Space-Efficient Path-Reporting Approximate Distance Oracles

Michael Elkin* Ofer Neiman† Christian Wulff-Nilsen‡

February 8, 2016

Abstract

We consider approximate *path-reporting* distance oracles, distance labeling and labeled routing with extremely low space requirements, for general undirected graphs. For distance oracles, we show how to break the $n \log n$ space bound of Thorup and Zwick if approximate *paths* rather than distances need to be reported. For approximate distance labeling and labeled routing, we break the previously best known space bound of $O(\log n)$ words per vertex. The cost for such space efficiency is an increased stretch.

1 Introduction

1.1 Distance oracles

Given a graph $G = (V, E)$ with edge weights, an *approximate distance oracle* for G is a data structure that can report approximate distance queries between vertex pairs efficiently. For any vertices $u, v \in V$, if $d_G(u, v)$ denotes the shortest path distance from u to v in G and if $\tilde{d}(u, v)$ denotes the approximate distance output by the oracle, we require that $d_G(u, v) \leq \tilde{d}(u, v) \leq \delta d_G(u, v)$, where $\delta \geq 1$ is the approximation (called also the (multiplicative) *stretch* parameter) of the oracle. The goal is to give an approximate distance oracle with small space, query time, stretch, and (perhaps to a lesser extent) preprocessing time.

Our focus is on undirected graphs as it can be shown that no non-trivial oracles exist for directed graphs [TZ05]. A seminal result in this area is that of Thorup and Zwick [TZ05]. For any positive integer k and a graph with non-negative edge weights and with m edges and n vertices, they gave an approximate distance oracle with space $O(kn^{1+1/k})$, stretch $2k - 1$, query time $O(k)$, and preprocessing time $O(kmn^{1/k})$. For constant k, the trade-off between the first three parameters is optimal, assuming a widely believed and partially proved girth conjecture of Erdős [Erd63]. For super-constant k, small improvements exist. In [WN13], it was shown how to improve the query time to $O(\log k)$ while keeping the same space, stretch, and preprocessing. More recently, Chechik [Che14] further improved this to $O(1)$ query time. Mendel and Naor [MN07] gave an oracle with $O(n^{1+1/k})$ space and $O(1)$ query time at the cost of a constant-factor increase in stretch.

So far, we have only discussed queries for approximate *distances* but it is natural to require the data structure to also be able to report corresponding *paths*. We say that an oracle is *path-reporting* if it can

*Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel. Email: elkinm@cs.bgu.ac.il.

†Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel. Email: neimano@cs.bgu.ac.il. Supported in part by ISF grant No. (523/12) and by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°303809.

‡Department of Computer Science, University of Copenhagen. Email: koolooz@di.ku.dk, http://www.diku.dk/~koolooz/.

report those paths in time proportional to their lengths (in addition to the query time needed for distances), and we say that it is a not path-reporting oracle otherwise. The oracles of [TZ05, WN13, Che14] are path-reporting, but this is not the case for the oracle of Mendel and Naor [MN07]. Note that a space requirement of order $kn^{1+1/k}$ is $\Omega(n \log n)$ for any choice of k. In this paper, we focus on path-reporting distance oracles that use $o(n \log n)$ space, albeit at the price of increased stretch.

1.2 Distance Labeling

In a *labeling scheme* the goal is to assign as short labels as possible to each vertex of the input graph so that a query for any pair (u, v) of vertices can be answered (preferably efficiently) exclusively from the labels assigned to u and v. We are interested in a *distance labeling* scheme where given labels of two vertices u and v, a distance estimate $\tilde{d}(u, v)$ that satisfies $d_G(u, v) \leq \tilde{d}(u, v) \leq \delta \cdot d_G(u, v)$ can be efficiently computed.

Distance labeling was introduced in a pioneering work by Peleg [Pel00b]. The distance oracles of Thorup and Zwick [TZ05] and their refinements [WN13, Che14] can serve as distance labeling schemes as well. (The maximum label size becomes $O(n^{1/k} \cdot \log^{1-1/k} n)$ words, and other parameters stay intact.) Mendel-Naor's oracle [MN07] can also be viewed as a distance labeling scheme, but its label size is $O(k \cdot n^{1/k})$ (i.e., $\Omega(\log n)$ space per label as well).

To summarize, *all* existing distance labeling schemes use $\Omega(\log n)$ words per label in the worst case. The labeling scheme that we devise in the current paper uses $o(\log n)$ words per label, for graphs with polynomially bounded diameter. On the other hand, its stretch guarantee is much larger than that of [TZ05, WN13, Che14].

1.3 Labeled Routing

In a closely related *labeled routing* problem we want to precompute two pieces of information for every vertex u of the input graph. These are the *label* of u and the *routing table* of u. Given a label of another vertex v, the vertex u should decide to which neighbor w of u to forward a message intended for v based on its local routing table and on the label of v. Given this forwarded message with the label of v, the neighbor w selects one of its own neighbors, and forwards it the message, and so forth. The *routing path* is the u-v-path which will eventually be taken by a message originated in u and intended for v. (Assuming that the routing scheme is correct, the path will indeed end in v.) The *stretch* of a routing scheme is the maximum ratio between a length of a routing u-v path and the distance $d_G(u, v)$ between u and v, taken over all (ordered) pairs (u, v) of vertices.

Labeled routing problem was introduced in a seminal paper by Peleg and Upfal [PU88], and it was studied in [Cow99, EGP98, AP92, ANLP90]. A labeled routing scheme was devised by Thorup and Zwick [TZ01]. It provides stretch $4k - 5$ and uses routing tables of size $O(polylog(n) \cdot n^{1/k})$ and labels of size $O(k \cdot \frac{\log^2 n}{\log \log n})$. The stretch was recently improved by Chechik [Che13] to $(4 - \epsilon)k$ for some $\epsilon > 0$.

The space usage by current routing schemes is at least logarithmic in n (counted in words; each word is $O(\log n)$ bits). In many settings such space requirement is prohibitively large. In this paper we show a labeled routing scheme in which the space requirement per vertex (both labels and routing tables) can be as small as one wishes, for graphs with diameter at most some polynomial in n. On the other hand, similarly to the situation with distance labeling schemes, the stretch guarantee of our scheme is much larger than that of [TZ01].

1.4 Our Results

We introduce two new data structures that report paths in undirected graphs. All have query time proportional to the length of the returned path. The first applies to weighted graphs with diameter polynomially bounded in n. For any $t \geq 1$, it reports paths of stretch $O(\sqrt{t}n^{2/\sqrt{t}})$ using space $O(tn)$. It may be distributed as a labeling scheme using $O(t)$ space per vertex (or $O(t \log n)$ bits), and the preprocessing time is $O(tm)$. See Theorem 2 for the formal statement. [1] This data structure can also be modified to provide labeled routing. Specifically, using tables of size $O(t)$ and labels of size $O(\sqrt{t})$ our routing scheme provides stretch $O(\sqrt{t} \cdot n^{2/\sqrt{t}} \cdot \log n)$.

The second data structure is a distance oracle that applies only to unweighted graphs. In one of the possible settings, it can provide for any parameters $k \geq 1$ and $\epsilon > 0$, a path-reporting distance oracle with stretch $O(kn^{1/k} \cdot (k + n^{\epsilon/k}))$, using space $O(kn/\epsilon)$ and preprocessing time $O(kmn^{1/k})$. See Theorem 6, and also Theorem 5 for more possible tradeoffs.

To our knowledge, our distance labeling and labeled routing schemes are the first that use $o(\log n)$ words per vertex. Our distance oracles are the first path-reporting oracles for general graphs that use space $o(n \log n)$.

We show that our techniques are useful also for the opposite part of the stretch-space tradeoff curve, in the setting of graphs excluding a minor. Previous labeling schemes for K_r-free graphs [AG06, KKS11], obtained arbitrarily low stretch using small space, but the query time is at least $\Omega(f(r) \cdot \log n)$, where $f(r)$ is an extremely fast growing function. We devise constant stretch distance oracles and labeling schemes, which have very fast query time (independent of the excluded minor). The label size in our schemes is polylogarithmic in n.

1.5 Overview of Techniques.

Our first oracle is based on a collection of sparse covers. Roughly speaking, a sparse cover for radius ρ has two parameters: β is the *radius blow-up*, and s the *overlap*. The cover is a collection of clusters, each of diameter at most $\beta\rho$, such that every ball of radius ρ is fully contained in at least one cluster, and every vertex is contained in at most s clusters (see Definition 1 below for a formal definition). Sparse covers were introduced by [AP90b], and found numerous applications in distributed algorithms and routing (see, e.g. [PU88, AP90a, Pel93, AP95, AGM+08]). For the application to distance oracles and labeling schemes, the radius blow-up corresponds to stretch and the overlap to space. The standard construction of [AP90b] for parameter $k \geq 1$ has radius blow-up k and overlap $O(kn^{1/k})$. This overlap is at least $\Omega(\log n)$, and translates to such space usage per vertex. Here we show that one can obtain the inverse parameters: radius blow-up $O(kn^{1/k})$ with overlap $2k$ (in fact we can obtain overlap $(1 + \epsilon)k$ for any fixed $\epsilon > 0$).

Our first construction of a distance labeling scheme is very simple: it uses a collection of such sparse covers for all distance scales, and maintains a shortest-path tree for each cluster. In order to answer a path query, one needs to find an appropriate cluster in the right scale, and return a path from the corresponding tree.

Our second data structure combines sparse covers with a variation on the Thorup-Zwick (TZ) distance oracle. In order to save space, the "bunches" of the TZ oracle are kept only for a small set of carefully selected vertices. Furthermore, the TZ trees (from which the path is obtained) are pruned to contain only few important vertices. Given a path query, our pruned TZ oracle can only report a "skeleton" of the approximate shortest path in the original graph. This skeleton contains few vertices (roughly one vertex per

[1] For arbitrary diameter Λ, the space and preprocessing time increase by a factor of $O(\log_n \Lambda)$.

p steps, for some parameter p). We then use a sparse cover to "fill in" the gaps in the path, which induces additional stretch.

Our results for minor-free graphs are based on a novel construction of sparse covers, whose parameters are incomparable to previous works [AGMW10, BLT07]. Our covers are built using the recently developed padded decompositions of [AGG⁺14]

1.6 Related Work

There has been a large body of work on distance oracles, labeling and routing for certain graph families (planar, excluded-minor, etc.) and bounded doubling dimension metrics [Tho04, HPM06, AG06, KKS11]. In these settings the stretch factor is usually $1+\epsilon$, which cannot be obtained with $o(n^2)$ space for general graphs.

For sparse graphs, very compact distance oracles were recently devised by Agarwal et al. [AGHP11, AGHP12]. They devise two types of distance oracles. One of them has small stretch but requires large space. This distance oracle is indeed path-reporting, but due to their large space requirement they are irrelevant to the current discussion. The other type of distance oracles in [AGHP11] has stretch at least 3. These latter distance oracles are very sparse, but they are not path-reporting. [2]

Following our work, [EP15] devised a distance oracle with stretch $O(\text{polylog}\, n)$, space $O(n\log\log n)$ and query time $O(\log\log n)$ (this oracle does not give rise to a labeling scheme nor to a routing scheme).

1.7 Organization of the Paper

After some basic definitions in Section 2, we introduce sparse covers with small overlap in Section 3. Our first data structure for weighted graphs with diameter polynomially bounded in n, is presented in Section 4, and its adaptation for compact routing in Section 5. The second data structure with improved parameters for unweighted graphs is given in Section 6. Finally, in Section 7 we discuss improved results for graphs excluding a minor.

2 Preliminaries

Let $G=(V,E)$ be an undirected weighted graph, with the usual shortest path metric d_G. We always assume the minimal distance in G is 1. For a subset $U\subseteq V$ let $G[U]$ denote the induced graph on U. For a parameter $\rho>0$, and two sets of balls $\mathcal{B},\mathcal{S}\subseteq\{B(v,\rho)\mid v\in V\}$, define $\partial_{\mathcal{B}}(\mathcal{S})=\{B\in\mathcal{B}\mid \exists S\in\mathcal{S},\ B\cap S\neq\emptyset\}$ to be the subset of balls from $\mathcal{B}$ that intersect with a ball from $\mathcal{S}$.

Definition 1. *A collection of clusters $\mathcal{C}=\{C_1,\dots,C_t\}$ is called a strong diameter (β,s,ρ)-*sparse cover *if*

- Radius blow-up: $\text{diam}(G[C_i])\le\beta\rho$ *for all* $i\in[t]$.
- Padding: *For each $v\in V$, there exists $i\in[t]$ such that $B(v,\rho)\subseteq C_i$.*
- Overlap: *For each $v\in V$, there are at most s clusters in $\mathcal{C}$ that contain v.*

For a vertex v and a cluster C_i such that $B(v,\rho)\subseteq C_i$, we say that the vertex v is padded *by the cluster C_i.*

[2] The paper erroneously claims that they are [AGHP14].

3 Sparse Covers with Small Overlap

In this section we show how to construct a sparse cover with arbitrarily low overlap. Our construction essentially inverts the parameters in the classical tradeoff of [AP90b], which has low radius blow-up. We use a region growing technique on the set of balls of radius ρ.

Algorithm 1 `Sparse-Cover`(G, ρ, k)

1: $\mathcal{C} = \emptyset$.
2: $\mathcal{U} = \{B(v, \rho) \mid v \in V(G)\}$.
3: **while** $\mathcal{U} \neq \emptyset$ **do**
4:     $\mathcal{R} = \mathcal{U}$.
5:     **while** $\mathcal{R} \neq \emptyset$ **do**
6:         Let $B \in \mathcal{R}$.
7:         Let $\mathcal{S} = \{B\}$.
8:         **while** $|\partial_{\mathcal{R}}(\mathcal{S})| \geq |\mathcal{S}| \cdot \left(1 + \frac{\log n}{k \cdot n^{1/k}}\right)$ **do**
9:             $\mathcal{S} \leftarrow \partial_{\mathcal{R}}(\mathcal{S})$.
10:         **end while**
11:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{\bigcup_{B' \in \mathcal{S}} B'\}$.
12:         /* A new cluster $C = \bigcup_{B' \in \mathcal{S}} B'$ is added to $\mathcal{C}$. */
13:         $\mathcal{R} \leftarrow \mathcal{R} \setminus \partial_{\mathcal{R}}(\mathcal{S})$.
14:         $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathcal{S}$.
15:     **end while**
16: **end while**

Theorem 1. *For any weighted graph G on n vertices, any $\rho > 0$ and $k \geq 1$, there exists a strong diameter $(8k \cdot n^{1/k}, 2k, \rho)$-sparse cover.*

Proof. Consider Algorithm 1 for creating a sparse cover. Observe that we only throw a ball from $\mathcal{U}$ when it is contained in $\mathcal{S}$ and will surely be contained in a cluster. Thus when the algorithm terminates all ρ-balls are padded.

Let n_i denote the number of balls in $\mathcal{U}$ at the end of the i-th iteration of the outer loop. Then $n_0 = n$, and by the termination condition of the while loop on line 8,

$$n_{i+1} < n_i \cdot \left(\frac{\log n}{k \cdot n^{1/k}}\right) .$$

This implies that

$$n_{2k} < n \cdot \left(\frac{\log n}{k \cdot n^{1/k}}\right)^{2k} = \frac{1}{n} \cdot \left(\frac{\log n}{k}\right)^{2k} \leq 1 ,$$

where the last inequality holds because the function $((\log n)/k)^{2k}$ is maximal when $k = (\log n)/2$, in which case it is n. We conclude that the algorithm terminates after at most $2k$ phases. When forming a cluster $C = \bigcup_{B \in \mathcal{S}} B$, all balls in $\partial_{\mathcal{R}}(\mathcal{S})$ (the balls that intersect C) are removed from $\mathcal{R}$ and thus will not be considered in the current phase (the loop starting at line 5), which implies that every point $v \in V(G)$ can belong to at most a single cluster per phase. So the total overlap is at most $2k$.

It remains to bound the strong diameter of any cluster. A cluster $\mathcal{S}$ starts as a ball of diameter at most 2ρ, and in each iteration of line 8 its size (number of balls it contains) increases by a factor of at least $\left(1 + \frac{\log n}{k \cdot n^{1/k}}\right)$. After $2k \cdot n^{1/k}$ iterations its size will be at least

$$\left(1 + \frac{\log n}{k \cdot n^{1/k}}\right)^{2k \cdot n^{1/k}} > e^{\log n} > n \ . \tag{1}$$

For the last inequality we used that $1+x > e^{x/2}$ when $0 < x \leq 1$ (indeed $(\log n)/(k \cdot n^{1/k}) \leq 1$). Inequality (1) is a contradiction, so the number of iterations in line 8 is less than $2k \cdot n^{1/k}$. In each such iteration the diameter can increase by at most 4ρ, so the total diameter is bounded by $8k \cdot n^{1/k} \cdot \rho$. □

Remark: A similar algorithm and calculation shows that for any $1/k \leq \epsilon \leq 1$, one can obtain also a $(8k \cdot n^{1/k}/\epsilon, (1+\epsilon)k, \rho)$-sparse cover, though we shall not require this generalization here.

3.1 Fast Construction of Sparse Covers

In this section we show a fast construction of sparse covers, with slightly worse constants.

For a weighted graph $G = (V, E)$, we show a probabilistic construction of $(64k \cdot n^{1/k}, 2k, \rho)$-sparse cover in time $O(k \cdot |E|)$, for any $\rho > 0$. The main building block are padded partitions. A *partition* $P = \{C_1, \ldots, C_t\}$ of the graph G is a collection of pairwise disjoint clusters whose union covers V. We say that the partition is strong diameter Λ-bounded if $\mathrm{diam}(G[C_i]) \leq \Lambda$. For a partition P and a vertex x, let $P(x)$ denote the cluster of P that contains x. We use the following Lemma that appears (implicitly) in [Bar96].

Lemma 1. *For any weighted graph on n vertices there exists a distribution $\mathcal{P}$ over strong diameter Λ-bounded partitions, such that for all $v \in V$ and $0 \leq \beta \leq 1/8$,*

$$\Pr_{P \sim \mathcal{P}}[B(x, \beta\Lambda) \subseteq P(x)] \geq n^{-16\beta} \ .$$

Furthermore, one can sample from this distribution in linear time.

In order to construct a cover, just sample a partition according to the distribution of Lemma 1 for $2k$ times, with parameters $\Lambda = 64k \cdot n^{1/k} \cdot \rho$ and $\beta = \rho/\Lambda$, and return the collection of clusters obtained. The radius bound on each cluster is Λ, and since each partition consists of disjoint clusters, each point will be covered exactly $2k$ times. The probability that a certain ball of radius $\rho = \beta\Lambda$ is not contained in any of the $2k$ partitions is at most

$$\left(1 - n^{-16\beta}\right)^{2k} = \left(1 - e^{-16 \ln n/(64kn^{1/k})}\right)^{2k} \leq \left(\frac{\log n}{2kn^{1/k}}\right)^{2k} < 1/n^{3/2} \ ,$$

which holds since $(\log n/(2k))^{2k} \leq n^{1/2}$. Using a union bound over all n balls, there is high probability that each of them will be contained in some cluster. As each partition is created in linear time, the total running time is $O(|E| \cdot k)$.

4 Small Space Distance Labeling Scheme

In this section we provide a distance labeling scheme, that can also serve as a path-reporting distance oracle, which is built from a collection of sparse covers. Its parameters are somewhat inferior to the parameters of the distance oracle from Section 6. On the other hand, the latter construction applies only to unweighted graphs, while the construction is this section applies to weighted graphs. Also, it is not clear to us if the construction of Section 6 can be converted into a distance labeling scheme.

Theorem 2. *For any weighted graph $G = (V, E)$ on n vertices with diameter Λ, and any $t \geq 1$, there exists a distance labeling scheme with stretch $O(\sqrt{t} \cdot n^{2/\sqrt{t}})$ using $O(t \cdot \log_n \Lambda)$ space (or $O(t \cdot \log \Lambda)$ bits) per vertex, that can be constructed in $O(t|E| \cdot \log_n \Lambda)$ time. Furthermore, this data structure can also serve as a path-reporting distance oracle, whose query time is proportional to the length of the returned path, plus $O(\log(t \cdot \log_n \Lambda))$.*

Remark: Observe that when the diameter Λ is at most polynomial in n, the required space is $O(t)$ words per vertex.

Proof. Fix a parameter $1 \leq k$, and let $\Lambda = \mathrm{diam}(G)$, $\gamma = 8kn^{1/k}$, $q = \lceil \log_{n^{1/k}} \Lambda \rceil = \lceil k \log_n \Lambda \rceil$. For each $i \in \{0, 1, \ldots, q\}$ create a $(\gamma, 2k, n^{i/k})$-sparse cover $\mathcal{C}_i$. For each cluster $C \in \mathcal{C}_i$ choose an arbitrary shortest path tree (SPT) spanning $G[C]$. Every vertex stores a hash table containing the names of the SPTs it is contained in, and for each such tree the vertex only needs to store a pointer to its parent in the tree and the distance to the root of the tree. Since every vertex is contained in at most $2k$ clusters per level, the total space used is $O(k \cdot q)$ per vertex. Observe that if $\Lambda = \mathrm{poly}(n)$ then $q = O(k)$. In addition, for every vertex $u \in V$ and $i \in [q]$, store a pointer to the SPT of a cluster $C_i(u) \in \mathcal{C}_i$ such that $B(u, n^{i/k}) \subseteq C_i(u)$.

Next we describe an algorithm for answering a path query between $u, v \in V$. Let $i \in [q]$ be such that $n^{(i-1)/k} \leq d_G(u, v) < n^{i/k}$. Let $J = \{j \in [0, q] : v \in C_j(u)\}$ be the set of indices j such that $v \in C_j(u)$. By the padding property of the sparse cover, $v \in B(u, n^{j/k}) \subseteq C_j(u)$, for every $j \geq i$. Hence every index $j \geq i$ belongs to J. We will conduct a binary search on $[0, q]$ to find an index j such that $j \in J$ and $j - 1 \notin J$. (Alternatively, we will discover that $0 \in J$.) By the above considerations the index j that we will find satisfies $j \leq i$. As u holds a pointer to $C_j(u)$ for every $j \in \{0, 1, \ldots, q\}$ and v stores the names of clusters containing it in a hash table, deciding if $j \in J$ requires $O(1)$ time. Next, both u, v follow the path to the root in the SPT created for $C_j(u)$. By taking a step towards the root in the path with the longer remaining distance, we can guarantee that the paths will meet at the least common ancestor of u, v.

The query time is bounded by length of the returned path, which is $O(\mathrm{diam}(C_i(u)) = O(\gamma n^{i/k}) = O(\gamma n^{1/k} \cdot d_G(u, v))$, so the stretch is $O(\gamma n^{1/k}) = O(kn^{2/k})$. In addition we spend $O(\log q) = O(\log(k \log_n \Lambda))$ time for the binary search. If one is willing to settle for $\gamma = 64kn^{1/k}$ (rather than $8kn^{1/k}$), then the preprocessing expected time is $O(qk \cdot |E|)$, using the construction of Section 3.1.

Finally, note that for the labeling scheme, we can find the appropriate $j \in J$ and return the sum of distances from u, v to the root of the SPT of $C_j(u)$, just by inspecting the labels of u, v.

□

5 Routing

We consider a compact routing framework, in which every vertex in the graph has a short label (word size), and stores a routing table. Given a vertex u and a label of v, using the routing tables starting at u and given only the label of v, we should route from u to v quickly. Specifically, we show the following result.

Theorem 3. *Fix any parameter k. Any weighted graph $G = (V, E)$ on n vertices with diameter Λ admits a compact routing scheme, in which the labels are of size $O(k \log_n \Lambda)$ and the routing tables are of size $O(k^2 \log_n \Lambda)$. For any two vertices $u, v \in V$, the scheme produces routing paths from u to v of length at most $O(kn^{2/k} \log n \cdot d_G(u, v))$.*

We shall use Interval Tree Routing described in [Pel00a], lemma 26.1.2.

Theorem 4. *Let T be a tree on n vertices with depth d, then there exists a compact routing scheme that uses a single word ($O(\log n)$ bits) as a label and produces paths of length $O(d \log n)$.*

Using the same framework as Section 4, one can extend the distance labels described there to a compact routing scheme with almost the same parameters: we only lose a factor of $O(\log n)$ in the routing time. Each vertex u will have a *routing table* of size $O(k \cdot q)$. Specifically, for each level $i \in [q]$ and each SPT containing u in this level, the vertex will store the relevant information required for interval-tree routing (see Theorem 4). The *label* of u will be much shorter, of size $O(q)$: for every $i \in [q]$ store the information only for the SPT created from $C_i(u)$, i.e., the cluster in which u is padded. In order to route from u to v, we find an index i such that $v \in C_i(u)$ but $v \notin C_{i-1}(u)$ (this can be done since we have all the information for u) and route in the corresponding SPT using interval-tree routing (Theorem 4). Recall that $v \notin C_{i-1}(u)$ implies that $d_G(u, v) \geq n^{(i-1)/k}$.

The depth of the SPT is bounded by the diameter of the cluster $C_i(v)$, and $\mathrm{diam}(C_i(v)) \leq \gamma \cdot n^{i/k} = O(\gamma n^{1/k} \cdot d_G(u, v))$. (Recall that $\gamma = 8kn^{1/k}$.) So the the length of the routing path in the tree is $O(\gamma n^{1/k} \log n \cdot d_G(u, v)) = O(kn^{2/k} \log n \cdot d_G(u, v))$.

6 Small Space Path-Reporting Distance Oracles

In this section we show a path-reporting distance oracle with improved stretch, at the price of being applicable only for unweighted graphs. Also we do not know if it is possible to distribute the information among vertices, i.e., to convert this oracle into a labeling scheme. The distance oracle in this section has both additive and multiplicative stretch. For $\alpha \geq 1$ and $\beta \geq 0$, we say that a distance estimate $\tilde{d}$ has (α, β)-*stretch* if for all $u, v \in V$, $d_G(u, v) \leq \tilde{d}(u, v) \leq \alpha \cdot d_G(u, v) + \beta$.

Theorem 5. *For any unweighted graph $G = (V, E)$ on n vertices, any integers $k, p, t \geq 1$, there exists a path-reporting distance oracle with $\big(O(t \cdot kn^{1/k}), O(p \cdot kn^{1/k})\big)$-stretch, using $O(kn + tn^{1+1/t}/p)$ space. Furthermore, the query time is proportional to the length of the returned path. The oracle can be constructed in $O(tmn^{1/t})$ time.*

Proof Overview: Fix a parameter p, we partition the distances to those smaller than p and those larger. In order to be space efficient, we "prune out" most of the vertices in the distance oracle of Thorup-Zwick. We will choose a subset $N \subseteq V$, of size n/p, that touches the (approximately) p-neighborhood of any vertex of V. The TZ-oracle will be responsible for the large distances between any two vertices in N: it should be able to report a sufficiently dense "skeleton" of an approximate shortest path. All consecutive distances on the path are roughly p. We show that one can significantly reduce the size of each of the TZ trees, while still maintaining this usability. We augment our data structure with a sparse cover that will handle all the small distances: specifically we need to "fill in" the paths between consecutive vertices in the skeleton, and the paths between each vertex to its representative in N.

6.1 Construction

We shall use the following Lemmata. The first one is folklore.

Lemma 2. *For every unweighted graph on n vertices and parameter r, there is a set of at most $2n/r$ vertices that intersects every ball of radius r.*

The next lemma can be found, e.g., in [NS07], Lemma 12.1.5.

Lemma 3. *For every tree T on n vertices and parameter r, there is a set of at most $2n/r$ vertices whose removal separates T into components of size at most r each.*

One of the building blocks of our oracle is a variation of the Thorup-Zwick oracle [TZ05]. We briefly recall the TZ construction with stretch parameter t: Define $A_0 = V$ and for each $i \geq 1$ sample A_i from A_{i-1} by including every element of A_{i-1} independently with probability $n^{-1/t}$. Finally, set $A_t = \emptyset$. For $u \in V$ define the *bunch* of u as $B(u) = \{w \in A_{i-1} \mid d_G(u, w) < d_G(u, A_i), i \geq 1\}$. For each $w \in V$, if i is such that $w \in A_{i-1} \setminus A_i$, define $C(w) = \{u \in V \mid d_G(w, u) < d_G(u, A_i)\}$. (Note that the cluster $C(w)$ contains all vertices u such that $w \in B(u)$). It can be shown that for every $u \in C(w)$, all vertices on the shortest path between u and w also belong to $C(w)$ as well. As a result, an SPT for $C(w)$ is a subtree of an SPT rooted at w for the entire graph G. During the preprocessing such a tree spanning $C(w)$ is created for each $w \in V$. We denote it by T_w.

In the original data structure, each vertex $u \in V$ stored the vertices in $B(u)$ and their distances from u in G. For each $i = 0, \ldots, t-1$, it also stored the special vertex $p_i(u) \in A_i$, which is the closest vertex to u in A_i. The query algorithm on u, v uses only the information stored by the query vertices to produce some $w \in B(u) \cap B(v)$ such that $d_G(u, v) \leq d_G(u, w) + d_G(w, v) \leq (2k-1)d_G(u, v)$, and the actual path could be obtained from T_w. It is also shown in [TZ05] that for each $v \in V$, the expected size of $B(v)$ is $O(kn^{1/k})$.

As we aim to save space, we will only store the bunches $B(u)$ for a few vertices. Fix a parameter p, and let N be a set of size n/p that hits every ball of radius $2p$. (See Lemma 2, $r = 2p$.) Only the vertices $v \in N$ will store the bunches $B(v)$ and special vertices $p_i(v)$ of the TZ-oracle. Since the total size of the trees $\{T_w\}_{w \in V}$ is equal to the total size of the bunches, we also need to prune these trees. For each $w \in V$, let R_w be the set given by Lemma 3 applied on the tree T_w with $r = p$, of size at most $2|T_w|/p$, and let $R = \cup_{w \in V} R_w$. Let $\bar{T}_w$ be the pruned tree that contains only the vertices of T_w that are in $R_w \cup N \cup \{w\}$. Specifically, each vertex in the pruned tree $\bar{T}_w$ will store a pointer to its nearest ancestor which is also in $\bar{T}_w$, the distance to it, and the distance to the root.

We shall also require a sparse cover (as constructed in Section 3). For a parameter $k \geq 1$, let $\mathcal{C}$ be a $(8kn^{1/k}, 2k, 3p)$-sparse cover, and for each cluster $D \in \mathcal{C}$ create an SPT spanning $G[D]$. As before, in each tree a vertex stores a pointer to its parent and the distance to the root. Additionally, every vertex $u \in V$ stores a hash table of trees containing it, a pointer to $D(u)$, a cluster in which it is padded, and a pointer to some $u' \in N$ such that $d_G(u, u') \leq 2p$. Note that $u' \in D(u)$.

Remark: Observe that we build the data structure on all of V and then prune the obtained TZ trees, rather than applying the TZ structure restricted to the vertices of N (which would seem an obvious simplification). This is because the TZ trees that will be produced from the metric induced on N may have arbitrarily large weights on the edges. One then would need a different mechanism for replacing these edges by paths of the original graph. This is because the cover $\mathcal{C}$ can only help filling in gaps of length up to $3p$.

Bounding the Size of the Oracle: Next we show that the space used by our oracle is $O(kn + tn^{1+1/t}/p)$. To see this, note that since the cover overlap is $2k$, to store the SPTs of the cover and the relevant pointers for each $u \in V$ requires only $O(kn)$ space. Next, we bound the size of the stored bunches. The number of vertices in N is at most n/p, and since the expected bunch size for each vertex is $O(tn^{1/t})$, the total (expected) size of the bunches we store is $O(tn^{1+1/t}/p)$. It remains to bound the size of the pruned trees. Since every vertex $u \in N$ is expected to appear in $O(tn^{1/t})$ trees (the number of trees equals its bunch size), the contribution of vertices in N to the size of the trees $\{\bar{T}_w\}_{w \in V}$ is again $O(tn^{1+1/t}/p)$. Finally, recall that the (expected) size of all the trees $\{T_w\}_{w \in V}$ is $O(tn^{1+1/t})$. Lemma 3 implies that in each tree only fraction of $2/p$ of the vertices are in R (rounded up), thus the contribution of vertices in R to the pruned trees is $O(n + tn^{1+1/t}/p)$. The roots of the pruned trees contribute only $O(n)$ to the size.

Construction Time: The bottleneck in our construction time is to find the clusters $C(w)$. With the construction of Thorup and Zwick, we get a bound of $O(tmn^{1/t})$.

6.2 Answering Path Queries

In order to answer a path query on $u, v \in V$, we first check if $v \in D(u)$. If so, taking the paths to the root in the SPT created from $D(u)$ from both u and v, as done in Section 4, will give a path of length $O(\mathrm{diam}(D(u)) = O(kn^{1/k} \cdot p)$, which induces such additive stretch.

Note that if $d_G(u, v) \le 2p$ it must be that $v \in D(u)$, so the complementary case is when $d_G(u, v) > 2p$. We shall use the pruned TZ data structure in the following way. First use the pointers stored at u, v to find $u' \in D(u) \cap N$ and $v' \in D(v) \cap N$. Using the TZ algorithm, which only requires the information stored by u' and v', we find $w \in B(u') \cap B(v')$ with $d_G(u', w) + d_G(v', w) \le (2t-1) \cdot d_G(u', v')$. Since w is contained in both bunches, and $u', v' \in N$, we get that $u', v' \in \bar{T}_w$. Since $\bar{T}_w$ is also a shortest path tree from w,

$$d_{\bar{T}_w}(u', v') \le d_G(u', w) + d_G(v', w) \le (2t-1) d_G(u', v') \; .$$

The "skeleton path" $u' = u_0, u_1, \ldots u_{l'} = v'$ induced by the (pruned) tree $\bar{T}_w$ from u' to v' has stretch $2t-1$. It can be obtained efficiently by following paths towards the root w from u', v', as done above. Our goal now is to show that there is a subpath, in which all consecutive distances are in the range $[p, 3p]$, these "gaps" will be covered by the sparse cover. Since removing R_w partitions T_w into subtrees of size at most p, it cannot be the case that there is a path in T_w of length p that does not intersect R_w (such a path induced a subtree with $p+1$ vertices). We conclude that for each $j \in [l']$, $d_G(u_{j-1}, u_j) \le p$. We further prune this skeleton path, to get a sub-path in which all consecutive distances are in the range $[p, 3p]$. This can be achieved by greedily deleting excessive points (those closer than p to the last point we kept) while traversing the path, and making sure to keep both u', v'. It is not hard to verify that the maximum distance between consecutive points will be at most $3p$. Let $u' = v_0, v_1, \ldots, v_l = v'$ be the resulting skeleton path.

For each $j \in [l]$ find a path in G from v_{j-1} to v_j using the sparse cover. Since $d_G(v_{j-1}, v_j) \le 3p$ we get that $v_j \in D(v_{j-1})$. So we can obtain a path in G from v_{j-1} to v_j of length at most $O(p \cdot kn^{1/k})$, in the same manner we handled the base case above (where $v \in D(u)$). Note that this induces a $O(kn^{1/k})$ stretch for each j (because $d_G(v_{j-1}, v_j) \ge p$), so the final multiplicative stretch is $O(t \cdot kn^{1/k})$. In a similar manner, since both $d_G(u, u'), d_G(v, v') \le 2p$, we obtain from the sparse cover paths from u to u' and from v' to v of distance at most $O(p \cdot kn^{1/k})$. The latter contributes to the additive stretch.

The running time of the query is proportional to the length of the path returned, since after finding the tree $\bar{T}_w$, we just follow pointers to the roots in both the pruned TZ-trees and in the SPT of the cover, in constant time per step. Note that the $O(t)$ time to find T_w is dominated by the stretch factor which we can assume is bounded by the length of the path. This proves Theorem 5.

6.3 Improved Multiplicative Stretch using Several Covers

Choosing $t = k$ and $p = n^{1/k}$ in the parameters of Theorem 5 yields stretch of $(O(k^2 n^{1/k}), O(kn^{2/k}))$. In terms of purely multiplicative stretch it is $O(kn^{1/k} \cdot (k + n^{1/k}))$. Next we show how to improve one of the factor of $n^{1/k}$ at the cost of increased space. Instead of a single cover, we use a collection of s sparse covers, and obtain the following theorem.

Theorem 6. *For any unweighted n-vertex graph $G = (V, E)$, any positive integer parameter k, and any parameter $\epsilon > 0$, there exists a path-reporting distance oracle with space $O(kn/\epsilon)$ and stretch $O(kn^{1/k} \cdot (k + n^{\epsilon/k}))$. Furthermore, the query time is proportional to the length of the returned path.*

Proof. For each $i \in [s]$ let $\mathcal{C}_i$ be a $(8kn^{1/k}, 2k, (3p)^{i/s})$-sparse cover, and for each cover store the same information per vertex as above. For $i \in [s]$, denote by $D_i(u)$ the cluster in $\mathcal{C}_i$ in which u is padded. Recall that the (additive) factor of $O(p \cdot kn^{1/k})$ in the stretch was inflicted in the base case when $d_G(u, v) \leq 2p$, and also from completing the path from u to u' and from v to v'.

Given some $u, v \in V$ with the guarantee that $d_G(u, v) \leq 3p$, we can find an index i such that $v \in D_i(u)$ and $v \notin D_{i-1}(u)$ (or that $v \in D_1(u)$) by binary search. The path between u, v in the SPT induced from $D_i(u)$ is of length at most $O(p^{i/s} \cdot kn^{1/k})$, and can be found in the same way as was described above. Since $v \notin D_{i-1}(u)$, $d_G(u, v) \geq (3p)^{(i-1)/s}$, and thus the stretch factor is only $O(p^{1/s} \cdot kn^{1/k})$. Combining this with the stretch factor of $O(t \cdot kn^{1/k})$ on the path from u' to v', we get total stretch $O((t + p^{1/s}) \cdot kn^{1/k})$. Note that we only use the collection of s covers twice per query. Specifically, all the skeleton missing paths will be filled in using the cover $\mathcal{C}_s$ as before. (The cover $\mathcal{C}_s$ has exactly the same parameters as the cover $\mathcal{C}$ from Section 6.1.) So the additive $O(\log s)$ term for the query time is surely dominated by the stretch. Choosing $t = k$, $p = n^{1/k}$ and $s = \lceil 1/\epsilon \rceil$ completes the proof. $\square$

7 Excluded Minor Graphs

In this section we show labeling scheme and path-reporting distance oracles for graphs that exclude a minor. Recall that a graph G excludes H as a minor, if no sequence of edge contractions and edge or vertex deletions on G can produce H. We focus on the family of K_r excluded graphs, as this is the richest family for a minor of size r. Our construction is based on sparse covers, similarly to Section 3, and yields the following result.

Theorem 7. *Let G be a graph on n vertices and diameter Λ which excludes K_r as a minor, then the following structures exist:*

- *A labeling scheme with stretch $O(1)$, space $O(\log n \log \Lambda \cdot e^{O(r)})$ per vertex and query time $O(\log \log \Lambda)$.*
- *A path-reporting distance oracle that can be distributed as a labeling scheme, with stretch $O(r^2)$, space $O(\log \Lambda \cdot \log n)$ per vertex and query time proportional to the returned path length plus $O(\log \log \Lambda)$.*

Previous work on labeling schemes for minor free graphs are essentially based on the *path separators* of [AG06], which extend the path separator for planar graphs due to [Tho04] (based on the classic separators of [LT79]). The main drawback of relying on these path separators, is that the dependency on the size of the excluded minor is quite bad. In [AG06], a distance labeling scheme for K_r-free graphs is shown with $1 + \epsilon$ stretch, $f(r) \cdot \log n/\epsilon$ words per vertex, and query time $f(r) \cdot \log n/\epsilon$, for any $\epsilon > 0$, where $f(r)$ is an extremely fast-growing function of r. In [KKS11] it was shown how to obtain a distance oracle with $O(n)$

space, at the price of increasing the query time to $f(r) \cdot (\log n/\epsilon)^2$. (Also, it is not known if their oracle can be converted into a distance labeling scheme.)

While both stretch and space of our construction are worse than previous work, the point is that the query time is only $O(\log\log\Lambda)$, and does not have the bad dependency on r. Also our space usage compares favorably with [AG06] for large values of r.

7.1 Covers for Minor-Free Graphs

In what follows we assume that the graph G excludes K_r as a minor. The seminal work of [KPR93], and its improvement by [FT03], can provide for any $\gamma > 0$, a *weak-diameter* $(O(r), 2^r, \gamma)$-sparse cover for G. If one desires a strong diameter, [AGMW10] showed a $(O(r^2), 2^r(r+1)!, \gamma)$-sparse cover, and concurrently [BLT07] showed a $(4, f(r)\cdot\log n, \gamma)$-sparse cover, where $f(r)$ is an extremely fast-growing function (coming from the Robertson-Seymour structure theorem).

We now use the latest padded partitions for minor free graphs to obtain a sparse cover. Our cover improves upon [AGMW10] when $r \geq \log\log n$. While the cover of [BLT07] has constant radius blow-up, the overlap of our cover is far better in terms of dependency on r.

Theorem 8. *For any graph G on n vertices which excludes K_r as a minor, the following covers exist for any γ:*

- *A weak diameter $(O(1), e^{O(r)}\cdot\log n, \gamma)$-sparse cover.*
- *A strong diameter $(O(r^2), \log n, \gamma)$-sparse cover.*

The following lemma is in [AGG+14].

Lemma 4. *Let G be a K_r-free graph, and let Δ be a parameter. Then there exists a universal constant $c > 0$ and a distribution over strong diameter Δ-bounded partitions, so that for all $v \in V$ and $0 \leq \beta \leq c/r^2$,*

$$\Pr_{P\sim\mathcal{P}}[B(x,\beta\Delta) \subseteq P(x)] \geq e^{-O(\beta r^2)} .$$

Also, there is exists a distribution of weak diameter Δ-bounded partitions, so that for all $v \in V$ and $0 \leq \beta \leq c$,

$$\Pr_{P\sim\mathcal{P}}[B(x,\beta\Delta) \subseteq P(x)] \geq e^{-O(\beta r)} .$$

Proof of Theorem 8. We start with the strong diameter partition. Using the same argument as in Section 3.1, one can construct a $(Cr^2n^{1/k}, k, \rho)$-sparse cover for sufficiently large constant C and any $\rho > 0$, by sampling a strong diameter partition k times from Lemma 4, with parameters $\Delta = Cr^2n^{1/k}\cdot\rho$ and $\beta = \rho/\Delta$ (which satisfy the condition on β when C is sufficiently large). The bounds on the radius and overlap are the same as in Section 3.1. We compute the probability that all ρ-balls are padded. The probability that a certain ball of radius $\rho = \beta\Delta$ is not padded by any of the k strong-diameter partitions is at most

$$\left(1 - e^{-O(\beta r^2)}\right)^k = \left(1 - e^{-O(r^2)/(Cr^2n^{1/k})}\right)^k \leq \left(\frac{1}{2n^{1/k}}\right)^k < 1/(2n) ,$$

where the inequality holds when C is sufficiently large. Using a union bound there is constant probability that all the balls are padded. In particular, setting $k = \log n$ we obtain a strong diameter $(O(r^2), \log n, \gamma)$-sparse cover. (More generally, fixing some parameter $t \geq r^2/c$, and setting $k = \log_{t/r^2} n$, yields a $(O(t), \log_{t/r^2} n, \gamma)$-sparse cover.)

A similar calculation for weak diameter, using the second part of Lemma 4, yields a weak diameter $(O(1), e^{O(r)}\cdot\log n, \gamma)$-sparse cover.

$\square$

7.2 Labeling Schemes and Path-Reporting Oracles for Minor Free Graphs

We now sketch the proof of Theorem 7. For the labeling scheme, we use the weak diameter $(O(1), e^{O(r)} \cdot \log n, \gamma)$-sparse cover of Theorem 8. Plugging this into the construction of Section 4 on a K_r-free graph with diameter Λ, yields a distance labeling scheme with stretch $O(1)$ and space $\log n \log \Lambda \cdot e^{O(r)}$ words per vertex. Note that in order to answer a query on u, v, we do a binary search on the $\log \Lambda$ levels to find an appropriate cluster $C_j(u)$, which takes only $O(\log\log \Lambda)$ time.

For the path-reporting distance oracle, we use the strong-diameter $(O(r^2), \log n, \gamma)$-sparse cover in a similar manner. Observe that the strong diameter is only required when we need to report paths, rather than just distances.

8 Conclusions

We gave space-efficient approximate distance oracles, distance labeling, and labeled routing for undirected graphs. Our distance oracles break the $n \log n$ space bound of Thorup and Zwick and can report approximate shortest paths in time proportional to their length. The cost is an increase in (multiplicative and/or additive) stretch. For distance labeling and routing, we break the previously best known space bound of order $\log n$ words at the cost of larger stretch.

It might be possible to improve preprocessing of our distance oracles, e.g., by using techniques from [WN12] for graphs that are not too sparse. Note that the oracle of Mendel and Naor achieves linear space and logarithmic stretch but it can only report approximate distances, not paths. We state it as an open problem whether a path-reporting oracle with linear space and polylogarithmic stretch exists which reports a path in time proportional to its length.

References

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In *PODC*, pages 188–197, 2006.

[AGG+14] Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. In *To appear in Proceedings of the 46th ACM Symposium on Theory of Computing*, STOC '14, 2014.

[AGHP11] Rachit Agarwal, Philip Brighten Godfrey, and Sariel Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM*, pages 1754–1762, 2011.

[AGHP12] Rachit Agarwal, Brighten Godfrey, and Sariel Har-Peled. Faster approximate distance queries and compact routing in sparse graphs. *CoRR*, abs/1201.2703, 2012.

[AGHP14] Rachit Agarwal, Brighten Godfrey, and Sariel Har-Peled, 2014. Personal communication.

[AGM+08] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. *ACM Trans. Algorithms*, 4(3):37:1–37:12, July 2008.

[AGMW10] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. *Theory Comput. Syst.*, 47(4):837–855, 2010.

[ANLP90] Baruch Awerbuch, Amotz Bar Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *J. Algorithms*, 11(3):307–341, September 1990.

[AP90a] B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, SFCS '90, pages 514–522 vol.2, Washington, DC, USA, 1990. IEEE Computer Society.

[AP90b] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, SFCS '90, pages 503–513 vol.2, Washington, DC, USA, 1990. IEEE Computer Society.

[AP92] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM J. Discret. Math.*, 5(2):151–162, May 1992.

[AP95] Baruch Awerbuch and David Peleg. Online tracking of mobile users. *J. ACM*, 42(5):1021–1058, September 1995.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 184–, Washington, DC, USA, 1996. IEEE Computer Society.

[BLT07] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Improved sparse covers for graphs excluding a fixed minor. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 61–70, New York, NY, USA, 2007. ACM.

[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.

[Che14] Shiri Chechik. Approximate distance oracles with constant query time. In *To appear in Proceedings of the 46th ACM Symposium on Theory of Computing*, STOC '14, 2014.

[Cow99] Lenore J. Cowen. Compact routing with minimum stretch. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 255–260, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[EGP98] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor (extended abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 11–20, New York, NY, USA, 1998. ACM.

[EP15] Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. In *SODA*, pages 805–821, 2015.

[Erd63] Paul Erdos. Extremal problems in graph theory. *Theory of Graphs and its Applications*, 1963.

[FT03] Jittat Fakcharoenphol and Kunal Talwar. An improved decomposition theorem for graphs excluding a fixed minor. *RANDOM-APPROX*, pages 36–46, 2003.

[HPM06] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput*, 35(5):1148–1184, 2006.

[KKS11] Ken-Ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I*, ICALP'11, pages 135–146, Berlin, Heidelberg, 2011. Springer-Verlag.

[KPR93] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *STOC*, pages 682–690, 1993.

[LT79] Richard J. Lipton and Robert E. Tarjan. A SEPARATOR THEOREM FOR PLANAR GRAPHS. *Siam Journal on Applied Mathematics*, 1979.

[MN07] Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. *Journal of the European Mathematical Society*, 9(2):253–275, 2007.

[NS07] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, New York, NY, USA, 2007.

[Pel93] David Peleg. Distance-dependent distributed directories. *Inf. Comput.*, 103(2):270–298, April 1993.

[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.

[PU88] David Peleg and Eli Upfal. A tradeoff between space and efficiency for routing tables. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 43–52, New York, NY, USA, 1988. ACM.

[Tho04] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, November 2004.

[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.

[WN12] Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 202–208. SIAM, 2012.

[WN13] Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 539–549. SIAM, 2013.