*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

Keywords

concurrent incomplete breadth-first search; distance and diameter computation; all-to-all shortest paths; compact routing; additive spanners; Bellmann-Ford

1. INTRODUCTION

This work concerns a basic network-algorithmic task hereafter referred to as the *source detection* or (S, d, k)*-detection* task. Given a subset $S \subseteq V$ of source nodes and a node $v \in V$, let $\mathcal{L}_v^{(\infty)}$ denote the (ascending) lexicographically ordered list of pairs $(d(v, s), s)$, where $s \in S$ and $d(v, s)$ is the length of a shortest path from v to s. The (S, d, k)*-detection* problem requires that each node $v \in V$ learns the first $\min\{k, \lambda_v^d\}$ entries of $\mathcal{L}_v^{\infty}$, where λ_v^d is the number of sources $s \in S$ satisfying that $d(s, v) \leq d$. The paper develops a time efficient algorithm for this task in the CONGEST model, and illustrates its usefulness by discussing some of its applications.

To motivate the source detection task, and illustrate its scope of applicability, let us discuss two application domains where this task can be utilized. The first is the construction of shortest-path spanning trees, which is one of the most fundamental problems in algorithmic graph theory. Shortest-path spanning trees, and breadth-first search (BFS) trees in the special case of unweighted graphs, play a key role in a wide variety of applications in graph algorithms, communication networks, distributed and parallel computing, and many other areas of computer science. In many cases one is interested in but one complete shortest path (or BFS) spanning tree. Nevertheless, frequently some other variant is in demand. In particular, for certain applications one is interested in finding *all* shortest path trees (or BFS trees) of the given graph, one for each source node. In other applications, one is interested in a *partial* shortest path tree (or BFS tree), e.g., one spanning all nodes at distance up to d from the source. A basic task encompassing all of the above variants of the problem is the *multiple source partial BFS trees* problem. This problem is characterized by two parameters: a subset $S \subseteq V$ of source nodes, and an integer d. The problem is to construct, for each source node $s \in S$, a partial BFS tree T_s^d spanning all the nodes at distance at most d from s. Clearly, this problem can be cast as a special case of the source detection task.

A second application domain concerns *fault-tolerant centers* and *facility location* problems. Consider a setting where it is necessary to place some *service centers*, or *production facilities*, in some of the nodes of the graph at hand. Those centers are planned to provide certain crucial services to some other nodes of the network, acting as *clients*. A client $v \in V$ selects the nearest server, $\varphi(v)$, and gets served by that server. Hence the relevant efficiency measure is the distance $d(v, \varphi(v))$ from v to its designated server (or sometimes the product of that distance times the *demand* issued by v). Hence it is important to optimize the spread of servers in the network. The problem of selecting locations for centers or facilities is typically NP-hard, and is not considered here. Rather, we are concerned with the situation *after* a set S of service centers has already been fixed. Let us further concentrate on a scenario where service centers might occasionally *fail*, forcing their clients to re-compute their choice of a temporary designated server. In network settings, where such failures occur frequently, it may be important to devise an efficient distributed algorithm for re-computing the assignment of centers to clients. In particular, one solution approach would be to prepare to the eventuality of (up to f) failures of centers by requiring each client node to learn in advance a list of the $f+1$ service centers closest to it, so that in case some of its closest centers fail, it may switch without delay to using the next closest center. (Here we assume that failures make a center incapable of providing service, but the associated network node can still be used for communiation.) This fault-tolerant centers problem maps to $(S, \infty, f+1)$-detection. If, in addition, clients refuse using very distant centers, then a threshold d, which is smaller than the diameter D of the network (namely, the maximum distance between network nodes) may be imposed on the maximum allowed service distance, yielding an instance of the $(S, d, f+1)$-detection problem.

The complexity of the source detection task clearly depends on the model at hand. Here we focus on the CONGEST model (cf. [19]), formally defined in Section 3, which takes bandwidth limitations into account. The source detection task has a straightforward optimal algorithm in a distributed model with no bandwidth limitations, such as the LOCAL model (which is identical to the CONGEST model except that it imposes no restrictions on the size of messages). Our main goal in this paper is to develop a time-efficient distributed algorithm for the (S, d, k)-detection task, and in turn, for the multiple source partial BFS trees problem, in the CONGEST model, where bandwidth is limited. It is noteworthy that as a byproduct of our technique, one may also compute for every node v the next hop on the shortest paths to its sources. This makes our algorithm a powerful tool for computing or approximating the network diameter, routing, and many other related tasks.

Contributions

Our main contribution is the introduction of the basic paradigm of (S, d, k)-detection in networks and an efficient distributed algorithm for solving it in the CONGEST model in time $\min\{d, D\} + \min\{k, |S|\}$ (Section 4). In contrast, solving (S, d, k)-detection using existing techniques results in time complexity $\Omega(|S|)$. Hence, we obtain improvements of $\Theta(n)$ for extreme cases (where n is the number of nodes in the network). Note that the time complexity of our solution is not asymptotic, and the dependency on d and k is optimal due to a trivial lower bound. (With respect to k this bound requires that only one identifier fits into a message; otherwise the lower bound, as well as the upper bound of the algorithm, decrease by the respective factor in the k-summand.) Even for the special case of (S, ∞, ∞)-detection (or, equivalently, (S, D, n)-detection), the appealing simplicity of our approach therefore results in $(1+o(1))$-optimal solutions for several tasks granted that $\log D \in o(\log n)$.[1] While this is a mere constant-factor improvement over the results from [15, 20], we consider it valuable because it affects the complexity of some fundamental tasks.

We further motivate this result by leveraging it for applications in Section 5:

- **All-to-all shortest paths and diameter:** Each node needs to learn the distance and next routing hop to each other node. From this information one can also compute the diameter in $\mathcal{O}(D)$ time. Variations of this problem result from considering different wake-up mechanisms. In all cases, we obtain a deterministic $(n + \mathcal{O}(D))$-round solution.
- **$3/2$-approximation of the diameter:** The task is to compute an estimate $\tilde{D}$ of the diameter in the range $[2D/3, D]$. Utilizing our solution for (S, d, k)-detection, this can be done by a randomized algorithm within $\mathcal{O}(\sqrt{n}\log n + D)$ rounds.
- **Approximation of all-to-all shortest paths:** Here it is required to construct a mechanism allowing all nodes to determine the next routing hop on a path that is at most by some multiplicative *stretch* factor longer than a shortest path to the destination. Furthermore, we allow assigning (new) labels to the nodes, replacing their identifiers, as otherwise nodes cannot learn the entire namespace quickly. For any $\gamma \in \{1, \ldots, \log n\}$, it is possible to construct such a mechanism ensuring stretch $\mathcal{O}(\gamma)$ by a randomized preprocessing algorithm requiring $\tilde{\mathcal{O}}(n^{1/2+1/\gamma} + D)$ rounds.[2]
- **Compact Routing:** The task is the same as in the previous case, but we now also ask for low memory requirements by the routing tables. We can construct such compact routing tables achieving stretch $\mathcal{O}(\gamma^2)$ and using $\mathcal{O}(n^{1/\gamma}\log^{\mathcal{O}(1)} n)$ memory by a randomized preprocessing algorithm requiring $\tilde{\mathcal{O}}(n^{1/2+1/(2\gamma)} + D)$ rounds.
- **Constructing 2-additive spanners:** We seek to select a small subset of the edges so that in the resulting graph, distances do not increase by more than 2. We obtain a randomized construction of running time $\mathcal{O}(\sqrt{n}\log n + D)$ that produces 2-additive spanners with $\mathcal{O}(n^{3/2}\log n)$ edges.

For all the listed tasks, previous solutions exist, but are slower than the ones we provide. In addition, in many cases our approach simplifies the resulting algorithms. All randomized constructions succeed with high probability.

Before presenting our algorithm for (S, d, k)-detection, we discuss related work in the following section and introduce the system model in Section 3.

2. RELATED WORK

The all-pairs shortest path (APSP) problem has been studied extensively in the sequential setting, and was also given several solutions in the distributed setting [3, 8, 14, 16, 23]. The algorithm of [16] is fast ($O(n)$ time) but involves using large messages, hence does not apply in the CONGEST model. The algorithm of [3] uses short ($O(\log n)$

[1]Otherwise the size of distance counters is not negligible, and we can conclude asymptotic optimality only.

[2]Here, the $\tilde{\mathcal{O}}(x)$ notation hides any polylogarithmic factors in x.

bits) messages, hence it can be executed in the CONGEST model, but it requires time $O(n \log n)$, and moreover, it applies only to the special family of BHC graphs, which are graphs structured as a balanced hierarchy of clusters. Most of the distributed algorithms for the APSP problem aim at minimizing the message complexity, rather than the time. The algorithm of [14] requires time $O(n^2)$.

In the CONGEST model, a trivial lower bound of $\Omega(n)$ applies, which has independently been asymptotically matched by two algorithms [15, 20]. Our solution improves on these works in that we achieve an optimal multiplicative constant with respect to n.

Note that the network diameter can easily be computed exactly by first solving the APSP problem. It was shown in [12] that, in the CONGEST model, computing the exact diameter requires time $\tilde{\Omega}(n)$; this nontrivial lower bound holds even for networks of constant diameter. Hence these results also imply near-optimal algorithms for computing the exact diameter in the CONGEST model. In the LOCAL model, computing the diameter exactly trivially requires exactly D rounds (using large messages). An efficient algorithm for doing this in $\mathcal{O}(D)$ rounds (using $O(n \log n)$ bit messages) is found in [2].

Similar to the exact setting, an approximation of the diameter can be obtained by any algorithm that computes an approximation of APSP (see for example [4, 9, 10]). A sequential algorithm that approximates the diameter in weighted directed graphs in $\tilde{O}(n^2 + |E|\sqrt{n})$ steps was presented in [1]. This algorithm returns an acyclic path of length at least $2D/3$. It has been leveraged for computing a distributed 3/2-approximation in the CONGEST model in $\tilde{\mathcal{O}}(\sqrt{n}D)$ randomized rounds [20]. An algorithm that computes a $(1+\epsilon)$-approximation of the diameter in time $O(n/D+D)$ was presented in [15], and combining these two algorithms yielded a randomized 3/2-approximation within time $\mathcal{O}(n^{3/4}+D)$. This running time was very recently improved to $\tilde{\mathcal{O}}(\sqrt{n}+D)$ by the authors of [15, 20], relying also on a new result of [22]. The same result follows from our technique, solely relying on the original algorithm from [1], and with a smaller leading constant factor. These results are tight up to a factor of $\mathcal{O}(\log^2 n)$, as is shown in [12]: even approximating the diameter by a factor of $3/2-\epsilon$ may sometimes require $\tilde{\Omega}(\sqrt{n})$ time, for any constant $\epsilon > 0$. (In contrast, a 2-approximation for D is easily computed in time $\mathcal{O}(D)$ by constructing one BFS tree and measuring its depth.)

Our results with respect to approximation of all-to-all shortest paths build on recent results for this problem in weighted graphs [18]. Specialized to the unweighted case, our solution to (S,d,k)-detection can replace a subroutine from [18] in order to achieve a stretch of $\mathcal{O}(\gamma)$ within time $\tilde{\mathcal{O}}(n^{1/2+1/\gamma}+D)$. This notably improves the trade-off between stretch and running time; for the same running time bound, the stretch guarantee is $\mathcal{O}(\gamma \log \gamma)$ for the algorithm from [18]. In contrast to the results discussed above, this non-constant improvement of the running time cannot be achieved by previous algorithms for building multiple BFS trees concurrently. The latter is due to the fact that the algorithm from [18] requires to build partial trees only, i.e., $k \ll n$ in the corresponding instances of (S,d,k)-detection. The same holds for our statement on compact routing, where we also build on [18]; however, [18] itself does not provide a result regarding compact routing, as we need to exploit that distances are unweighted in order to store routing paths efficiently in a distributed manner. Both the shortest paths and the compact routing problems have been studied extensively (cf. [4, 6, 13, 17, 19, 21, 24] and references). However, most previous work on these problems either focused on efficient performance (stretch, memory) and ignored the time-efficiency of the preprocessing stage, provided time-efficient seqential (centralized) preprocessing algorithms, or gave time-efficient distributed algorithms in the LOCAL model (which allows unbounded message size).

There are essentially three known constructions for additive spanners. Constructions for 2-additive spanners with $\mathcal{O}(n^{3/2})$ edges were presented and expanded in [1, 9, 11, 21, 25]. An efficient construction for 6-additive spanners with $\mathcal{O}(n^{4/3})$ edges was later presented in [5]. Recently, a construction for 4-additive spanners with $\mathcal{O}(n^{7/5})$ edges was presented in [7]. In the distributed case, the construction for 2-additive spanners requires to concurrently build $\tilde{\mathcal{O}}(\sqrt{n})$ complete BFS trees. Hence, it can be implemented in the CONGEST model utilizing the results from [15] in $\tilde{\mathcal{O}}(\sqrt{n}+D)$ randomized rounds. Again, employing our routine yields the same result, but with a smaller multiplicative constant in the dominating term of the running time.

3. MODEL

We follow the CONGEST model as described in [19]. The distributed system is represented by a simple, connected graph $G=(V,E)$ of $n=|V|$ nodes. Each node $v \in V$ has a unique identifier of $\mathcal{O}(\log n)$ bits that we identify with the node, i.e., v denotes both the node and its identifier. Communication is synchronous; in each round, each node $v \in V$ can send a (possibly distinct) message comprising $\mathcal{O}(\log n)$ bits to each of its neighbors $\mathcal{N}_v = \{w \in V \mid \{v,w\} \in E\}$. Initially, node $v \in V$ is aware of $\mathcal{N}_v$ only. In the case of randomized algorithms, each node has access to an independent, infinite, and unbiased source of random bits.

We use the following additional notation. A path p of length d is a sequence of nodes $(v_0,\ldots,v_d)$ satisfying that $\{v_{i-1},v_i\} \in E$ for all $i \in \{1,\ldots,d\}$. For $v,w \in V$, let $d(v,w)$ denote the minimal length of any path from v to w. The diameter of the graph is then $D = \max_{v,w\in V}\{d(v,w)\}$, the maximum distance between nodes in the graph.

We consider a setting where a subset $S \subseteq V$ of the nodes in the graph is marked as *source* nodes. In the distributed context, each node v maintains a variable s_v set to $s_v = v$ if $v \in S$ and $s_v = \bot$ otherwise. Given a node $v \in V$, the *S-proximity-list* (or simply *S-list*) of v, denoted $\mathcal{L}_v^{(\infty)}$, is the (ascending) lexicographically ordered list of pairs $(d(v,s),s)$, for every $s \in S$. Here, by ascending lexicographical order we mean that $(d(v,s),s) < (d(w,s'),s')$ iff $d(v,s) < d(w,s')$ or $d(v,s) = d(w,s')$ and $s < s'$ (where the latter is determined by an order on the node identifiers). For an integer $d > 0$, denote the prefix of v's S-proximity-list consisting of the sources at distance at most d from v by $\mathcal{L}_v^{(d)} = \{s \in S \mid d(v,s) \le d\}$, and let $\lambda_v^d = |\mathcal{L}_v^{(d)}|$.

4. DETECTING NEARBY SOURCES

In this section, we present an efficient deterministic algorithm for the *source detection* task. The *(S,d,k)-detection* problem requires that each node $v \in V$ learns the first

$\min\{k, \lambda_v^d\}$ entries of $\mathcal{L}_v^{(\infty)}$ (i.e., either all sources up to distance d or just the k closest ones, if there are too many).

In absence of restrictions on bandwidth, the straightforward distributed implementation of the classical Bellmann-Ford algorithm matches the requirements of this problem perfectly. Each node v maintains a list L_v of the distance/source pairs that it knows about. The list is initially empty if $s_v = \perp$ and contains $(0, s_v)$ otherwise. In each round, each node v broadcasts L_v. Upon reception of such a message, for each received pair (d_s, s) for which there is no own pair $(d'_s, s) \in L_v$, it adds $(d_s{+}1, s)$ to L_v. After d rounds, v knows the sources within distance d from itself, and their correct distance; thus it is able to order the source/distance pairs correctly. This approach results in concurrently constructing BFS trees for all sources $s \in S$ up to depth d. (Note that the algorithm as presented does not store the parent of each node, but the respective modification is trivial.)

The situation gets more involved if nodes may only send $\mathcal{O}(\log n)$ bits in each round, as this implies that they can communicate only a constant number of entries out of each list per round. Trivially, one can simply dedicate $|S|$ rounds to the simulation of a single round of the unconstrained algorithm. This results in a running time of $\mathcal{O}(|S|d)$ for an algorithm solving (S, d, k)-detection for arbitrary k. More generally, sending only the $k \in \mathbb{N}$ smallest distance/source pairs (according to the lexicographical order) solves (S, d, k)-detection in kd rounds (cf. [18]).

Observe, however, that a trivial lower bound of $d + k - 1$ holds if one can send only one identifier per round. To see this, consider a chain of nodes of length $d - 1$ and append k sources to one of its endpoints; since the other endpoint needs to receive the k source identifiers sequentially and the first one arrives no earlier than round d, the lower bound follows.

In light of this lower bound, the question arises whether one can solve the task in $\mathcal{O}(d + k)$ rounds in the CONGEST model. Clearly, such an improved time bound requires a more sophisticated strategy, as resending the entire list L_v is wasteful if there are no relevant updates. In [15, 20], n BFS constructions are started in a staggered fashion, essentially ensuring that on each round, at most one update needs to be sent on every edge. In [15], a second algorithm resolves this issue slightly differently, by giving the updates preference according to the (total) order of source identifiers and requiring that in each iteration only one update is transmitted over each edge, i.e., for nodes v and $w \in \mathcal{N}_v$ either v sends an update to w or vice versa, not both. It is shown that this ensures that each node will receive accurate distance information for each source with the first update it receives regarding the respective source. Neither of these strategies is sufficient to achieve a complexity bound of $\mathcal{O}(d + k)$ for the (S, d, k)-problem for the entire range of parameters, though, as they use the *identifiers* as the primary means of ordering, rather than the *distances*.

Consequently, in this paper we use the arguably even simpler mechanism that assigns priorities according to distance: In each round, each node v just sends to all neighbors the smallest pair in L_v that it has not transmitted yet. Note that this entails that a node might possibly send multiple updates regarding the same source, each notifying its neighbors that it learned that the respective source is in fact closer to it than previously announced. The pseudo-code of this approach is given by Algorithm 1.

We first establish correctness of the algorithm, by arguing that eventually the lists L_v do not change any more, and this final state must satisfy that all source/distance pairs reflect actual distances in G. In order to formalize this statement, denote by $L_v^{(r)}$ the state of L_v at the beginning of round $r \in \mathbb{N}$ of the algorithm.

LEMMA 4.1. *For any graph and any $S \subseteq V$, there is some round $r_0 \in \mathbb{N}$ such that no node $v \in V$ sends messages or modifies L_v in rounds $r \geq r_0$. Moreover, $L_v^{(r_0)} = \mathcal{L}_v^{(\infty)}$, i.e., $d_s = d(v, s)$ for every $(d_s, s) \in L_v^{(r_0)}$.*

PROOF. Each list is initially empty or contains some pair $(0, s)$. For a source $s \in S$ and round $r \in \mathbb{N}$, define

$$d_{\max}(s, r) = \max\{d_s \in \mathbb{N} \mid \exists v \in V : (d_s, s) \in L_v^{(r)}\}.$$

Clearly, $d_{\max}(s, r+1) \leq d_{\max}(s, r) + 1$, and in case of equality, some node v that did not have any pair $(d_s, s) \in L_v^{(r)}$ added such a pair to L_v in round r. Hence, $d_{\max}(s, r)$ is uniformly bounded by $n - 1$ for all s and r. It follows that each node $v \in V$ modifies its variables only finitely often: For each $s \in S$, it can add at most $n - 1$ pairs (d_s, s) to L_v during the course of the algorithm, and for each pair, it will initialize $\text{sent}_v(d_s, s)$ to FALSE and set it to TRUE at most once.

Now consider a round r in which no node modifies the contents of its variables; we just showed that such a round exists. In particular, no node sends a message (as this results in modifying the respective sent variable), implying that the same must hold for the next round (because no node v had an unsent entry in L_v, and contents of variables did not change). Consequently, no node v adds an entry to L_v in round $r + 1$, implying that no variables are changed in this round either. Repeating this argument inductively, we see that for the minimal round r_0 in which no variables change, it holds that the algorithm has reached a steady state in which all local variables, in particular the lists L_v, become invariant.

It remains to show that for all $s \in S$ and $v \in V$ it holds that $(d(v, s), s) \in L_v^{(r_0)}$ (this is sufficient because $(\cdot, s)$ is deleted from L_v before adding such an entry). We first prove that we have $(d_s, s) \in L_v^{(r_0)}$ with $d_s \leq d(v, s)$. To see this, consider a shortest path $(v_0 = s, v_1, \ldots, v_{d(v,s)} = v)$ from s to v. Since s initializes L_s to $(0, s)$ and $\text{sent}(0, s)$ to FALSE, it will send $(0, s)$ in the first round. We claim that each node v_i, $i \in \{1, \ldots, d(v, s)\}$, will eventually receive a message (d_{i-1}, s) with $d_{i-1} \leq i - 1$. This follows by induction anchored at $i = 1$, where the step consists of observing that if v_i receives (d_{i-1}, s), it must have already sent or will eventually send a message (d_i, s) with $d_i \leq d_{i-1} + 1$.

To complete the proof, we now show that also $d_s \geq d(v, s)$. At initialization, the contents of all lists satisfy that they represent accurate distance information, i.e., for the pairs $(0, s) \in L_s$, $s \in S$, it holds that $0 = d(s, s)$. Hence, assume for contradiction that the claim is false and suppose that v is a node that adds (d_s, s) with $d_s < d(v, s)$ to L_v in a minimal round $r \geq 1$. It follows that v received a message $(d_s - 1, s)$ from some neighbor $w \in \mathcal{N}_v$. Thus, this neighbor added $(d_s - 1, s)$ to L_w in an earlier round (or at initialization). However, since $d(w, s) \geq d(v, s) - 1$, it follows that there is an earlier violation of the claim, contradicting the assumption that r is minimal (or the observation that at initialization the claim holds). Thus indeed $d_s = d(v, s)$ for all $(d_s, s) \in L_v^{(r_0)}$, $v \in V$, concluding the proof. $\square$

Algorithm 1: DBF(S): Distributed Bellmann-Ford at node $v \in V$. Each node initially only needs to know whether it is in S itself. As stated, the algorithm does not terminate. The local termination condition depends on the application and is discussed later on.

```
1  $L_v := ()$        // list of distance/source pairs $(d_s, s) \in \mathbb{N}_0 \times S$, lexicographically ordered
2  $\text{sent}_v : L_v \to \{\text{TRUE}, \text{FALSE}\}$        // whether a pair in $L_v$ has already been sent by $v$
3  if $v \in S$ then
4      $L_v := ((0, v))$
5      $\text{sent}_v(0, v) := \text{FALSE}$
6  // each iteration of the loop takes one round
7  while TRUE do
8      if $\exists (d_s, s) \in L_v : \text{sent}(d_s, s) = \text{FALSE}$ then
9          $(d_s, s) := \text{argmin}\{(d'_s, s') \in L_v \mid \text{sent}_v(d'_s, s') = \text{FALSE}\}$
10         send $(d_s, s)$ to all neighbors
11         $\text{sent}_v(d_s, s) := \text{TRUE}$
12     for received $(d_s, s)$ from some neighbor do
13         if $\nexists (d'_s, s) : d'_s \le d_s + 1$ then
14             $L_v := L_v \setminus \{(\cdot, s)\}$        // remove outdated entry, if there is one
15             $L_v := L_v \cup \{(d_s + 1, s)\}$
16             $\text{sent}_v(d_s + 1, s) := \text{FALSE}$
```

Since the algorithm accepts that we might "waste" messages by sending updates that contain incorrect distance information, we need to show that there are not too many such wasted messages. As the previous lemma shows that eventually the algorithm determines correct values, this can be rephrased in terms of bounding the number of rounds for which the k smallest entries of L_v might change. To this end, for an entry $(d_s, s) \in L_v^{(r)}$, let $\ell_v^{(r)}(d_s, s)$ denote its index in the (lexicographically ordered) list $L_v^{(r)}$.

LEMMA 4.2. *For every node $v \in V$ and round $r \in \mathbb{N}$ of Algorithm 1,*

(i) v does not send a message (d_s, s) with

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

in round r,

(ii) v does not add a pair (d_s, s) to L_v with

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

in round r.

PROOF. Clearly, if (i) holds in round $r \in \mathbb{N}$, then (ii) also holds in round r because nodes increase d_s by 1 upon adding pairs to L_v. Moreover, (i) is trivially satisfied in round 1, as $\ell_v^{(r)}(d_s, s) \ge 1$ for all v, d_s, r, and s. Hence, in order to prove the claim, assume for contradiction that there is a minimal round $r > 1$ such that (i) is violated, while (ii) holds for all rounds $r' < r$.

Thus, some node $v \in V$ sends a message (d_s, s) with

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

in round r. Note that $d_s \neq 0$, i.e., (d_s, s) is not one of the entries of L_v at initialization since these are all sent in the first round. This implies that v has added (d_s, s) to L_v in some round $1 \le r' < r$. Because $\ell_v^{(r)}(d_s, s)$ is non-decreasing with r, we have that

$$\ell_v^{(r)}(d_s, s) \ge \ell_v^{(r')}(d_s, s).$$

By Statement (ii) for round r', this entails that

$$r > d_s + \ell_v^{(r)}(d_s, s) \ge d_s + \ell_v^{(r')}(d_s, s) > r',$$

yielding that in fact $r' \le r - 2$ as the involved values are all integer. Therefore, considering that v did not send (d_s, s) in round $r - 1$ but in round r, it must have sent a different pair $(d_{s'}, s')$ in round $r - 1$. By Statement (i) for round $r - 1$, it holds that

$$d_{s'} + \ell_v^{(r-1)}(d_{s'}, s') \ge r - 1.$$

However, since $(d_{s'}, s')$ must be lexicographically smaller than (d_s, s), we have that

$$d_{s'} + \ell_v^{(r-1)}(d_{s'}, s') < d_s + \ell_v^{(r-1)}(d_s, s).$$

Overall, we arrive at the contradiction that

$$\begin{aligned} d_s + \ell_v^{(r-1)}(d_s, s) &> d_{s'} + \ell_v^{(r-1)}(d_{s'}, s') \\ &\ge r - 1 \\ &\ge d_s + \ell_v^{(r)}(d_s, s) \\ &\ge d_s + \ell_v^{(r-1)}(d_s, s), \end{aligned}$$

where the last step once more applies the fact that $\ell_v^{(r)}(d_s, s)$ is non-decreasing with r. □

Together, both lemmas show that the algorithm is a near-optimal solution to the (S, d, k)-detection problem.

LEMMA 4.3. *Given an instance of the (S, d, k)-detection problem, for every $v \in V$ and for any round r of an execution of Algorithm 1 with*

$$r \ge r(S, d, k) = \min\{d, D\} + \min\{k, |S|\},$$

$L_v^{(r)}$ truncated to the (up to) k first entries $(d_s, s) \in L_v^{(r)}$ with $d_s \le d$ solves (S, d, k)-detection.

PROOF. Without loss of generality, $d \le D$ (no source can be further away) and $k \le |S|$ (for any $k \ge |S|$ all nodes need to learn about all sources). By Lemma 4.1, there is a round $r_0 \in \mathbb{N}$ such that for all $v \in V$ and $r' \ge r_0$, $L_v^{(r')} = L_v^{(r_0)} = \mathcal{L}_v^{(\infty)}$. By definition, the first (up to) k entries $(d_s, s) \in L_v^{(r_0)}$

with $d_s \leq d$ thus solve (S, d, k)-detection. By Lemma 4.2, these entries do not change in any round $r' \geq d + k = r(S, d, k)$, completing the proof. □

THEOREM 4.4. *Algorithm 1 solves (S, d, k)-detection in time* $\min\{d, D\} + \min\{k, |S|\}$.

We conclude this section with two remarks. First, the proof of Lemma 4.2 can be generalized to show that if up to $\alpha \in \mathbb{N}$ list entries are sent in each round, then node v will not add pairs with $(d_s, s) + \lceil \ell_v^{(r)}(d_s, s)/\alpha \rceil \leq r$ to L_v in round r. Therefore, if α list entries may be sent, then (S, d, k)-detection can be solved within $\min\{d, D\} + \lceil \min\{k, |S|\}/\alpha \rceil$ rounds. Likewise, we have a trivial lower bound of $d + \lceil k/\alpha \rceil$ for (S, d, k)-detection in this setting. Our technique is thus, up to the bits encoding the distance information d_s for an entry (d_s, s), optimal for the entire parameter range. In particular, if $\log D \in o(\log n)$, then it is $(1 + o(1))$-optimal for arbitrary message size.

Secondly, the algorithm can be generalized to permit that subsets of nodes are identified into a single "source cluster" $s \in S$. This is done simply by passing all nodes v in cluster s the input $s_v = s$. The feasibility of this modification follows from a simple simulation argument given in [18] that also applies in the unweighted case.

5. APPLICATIONS

In this section, we present applications of our solution to the generic (S, d, k)-detection problem.

Exact All-to-all Shortest Paths and Diameter

In the distributed version of the all-to-all shortest paths problem, we require that each node v learns for each other node w the next routing hop on a shortest path to w. This is equivalent to constructing for each node v a BFS tree rooted at it. The diameter then can be determined as the maximum of the depths of the trees. These tasks can be solved efficiently by means of Algorithm 1.

COROLLARY 5.1. *All-to-all shortest paths routing tables can be constructed in $n + \mathcal{O}(D)$ time by a uniform algorithm (i.e., n and D are initially unknown to the nodes).*

PROOF. With a slight modification of Algorithm 1, constructing BFS trees rooted at the sources comes for free: Instead of adding pairs (d_s, s) to the list L_v, node v stores triples (d_s, s, p), where the sender p of the message $(d_s - 1, s)$ causing the triple to be stored becomes the parent of v in the BFS tree rooted at s; p then is the next routing hop from v to s.

To solve the problem, we first construct a single BFS tree rooted at some node r (e.g., the one with smallest identifier), determine its depth (i.e., a 2-approximation of D) and count the number of nodes, and finally distribute these values to all nodes. This can be done by standard techniques in $\mathcal{O}(D)$ rounds (even with asynchronous wake-up). Subsequently r initiates a run of (the modified version of) Algorithm 1 (another D rounds) with $S = V$ that is stopped at all nodes after $n + \tilde{D}$ rounds, where $D \leq \tilde{D} \leq 2D$ is twice the depth of the BFS tree rooted at r. By Theorem 4.4, all nodes will have accurate distance information on all other nodes, i.e., all constructed trees are indeed complete BFS trees. □

COROLLARY 5.2. *All nodes can learn the diameter D in $n + \mathcal{O}(D)$ rounds, without any prior knowledge on n or D.*

PROOF. We construct for each node a BFS trees as in the previous corollary. Note that along with their parent with respect to each root, nodes store the distance to this parent. We collect $D = \max_{v \in V}\{\max_{s \in V}\{d(v, s)\}\}$ over the BFS tree rooted at r and distribute it to all nodes. As this takes at most $2D$ rounds, by the previous corollary the overall running time is $n + \mathcal{O}(D)$. □

3/2-Approximation of the Diameter

A deterministic centralized algorithm computing a 3/2-approximation to the diameter is given in [1]. A randomized distributed implementation of this algorithm in the CONGEST model that runs in $\tilde{\mathcal{O}}(\sqrt{n}D)$ rounds with high probability[3] (w.h.p.) is presented in [20]. This running time was very recently improved to $\tilde{\mathcal{O}}(\sqrt{n} + D)$ by the authors of [15, 20], relying also on a new result of [22]. The same result follows from Algorithm 1 in a simpler way and with a smaller leading constant factor.

COROLLARY 5.3. *An estimate $\tilde{D}$ of the diameter D satisfying that $D \leq \tilde{D} \leq 3D/2$ can be computed by a uniform randomized algorithm in $\mathcal{O}(\sqrt{n \log n} + D)$ rounds w.h.p.*

PROOF. For a parameter $\sigma \in \{1, \ldots, n\}$, the algorithm of [1] operates as follows.

1. For each node $v \in V$, compute the partial BFS tree consisting of the σ nodes closest to v.
2. Select a vertex v_0 whose partial BFS tree is deepest.
3. For each node in the partial BFS tree of v_0, construct a complete BFS tree.
4. Select a subset $W \subseteq V$ such that each node has a node from W among its σ closest nodes.
5. For each node $w \in W$, construct the complete BFS tree rooted at w.
6. Output 3/2 times the maximal depth of all constructed partial and complete BFS trees.

As explained before, nodes can obtain n and a 2-approximation of D in $\mathcal{O}(D)$ rounds. Thus, each node can compute the choice $\sigma = \lceil \sqrt{n \log n} \rceil$. By Theorem 4.4, running Algorithm 1 with $S = V$ for 2σ rounds (with the modification from Corollary 5.1) is sufficient to determine, at each node, the distances to the closest σ nodes; we will see that there is no need to explicitly construct all partial BFS trees. Taking the maximum such distance over all nodes, we can select and publish v_0 (using a convergecast over the distinguished BFS tree we already used to determine n and a 2-approximation $\tilde{D}$ of D). Since v_0 already knows its closest σ nodes and the next hop on a shortest path to them, informing all σ nodes in its partial subtree of their membership can be performed within another 2σ rounds (the depth of the tree is at most σ and at most σ messages need to be sent over each edge). Completing the third step of the algorithm thus can be done by calling Algorithm 1 with the nodes in the partial BFS tree of v_0 as source set S and terminating the instance after $\sigma + \tilde{D}$ rounds.

For the fourth step, we simply select each node with uniform and independent probability $\mathcal{O}(\log n/\sigma)$. As each node

[3]That is, with probability at least $1 - 1/n^c$ for an arbitrary predefined constant $c > 0$.

could be covered by $\sigma + 1$ nodes, by Chernoff's bound all nodes are covered and $|W| \in \Theta(n \log n/\sigma)$ w.h.p. Consequently, Theorem 4.4 shows that running Algorithm 1 a third time with source set W for $\mathcal{O}(n \log n/\sigma) + \tilde{D}$ rounds will, w.h.p., for each $w \in W$ correctly construct the complete BFS tree rooted at w. Finally, the last step of the algorithm is done by a convergecast and flooding on the initially constructed distinguished BFS tree.

Overall, we obtain a running time of $\mathcal{O}(\sigma + n \log n/\sigma + \tilde{D}) = \mathcal{O}(\sqrt{n \log n} + D)$ w.h.p. Since, w.h.p., our algorithm performs the same steps as a correct execution of the centralized algorithm of [1], the approximation ratio follows from the analysis of [1]. □

Approximation of All-to-All Shortest Paths

In [18], it is shown how to distributedly construct routing tables for approximate all-to-all shortest path, where the actual routing path may be longer than an optimal one by a multiplicative stretch. The bounds in [18] hold for the weighted case, i.e., there is a non-negative cost incurred by each traversed edge. A central subroutine employed in this work is a solution to (S, d, k)-detection with certain parameters, and the utilized subroutine is the aforementioned trivial translation of the Bellmann-Ford algorithm to the CONGEST model of running time $\mathcal{O}(dk)$. As a result, [18] needs to overcome the obstacle of the multiplicative running time by a fairly involved construction. This comes at a cost: For an overall running time bound of $\tilde{\mathcal{O}}(n^{1/2+1/\gamma} + D)$, the algorithm incurs a stretch of $\mathcal{O}(\gamma \log \gamma)$ and has the disadvantage that the destination's label must be known in order to find a routing path of bounded stretch;[4] the latter issue can be resolved, however only by suffering a much larger stretch of $\mathcal{O}(\gamma^3)$. For the unweighted case, the improved performance of Algorithm 1 permits to construct the required structure, ensuring a stretch of $\mathcal{O}(\gamma)$, in $\tilde{\mathcal{O}}(n^{1/2+1/\gamma} + D)$ rounds.

Corollary 5.4. *In $\tilde{\mathcal{O}}(n^{1/2+1/\gamma} + D)$ rounds, one can assign a unique label $\psi(v)$ of size $\mathcal{O}(\log n)$ to every node v and construct routing tables, so that for each label ψ, w.h.p. the routing tables indicate a routing path to the node w with $\psi(w) = \psi$ that is at most a factor $\mathcal{O}(\gamma)$ longer than the shortest path between v and w. Moreover, the set of labels is $\{1, \dots, n\}$.*

Proof. The construction in [18] combines a *short-range* and a *long-range* scheme. Using Algorithm 1 one can reduce the number of levels in the hierarchical short-range scheme to one as follows. The goal of the short-range construction is to provide a (small) top-level set of skeleton nodes $\mathcal{S}$ and, for each pair of nodes $v, w \in V$ that are not sufficiently far apart to suffer only a constant stretch when routing via the closest nodes in $\mathcal{S}$, to provide appropriate routing paths.

The set $\mathcal{S}$ is a uniformly random subset of V and therefore, if each node knows how to route to the $\mathcal{O}(n \log n/|\mathcal{S}|)$ nodes closest to it, there will be a node from $\mathcal{S}$ among these w.h.p. With Algorithm 1 at hand, we can achieve this in $\mathcal{O}(n \log n/|\mathcal{S}|)$ rounds using source set V (as already used in the proof of Corollary 5.3). If node v now cannot route to some node w using this information, it follows that its closest node $s_v \in \mathcal{S}$ is closer than w. Moreover, the closest node $s_w \in \mathcal{S}$ to w satisfies that $d(w, s_w) \leq d(w, s_v) \leq d(v, w) + d(v, s_v) \leq 2d(v, w)$; another application of the triangle inequality shows that $d(s_v, s_w) \leq 4d(v, w)$. Therefore, routing via s_v and s_w incurs a constant stretch.

[4]The algorithm needs to assign new labels to the nodes, as otherwise it is infeasible to achieve a running time that is sublinear in n on all graphs of $D \in o(n)$ [18].

The remaining ingredients to the routing scheme (relabeling of the nodes and constructing the routing paths from v to s_v, s_v to s_w, and s_w to w) can be taken from [18] without modification (where we use the variant that assigns labels $1, \dots, n$ from Theorem 5.9). In particular, the issues regarding the constructed labels that arise from the use of multiple levels in the short-range scheme are trivially resolved. □

Compact Routing Tables

In [18], a variant of the routing scheme is used to compute *distance sketches* of size $\mathcal{O}(n^{1/\gamma} \log^{\mathcal{O}(1)} n)$, computing labels such that each node v, given the label $\psi(w)$ of another node w, is capable of computing an $\mathcal{O}(\gamma^2)$ approximation of the distance $d(v, w)$. Because [18] considers weighted distances, the approach does not yield full-fleged, memory-efficient routing tables. The main obstacle preventing this is that in the weighted case, if a shortest path from v to a node w that is one of its k closest sources within d hops goes via $u \in \mathcal{N}_v$, this does not imply that w is among the k closest sources within d hops of u as well: Since edge weights are arbitrary, there could be a lot of sources that are exactly d and $d + 1$ hops away from u and v, respectively, but are among the sources closest to u in terms of weighted distance. This requires storing routing pointers for each iteration of the used variant of the Bellman-Ford algorithm, resulting in memory requirements of $\tilde{\Theta}(dk)$.

This is not true for the unweighted case: If u is the next hop on a shortest path from v to w, and $(d(u, w'), w') < (d(u, w), w)$ (i.e., w' is closer to u than w), then it follows that $(d(v, w'), w') \leq (d(u, w') + 1, w') < (d(u, w) + 1, w) = (d(v, w), w)$. Thus, if w is among the k sources closest to v and u is the next routing hop according to (the modified version of) Algorithm 1, then w is also among the k sources closest to v. Hence, storing routing pointers for the k first entries of the final lists L_v when calling (the modified version of) Algorithm 1 is sufficient for routing purposes.

Corollary 5.5. *For any integer $\gamma \in \{1, \dots, \log n\}$, in $\tilde{\mathcal{O}}(n^{1/2+1/(2\gamma)} + D)$ rounds it is possible to compute labels of size $\mathcal{O}(\gamma \log n)$ for each node and routing tables of size $\mathcal{O}(n^{1/\gamma} \log^{\mathcal{O}(1)} n)$ with respect to these labels at each node v that enable routing with stretch $\mathcal{O}(\gamma^2)$ w.h.p.*

Proof. We follow the strategy of Theorem 5.1 from [18], augmenting the distance sketches by the required routing information (cf. Corollary 4.20 in [18]). For routing, one needs to store the next routing hop for each node detected by a call to (the modified) Algorithm 1, which by the above reasoning requires up to $\mathcal{O}(n^{1/\gamma} \log^{\mathcal{O}(1)} n)$ memory bits for each of the $\gamma \leq \log n$ calls, and $\mathcal{O}(\gamma \log^{\mathcal{O}(1)} n)$ bits in order to permit routing from the nodes that are higher in the hierarchy to the nodes for which they serve as landmarks (see [18] for details; this component remains unchanged). □

2-Additive Spanners

A spanner for a given network is a (sparse) spanning subgraph enjoying certain useful properties. Spanner constructions are often based on building (complete or partial) BFS trees in the network. As a concrete example, consider additive spanners. An α-additive spanner H for the graph G is a spanning subgraph with the property that for every two nodes v and w the distance between v and w in H is at most α greater than their distance in G.

The well-known construction for 2-additive spanners [1, 9] is based on two main components. The first involves including in the spanner H all edges incident to nodes of degree at most $\sqrt{n}$. This is a local step requiring no communication. The second component requires selecting (say, uniformly at random) a set of nodes M in the graph, of size $|M| = \mathcal{O}(\sqrt{n}\log n)$, and constructing a (complete) BFS tree from every source $s \in M$. This part of the distributed construction dominates the total cost. A naive distributed implementation for this construction in the CONGEST model might cost $\mathcal{O}(D\sqrt{n}\log n)$ time in an n-node network of diameter D. Using the techniques of [15], the distributed time complexity of 2-additive spanner construction can be reduced to $\mathcal{O}(\sqrt{n}\log n + D)$. The same time bound (with a better leading constant) can be achieved using our efficient procedure for multiple source partial BFS tree construction.

Corollary 5.6. *For any graph G, an additive 2-spanner of G can be constructed in the CONGEST model within $\mathcal{O}(\sqrt{n}\log n + D)$ rounds w.h.p.*

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-AF-0937274, CNS-1035199, 0939370-CCF and CCF-1217506, the AFOSR under Contract No. AFOSR Award number FA9550-13-1-0042, the Swiss National Science Foundation (SNSF), the German Research Foundation (DFG, reference number Le 3107/1-1), the Israel Science Foundation (grant 894/09), the United States-Israel Binational Science Foundation (grant 2008348), the Israel Ministry of Science and Technology (infrastructures grant), and the Citi Foundation.

6. REFERENCES

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

[2] P. Almeida, C. Baquero, and A. Cunha. Fast distributed computation of distances in networks. In *Proc. 51st Conf. on Decision and Control (CDC)*, pages 5215–5220, 2012.

[3] J. Antonio, G. Huang, and W. Tsai. A fast distributed shortest path algorithm for a class of hierarchically clustered data networks. *IEEE Transactions on Computers*, 41:710–724, 1992.

[4] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *Proc. 47th Symp. on Foundations of Computer Science (FOCS)*, pages 591–602, 2006.

[5] S. Baswana, T. Kavitha, K. Mehlhorn, , and S. Pettie. Additive spanners and (α, β)-spanners. *ACM Transactions on Algorithms*, 7(1), 2010. Article No. 5.

[6] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms*, 2:557–577, 2006.

[7] S. Chechik. New additive spanners. In *Proc. 24th Symp. on Discrete Algorithms (SODA)*, 2013.

[8] S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Petricola. Partially dynamic algorithms for distributed shortest paths and their experimental evaluation. *Journal on Computers*, 2:16–26, 2007.

[9] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[10] M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1:283–323, 2005.

[11] M. Elkin and D. Peleg. $(1 + \epsilon, \beta)$-spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.

[12] S. Frischknecht, S. Holzer, and R. Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proc. 23rd Symp. on Discrete Algorithms (SODA)*, pages 1150–1162, 2007.

[13] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Distributed Computing*, 16:111–120, 2003.

[14] S. Haldar. An 'all pairs shortest paths' distributed algorithm using $2n^2$ messages. *Journal of Algorithms*, 24(1):20–36, 1997.

[15] S. Holzer and R. Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. 31st Symp. on the Principles of Distributed Computing (PODC)*, pages 355–364, 2012.

[16] S. Kanchi and D. Vineyard. An optimal distributed algorithm for all-pairs shortest-path. *International Journal Information Theories and Applications*, 11(2):141–146, 2004.

[17] T. Kavitha. Faster algorithms for all-pairs small stretch distances in weighted graphs. In *Proc. 27th Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 328–339, 2007.

[18] B. Patt-Shamir and C. Lenzen. Fast Routing Table Construction Using Small Messages [Extended Abstract]. In *Proc. 45th Symposium on the Theory of Computing (STOC)*, 2013. Full version at http://arxiv.org/abs/1210.5774.

[19] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[20] D. Peleg, L. Roditty, and E. Tal. Distributed algorithms for network diameter and girth. In *Proc. 39th Colloquium on Automata, Languages, and Programming (ICALP)*, pages 660–672, 2012.

[21] L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proc. 32nd Colloquium on Automata, Languages, and Programming (ICALP)*, pages 261–272, 2005.

[22] L. Roditty and V. Williams. Approximating the diameter of a graph. *CoRR*, abs/1207.3622, 2012.

[23] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, 29:23–35, 1983.

[24] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52:1–24, 2005.

[25] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. 17th Symp. on Discrete Algorithms (SODA)*, pages 802–809, 2006.