A Linear-Size Logarithmic Stretch Path-Reporting Distance Oracle for General Graphs

MICHAEL ELKIN, Ben-Gurion University of the Negev, Beer-Sheva, Israel
SETH PETTIE, University of Michigan, Ann Arbor

Thorup and Zwick [2001a] proposed a landmark distance oracle with the following properties. Given an n-vertex undirected graph $G = (V, E)$ and a parameter $k = 1, 2, \ldots$, their oracle has size $O(kn^{1+1/k})$, and upon a query (u, v) it constructs a path Π between u and v of length $\delta(u, v)$ such that $d_G(u, v) \leq \delta(u, v) \leq (2k-1)d_G(u, v)$. The query time of the oracle from Thorup and Zwick [2001a] is $O(k)$ (in addition to the length of the returned path), and it was subsequently improved to $O(1)$ [Wulff-Nilsen 2012; Chechik 2014]. A major drawback of the oracle of Thorup and Zwick [2001a] is that its space is $\Omega(n \cdot \log n)$. Mendel and Naor [2006] devised an oracle with space $O(n^{1+1/k})$ and stretch $O(k)$, but their oracle can only report distance estimates and not actual paths. In this article, we devise a path-reporting distance oracle with size $O(n^{1+1/k})$, stretch $O(k)$, and query time $O(n^{\epsilon})$, for an arbitrarily small constant $\epsilon > 0$. In particular, for $k = \log n$, our oracle provides logarithmic stretch using linear size. Another variant of our oracle has size $O(n \log\log n)$, polylogarithmic stretch, and query time $O(\log\log n)$.

For unweighted graphs, we devise a distance oracle with multiplicative stretch $O(1)$, additive stretch $O(\beta(k))$, for a function $\beta(\cdot)$, space $O(n^{1+1/k})$, and query time $O(n^{\epsilon})$, for an arbitrarily small constant $\epsilon > 0$. The tradeoff between multiplicative stretch and size in these oracles is far below Erdős's girth conjecture threshold (which is stretch $2k - 1$ and size $O(n^{1+1/k})$). Breaking the girth conjecture tradeoff is achieved by exhibiting a tradeoff of different nature between additive stretch $\beta(k)$ and size $O(n^{1+1/k})$. A similar type of tradeoff was exhibited by a construction of $(1+\epsilon, \beta)$-spanners due to Elkin and Peleg [2001]. However, so far $(1+\epsilon, \beta)$-spanners had no counterpart in the distance oracles' world.

An important novel tool that we develop on the way to these results is a distance-preserving path-reporting oracle. We believe that this oracle is of independent interest.

CCS Concepts: • **Theory of computation** → **Shortest paths**; **Routing and network design problems**; • **Information systems** → *Data compression*

Additional Key Words and Phrases: Distance oracles, distance preservers

50

A preliminary version of this article was published in SODA'15 [Elkin and Pettie 2015].
This research has been supported by the Israeli Academy of Science, grants 593/11 and 724/15, and by the Binational Science Foundation, grant 2008390. In addition, this research has been supported by the Lynn and William Frankel Center for Computer Science. A part of this research was performed while visiting the Center for Massive Algorithms (MADALGO), which is supported by Danish National Research Foundation grant DNRF84. Department of Computer Science, University of Michigan, Ann Arbor.
This research has been supported by the Binational Science Foundation, grant 2008390, and NSF grants CCF-1217338, CNS-1318294, and CCF-1514383. A part of this research was performed while visiting the Center for Massive Algorithms (MADALGO), which is supported by Danish National Research Foundation grant DNRF84.
Authors' addresses: M. Elkin, Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel; email: elkinm@cs.bgu.ac.il; S. Pettie, 2260 Hayward St., Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA; email: pettie@umich.edu.

DOI: http://dx.doi.org/10.1145/2888397

ACM Reference Format:
Michael Elkin and Seth Pettie. 2016. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. ACM Trans. Algorithms 12, 4, Article 50 (August 2016), 31 pages.
DOI: http://dx.doi.org/10.1145/2888397

1. INTRODUCTION

1.1. Distance Oracles for General Graphs

In the *distance oracle* problem, we wish to preprocess a weighted undirected n-vertex graph $G = (V, E)$. As a result of this preprocessing, we construct a compact data structure (which is called *distance oracle*) $\mathcal{D}(G)$, which, given a query pair (u, v) of vertices, will efficiently return a distance estimate $\delta(u, v)$ of the distance $d_G(u, v)$ between u and v in G. Moreover, the distance oracle should also compute an actual path $\Pi(u, v)$ of length $\delta(u, v)$ between these vertices in G. We say that a distance oracle is *path-reporting* if it does produce the paths $\Pi(u, v)$ as above; otherwise, we say that it is not path-reporting.

The most important parameters of a distance oracle are its stretch, its size, and its worst-case query time.[1] The *stretch* α of a distance oracle $\mathcal{D}(G)$ is the smallest (in fact, infimum) value such that for every $u, v \in V$, $d_G(u, v) \leq \delta(u, v) \leq \alpha \cdot d_G(u, v)$.

The term *distance oracle* was coined by Thorup and Zwick [2001a]. See their paper, also, for a very persuasive motivation of this natural notion. In their seminal paper, Thorup and Zwick [2001a] devised a path-reporting distance oracle (henceforth, TZ oracle). The TZ oracle with a parameter $k = 1, 2, \ldots$ has size $O(k \cdot n^{1+1/k})$, stretch $2k-1$, and query time $O(k)$. The size is measured in $\log n$-bit words. As argued in Thorup and Zwick [2001a], in Section 5, this tradeoff between size and stretch is essentially optimal for small k, assuming Erdős' girth conjecture, which implies that $\Omega(n^{1+1/k})$ bits of space are required for stretch $2k-1$, for any k. Note, however, that $k \cdot n^{1+1/k} = \Omega(n \cdot \log n)$, and Thorup and Zwick [2001a] left it open if one can obtain meaningful distance oracles of *linear* size, or more generally, of size $o(n \log n)$ words.

A partial answer to this question was provided by Mendel and Naor [2006], who devised a distance oracle with size $O(n^{1+1/k})$ words, stretch $O(k)$, and query time $O(1)$. Alas, their distance oracle is *inherently not path-reporting*. Specifically, the oracle of Mendel and Naor [2006] stores a collection of $O(k \cdot n^{1/k})$ hierarchically-separated trees (henceforth, HSTs; See Bartal [1996] for its definition), whose sizes sum up to $O(n^{1+1/k})$. The query algorithm for this oracle can return paths from these HSTs, i.e., paths which, at best, can belong to the metric closure of the original graph. These paths will typically not belong to the graph itself.

One can try to convert this collection into a collection of low-stretch spanning trees of the input graph G using star-decomposition or petal-decomposition techniques (see Elkin et al. [2005] and Abraham and Neiman [2012]). However, each of these spanning trees is doomed to have $n-1$ edges, making the size of the entire structure as large as $\Omega(k \cdot n^{1+1/k})$. In addition, with the current state-of-the-art techniques with low-stretch spanning trees, one can only achieve bounds which are somewhat worse than the optimal ones, achievable with HSTs. Hence, the approach that we have just outlined will probably produce an oracle with stretch $\omega(k)$, while using space $O(k \cdot n^{1+1/k})$.

Another result in this direction was recently obtained by Elkin et al. [2014]. For a parameter $t \geq 1$ their oracle uses size $O(n \cdot t \cdot \log_n w_{max})$ words, and provides stretch $O(\sqrt{t} \cdot n^{2/\sqrt{t}})$ for weighted graphs. (Henceforth, we will measure oracles' sizes in words.) The query time of their oracle is $O(\log(t \cdot \log_n w_{max}))$, where w_{max} is the aspect ratio of the graph, i.e., the ratio between the heaviest and the lightest edge. For unweighted

[1]The query time of all path-reporting distance oracles that we will discuss is of the form $O(q + |\Pi|)$, where Π is the path returned by the query algorithm. To simplify the notation, we will often omit the additive term of $O(|\Pi|)$.

graphs, their oracle exhibits roughly the same behavior. For a parameter $\epsilon > 0$, it has size $O(n \cdot t/\epsilon)$, provides stretch $O(t \cdot n^{1/t}(t + n^{\epsilon/t}))$, and has query time $O(1)$.

The distance oracles of Elkin et al. [2014] are the first path-reporting oracles that use $o(n \log n)$ space and provide non-trivial stretch. However, their stretch is by far larger than that of the oracles of Thorup and Zwick [2001a] and Mendel and Naor [2006]. Therefore, the tantalizing problem of whether one can have a linear-size path-reporting distance oracle with logarithmic stretch remained wide open. In the current article, we answer this question in the affirmative. For any k, $\frac{\log n}{\log \log n} \leq k \leq \log n$, and any arbitrarily small constant $\epsilon > 0$, our path-reporting distance oracle $D1$ has stretch $O(k)$, size $O(n^{1+1/k})$, and query time $O(n^{\epsilon})$. (When $\epsilon > 0$ is subconstant, the stretch becomes $O(k) \cdot (1/\epsilon)^{O(1)}$, the space becomes $O(n \log 1/\epsilon)$, and the query time is at most $O(n^{\epsilon} \cdot \log n)$. See Theorem 6.3, and the discussion that follows it, for more details.) Hence, our oracle achieves an optimal tradeoff, up to constant factors, between size and stretch in the range $\frac{\log n}{\log \log n} \leq k \leq \log n$, i.e., in the range "missing" in the Thorup-Zwick's result. Though our query time is n^{ϵ} for an arbitrarily small constant, $\epsilon > 0$ is much larger than Thorup-Zwick's query time, we stress that all existing path-reporting distance oracles either use space $\Omega(n \cdot \log n)$ [Thorup and Zwick 2001a; Wulff-Nilsen 2012; Chechik 2014] or have stretch $n^{\Omega(1)}$ [Elkin et al. 2014]. The query time of the TZ oracle was recently improved to $O(1)$ in Wulff-Nilsen [2012] and Chechik [2014]. The only previously existing path-reporting distance oracle that achieves the optimal tradeoff in this range of parameters can be obtained by constructing a $(2k-1)$-spanner[2] with $O(n^{1+1/k})$ edges and answering queries by conducting Dijkstra explorations in the spanner. However, with this approach, the query time is $O(n^{1+1/k})$. Our result is a drastic improvement of this trivial bound from $O(n \log n)$ to $O(n^{\epsilon})$, for an arbitrarily small constant $\epsilon > 0$.

We can also trade between the stretch and the query time. Specifically, a variant $D2$ of our oracle uses $O(n \log \log n)$ space, has stretch $O(\log^{\log_{4/3} 7} n) \approx O(\log^{6.76} n)$, and query time $O(\log \log n)$, and more generally, for any $k = 2, 3, \ldots, O(\log n)$, uses space $O(n^{1+1/k} \cdot \log k)$, has stretch $O(k^{6.76})$, and query time $O(\log k)$. For a comparison, the path-reporting distance oracle of Elkin et al. [2014] with this stretch uses space $\Omega(n \cdot \frac{\log n}{\log \log n})$ and has query time $O(\log \log n + \log \log_n w_{max})$, i.e., both its space and query time are larger than those of our oracle. Also, in the regime, when the oracle of Elkin et al. [2014] uses nearly the same space $O(n \log \log n)$ as our oracle,[3] its stretch becomes $2^{O(\frac{\log n}{\sqrt{\log \log n}})}$, while our stretch is polylogarithmic in n. The query time of the oracle of Elkin et al. [2014] in this regime is, however, $O(\log^{(3)} n + \log \log_n w_{max})$, while our query time is $O(\log \log n)$. These two expressions are incomparable. Our oracle exhibits analogous exponential improvements in the stretch in comparison with the oracle of Elkin et al. [2014] in many other points on the tradeoff curve in the relevant size range, (i.e., when the size is $o(n \log n)$), e.g., when the size is $O(n \log^{\delta} n)$, for any constant $\delta > 0$. In general, the size-stretch tradeoff of our oracle is better than that of Elkin et al. [2014] in the entire relevant size range, but in some points on the tradeoff curve, the query time of Elkin et al. [2014] might be (depending on w_{max}) smaller than ours. Also, the oracle of Elkin et al. [2014] provides meaningful results even for size $o(n \log \log n)$, while our oracle $D2$ never gets that sparse. Our oracle $D1$ can have linear size, but its query time is n^{ϵ}, for an arbitrarily small constant $\epsilon > 0$.

[2]For a parameter $t \geq 1$, $G' = (V, H)$ is a *t-spanner* of a graph $G = (V, E)$, $H \subseteq E$, if $d_H(u, v) \leq t \cdot d_G(u, v)$.
[3]In fact, the oracle of Elkin et al. [2014] uses a slightly larger space $O(n(\log \log n + \log \log_n w_{max}))$.

1.2. Distance Oracles with Stretch (α, β) for Unweighted Graphs

We say that a distance oracle $\mathcal{D}(G)$ provides stretch (α, β) for a pair of parameters $\alpha \geq 1, \beta \geq 0$ if for any query (u, v) it constructs a path $\Pi(u, v)$ of length $\delta(u, v)$ which satisfies $d_G(u, v) \leq \delta(u, v) \leq \alpha \cdot d_G(u, v) + \beta$. The notion of (α, β)-stretch is originated from the closely related area of *spanners*. A subgraph $G' = (V, H)$ is said to be an (α, β)-*spanner* of a graph $G = (V, E)$, $H \subseteq E$, if for every pair $u, v \in V$, it holds that $d_H(u, v) \leq \alpha \cdot d_G(u, v) + \beta$.

This notion was introduced in Elkin and Peleg [2001], where it was shown that for any $\epsilon > 0$ and $k = 1, 2, \ldots$, for any n-vertex unweighted graph $G = (V, E)$ there exists a $(1+\epsilon, \beta)$-spanner with $O(\beta \cdot n^{1+1/k})$ edges, where $\beta = \beta(\epsilon, k)$ is independent of n. Later, a number of additional constructions of $(1 + \epsilon, \beta)$-spanners with similar properties were devised in Elkin [2001], Thorup and Zwick [2006], and Pettie [2009].

It is natural to attempt converting these constructions of spanners into distance oracles with a similar tradeoff between stretch and size. However, generally so far, such attempts were not successful. See, e.g., the discussion titled "Additive Guarantees in Distance Oracles" in the introduction of Pătraşcu and Roditty [2010]. Pătraşcu and Roditty [2010] devised a distance oracle with stretch $(2, 1)$ and size $O(n^{5/3})$, and query time $O(1)$. Abraham and Gavoille [2011] generalized the result of Pătraşcu and Roditty [2010] to devise a distance oracle with stretch $(2k - 2, 1)$, query time $O(k)$, and size $\tilde{O}(n^{1+(2/(2k-1))})$.

Note, however, that neither of these previous results achieves multiplicative stretch $o(k)$ with size $O(n^{1+1/k})$, at the expense of an additive stretch. (This is the case with the result of Elkin and Peleg [2001] in the context of *spanners*, where the multiplicative stretch becomes as small as $1 + \epsilon$, for an arbitrarily small $\epsilon > 0$.) In this article, we devise the first distance oracles that do achieve such a tradeoff. Specifically, our path-reporting distance oracle has stretch $(O(1), \beta(k))$, size $O(n^{1+1/k})$, $\beta(k) = k^{O(1)}$, and query time $O(n^{\epsilon})$, for an arbitrarily small $\epsilon > 0$. The multiplicative stretch $O(1)$ here is a polynomial function of $1/\epsilon$, but it can be made much smaller than k. (Think, e.g., of $\epsilon > 0$ being a constant and k being a slowly growing function of n.) We can also have stretch $(o(k), \beta(k))$, size $O(n^{1+1/k})$, and query time $n^{O(k^{-\gamma})}$, where $\gamma > 0$ is a universal constant. Specifically, the theorem holds, e.g., for $\gamma = 1/7$.

In both these results, the tradeoff between multiplicative stretch and size of the oracle is below Erdős' girth conjecture barrier, which is stretch $2k - 1$ and space $O(n^{1+1/k})$. In fact, it is known that when the additive stretch is 0, distance oracles for general n-vertex graphs that have size $O(n^{1+1/k})$ must have multiplicative stretch $\Omega(k)$ [Thorup and Zwick 2001a; Lubotsky et al. 1988; Lazebnik and Ustimenko 1995]. Our results, like the results of Elkin and Peleg [2001] for spanners, break this barrier by introducing an additive stretch $\beta(k)$. To the best of our knowledge, our distance oracles are the first distance oracles that exhibit this behavior.

Using known lower bounds we also show that there exist no distance labeling schemes with stretch $(O(1), \beta(k))$ and maximum label size $O(\beta(k) \cdot n^{1/k})$, but rather one needs labels of size $n^{\Omega(1)}$ for this. This is also the case for routing schemes. (See Section 2 for relevant definitions.) We also show that in the cell-probe model of computation, any distance oracle for unweighted, undirected n-vertex graphs with stretch $(O(1), \beta(k))$ and space $O(\beta(k) \cdot n^{1+1/k})$ has query time $\Omega(k)$. This is in contrast to distance oracles with multiplicative stretch, which can have constant query time [Mendel and Naor 2006; Chechik 2014].

We also show a higher *conditional* lower bound on our oracle. Specifically, we show that if there exists a distance oracle D for general unweighted graphs with the same properties as those of our oracle (i.e., stretch $(O(1), k^{O(1)})$, size $O(n^{1+1/k})$), and *polylogarithmic* in n query time, then there exists a distance oracle D' for sparse unweighted

graphs with near-linear size $\tilde{O}(n)$, constant stretch and polylogarithmic in n query time. Moreover, if D is path-reporting, then D' is path-reporting, as well. On the other hand, the currently best-known distance oracle for sparse unweighted graphs with near-linear size and constant stretch has query time $n^{\Omega(1)}$ [Agarwal et al. 2011]. (See Sections 1.3 and 7.3 for more details.)

1.3. Distance Oracles for Sparse Graphs

In recent years a significant research effort was invested in distance oracles for *sparse* graphs. See, e.g., a recent survey of Roditty [2015] devoted specifically to this subject. Typically, by "sparse," one means a graph with $m = \tilde{O}(n)$ edges. This line of research is motivated by the fact that the lower bound based on Erdős girth conjecture is applicable only to dense graphs. Pătraşcu and Roditty [2010] devised a distance oracle for sparse unweighted graphs with stretch 2, query time $O(1)$, and space $\tilde{O}(n^{5/3})$. Pătraşcu et al. [2012] extended this result to weighted graphs, and generalized it to higher values of stretch and smaller space. Like in the results of Thorup and Zwick [2001a] and Mendel and Naor [2006], when the stretch is constant, the oracle of Pătraşcu et al. [2012] requires a superlinear size, even when $m = O(n)$. Agarwal et al. [2011], Porat and Roditty [2013], Agarwal and Godfrey [2013], and Agarwal [2014b] explored space-stretch-time tradeoff for distance oracles for sparse graphs when the stretch is at most 2. All these oracles [Agarwal et al. 2011; Porat and Roditty 2013; Agarwal and Godfrey 2013; Agarwal 2014b] also have superlinear size, even when $m = O(n)$. Their query time is also at least polynomial in n.

In addition to an oracle with stretch 2 with super-linear size and polynomial query time, Agarwal et al. [2011] also devised a not path-reporting [4] linear-size distance oracle for sparse graphs which, given a parameter $k = 2, 3, \ldots$, provides distance estimates with stretch $4k - 5$, and has query time $O(n^{1/k})$. (Both their and our results are, in fact, more general than this. We provide the results just for $m = O(n)$ to facilitate the comparison.) To our knowledge, prior to our work, this was the only oracle with linear size (for graphs with $m = O(n)$) and constant stretch. We devise the *first path-reporting counterpart* of their result. Our oracle (Corollary 6.4) also uses linear size, has stretch $O(k^{\log_{4/3} 7})$, and query time $O(n^{1/k})$, for any constant parameter k of the form $k = (4/3)^h$, $h = 1, 2, \ldots$.

From the technical perspective, the not path-reporting distance oracle of Agarwal et al. [2011] is very simple. One samples roughly $n^{1-\frac{1}{k}}$ landmarks and builds the TZ oracle for the metric induced by them with parameter k. On the other hand, our path-reporting oracle is more involved. Specifically, we use a hierarchy of sampled sets of landmarks, and distance-preserving path-reporting oracles (see Section 1.4) for each level of the hierarchy.

Finally, we remark that by the lower bounds of Sommer et al. [2009] for distance oracles for sparse graphs, any linear-size oracle with stretch $k^{O(1)}$ must have query time $\Omega(\frac{\log n}{k^{O(1)} \cdot \log \log n})$. There is a significant gap between the upper bound $O(n^{1/k})$ of Agarwal et al. [2011] and ours, and Sommer et al.'s lower bound [Sommer et al. 2009]. Nevertheless, the latter implies that in contrast to the situation with distance oracles for general graphs where the query time can be made constant [Thorup and Zwick 2001a; Wulff-Nilsen 2012; Chechik 2014], in the context of linear-size distance oracles for sparse graphs with constant stretch, the query time must be at least $\Omega(\frac{\log n}{\log \log n})$.

[4] It was erroneously claimed in Agarwal et al. [2011] that all their distance oracles are path-reporting. While their distance oracles with stretch smaller than three are path-reporting (albeit their space requirement is superlinear), this is not the case for their oracles with stretch $4k - 1$, $k \geq 1$ [Agarwal 2014a].

1.4. A Distance-Preserving Path-Reporting Distance Oracle

In Coppersmith and Elkin [2005], the authors showed that for any n-vertex graph $G = (V, E)$ and a collection $\mathcal{P}$ of P pairs of vertices, there exists a subgraph $G' = (V, H)$ of size $O(\max\{n + \sqrt{n} \cdot P, \sqrt{P} \cdot n\})$, so that for every $(u, v) \in \mathcal{P}$, $d_H(u, v) = d_G(u, v)$. In this article, we devise the first distance-oracle counterpart of this result. Specifically, our distance oracle uses $O(n + P^2)$ space, and for any query $(u, v) \in \mathcal{P}$ it produces the exact shortest path Π between u and v in $O(|\Pi|)$ time, where $|\Pi|$ is the number of edges in Π. This oracle is deterministic.

We employ this distance oracle very heavily in all our other constructions.

Remark. The construction time of our distance-preserving oracle is $O(n \cdot P^2) + \tilde{O}(m \cdot \min\{n, P\})$. The construction time of our path-reporting oracle for sparse graphs is $\tilde{O}(m \cdot n) = \tilde{O}(n^2\lambda)$, where $\lambda = m/n$. The construction time of our oracles with nearly-linear space for general graphs is $\tilde{O}(n^{2+1/k})$. Finally, the construction time of our oracle for unweighted graphs with a hybrid multiplicative-additive stretch is $\tilde{O}(\beta(k)n^{2+1/k}) = k^{O(\log\log k)}\tilde{O}(n^{2+1/k})$. In both cases, k is the stretch parameter of the respective oracle.

1.5. Related Work

There is a huge body of literature about distance oracles by now. The history of this subject can be traced back to a seminal paper by Peleg [2000] on distance labeling, where for any parameter $k = 1, 2, \ldots$, he implicitly devised a distance oracle for general undirected weighted graphs with size $O(n^{1+1/k} \cdot k \cdot \log n \cdot \log w_{max})$, stretch $O(k)$, and query time $O(n^{1/k} \cdot k \cdot \log n \cdot \log w_{max})$. A similar result can be derived from Matousek's embedding of general metrics into ℓ_∞ with distortion $2k-1$ and dimension $O(n^{1/k} \cdot \log n)$ [Matousek 1996].

Baswana and Sen [2006], Baswana and Kavitha [2006], and Baswana et al. [2008] improved the preprocessing time of the TZ oracle.

1.6. Structure of the Article

We start with describing our distance preserving oracle (Section 3). We then proceed with devising our basic path-reporting oracle for sparse graphs (Section 4). This oracle can be viewed as a composition of an oracle from Agarwal et al. [2011] with our distance-preserving oracle from Section 3. The oracle is described for graphs with small arboricity. Its extension to general sparse graphs (based on a reduction from Agarwal et al. [2011]) is described in Section 5. Then, we devise a much more elaborate multi-level path-reporting oracle for sparse graphs. The oracle of Agarwal et al. [2011] and our basic oracle from Section 4 both use just one set of sampled vertices. Our multi-level oracle uses a carefully constructed hierarchy of sampled sets which enables us to get the query time down from $n^{1/2+\epsilon}$ to n^{ϵ}. Next, we proceed (Section 6) to using this multi-level oracle for a number of applications. Specifically, we use it to construct a linear-size logarithmic stretch path-reporting oracle with query time n^{ϵ}, linear-size polylogarithmic stretch path-reporting oracle with query time $O(\log\log n)$, and finally, oracles that break the girth barrier for unweighted graphs. Our lower bounds can be found in Section 7.

2. PRELIMINARIES

For a pair of integers $a \le b$, we denote $[a, b] = \{a, a + 1, \ldots, b\}$, and $[b] = [1, b]$. The arboricity of a graph G is given by $\lambda(G) = \max_{U \subseteq V, |U| \ge 2} \frac{|E(U)|}{|U|-1}$, where $E(U)$ is the set of edges induced by the vertex set U. We denote by $\deg_G(u)$ the degree of a vertex u in G; we omit G from this notation whenever G can be understood from the context. We

use the notation $\tilde{O}(f(n)) = O(f(n)\text{polylog}(f(n)))$ and $\tilde{\Omega}(f(n) = \Omega(f(n)/\text{polylog}(f(n)))$. We say that a function $f()$ is *quasi-polynomial* if $f(n) \leq n^{\log^{O(1)} n}$.

Given two paths $\Pi = (x_1, x_2, \ldots, x_a)$ and $\Pi' = (x_a = y_1, y_2, \ldots, y_b)$, for some positive integers a, b, which share a common endpoint $x_a = y_1$, we denote by $\Pi \cdot \Pi'$ the *concatenation* path $(x_1, x_2, \ldots, x_a = y_1, y_2, \ldots, y_b)$.

A *distance-labeling scheme* for a graph $G = (V, E)$ assigns every vertex $v \in V$ a short label $\varphi(v)$. Given a pair of labels $\varphi(u), \varphi(v)$ of a pair of vertices $u, v \in V$, the scheme computes an estimate $\delta(\varphi(u), \varphi(v))$. This estimate has to be within a factor α, for some $\alpha \geq 1$, from the actual distance $d_G(u, v)$ between u and v in G. The parameter α is called the *stretch* of the labeling scheme, and the maximum number of bits employed by one of the labels is called the *(maximum) label size* of the scheme.

A closely related notion is that of *compact routing scheme*. Here, each vertex v is assigned a label $\varphi(v)$ and a routing table $\psi(v)$. Given a label $\varphi(u)$ of routing destination u and its own routing table $\psi(v)$, the vertex $v = v_0$ needs to be able to compute the next hop v_1. Given the table $\psi(v_1)$ of v_1 and the destination's label $\varphi(u)$, the vertex v_1 computes the next hop v_2, and so on. The resulting path $v = v_0, v_1, v_2, \ldots$ has to end up, eventually, in u, and its length needs to be at most α times longer than the length of the shortest $u - v$ path in G, for a stretch parameter $\alpha \geq 1$. In addition to stretch, another important parameter in this context is the maximum number of bits used by the label and the routing table (together) of any individual vertex. This parameter will be referred to as *maximum memory requirement* of a routing scheme.

3. A DISTANCE-PRESERVING PATH-REPORTING ORACLE

Consider an undirected weighted n-vertex graph $G = (V, E, \omega)$. Let Pairs $\subseteq V \times V$ be a subset of ordered pairs of distinct vertices. We denote its cardinality by $P = |\text{Pairs}|$. In this section, we describe a distance oracle which, given a pair $(u, v) \in$ Pairs, returns a shortest path $\Pi_{u,v}$ from u to v in G. The query time of the oracle is proportional to the number of edges (hops) $|\Pi_{u,v}|$ in $\Pi_{u,v}$. The oracle uses $O(n + P^2)$ space.

The construction of the oracle starts with computing a set Paths $= \{\Pi_{u,v} \mid (u, v) \in \text{Pairs}\}$ of shortest paths between pairs of vertices from Pairs. This collection of shortest paths is required to satisfy the property that if two distinct paths $\Pi, \Pi' \in$ Paths traverse two common vertices x and y in the same order (e.g., both traverse first x and then y), then they necessarily share the entire subpath between x and y. It is argued in Coppersmith and Elkin [2005] that this property can be easily achieved.

We will need the following definitions from Coppersmith and Elkin [2005].

For a path $\Pi = (u_0, u_1, \ldots, u_h)$ and a vertex $u_i \in V(\Pi)$, the *predecessor* of u_i in Π, denoted $\text{pred}_\Pi(u_i)$, is the vertex u_{i-1} (assuming that $i \geq 1$; otherwise, it is defined as NULL), and the *successor* of u_i in Π, denoted $\text{succ}_\Pi(u_i)$, is the vertex u_{i+1} (again, assuming that $i \leq h - 1$; otherwise, it is NULL).

Definition 3.1. Coppersmith and Elkin [2005] define a *branching event* (Π, Π', x) to be a triple with $\Pi, \Pi' \in$ being two distinct paths and $x \in V(\Pi) \cap V(\Pi')$ be a vertex that belongs to both paths and such that $\{\text{pred}_\Pi(x), \text{succ}_\Pi(x)\} \neq \{\text{pred}_{\Pi'}(x), \text{succ}_{\Pi'}(x)\}$. We will also say that the two paths Π, Π' branch at the vertex x.

Note that, under this definition, if Π traverses edges $(u_{i-1}, u_i), (u_i, u_{i+1})$ and Π' traverses edges $(u_{i+1}, u_i), (u_i, u_{i-1})$, then (Π, Π', u_i) is *not* a branching event.

It follows directly from the above property of the collection Paths (see also Coppersmith and Elkin [2005], Lemma 7.5, for a more elaborate discussion) that for every pair of distinct paths $\Pi, \Pi' \in$ Paths, there are at most two branching events that involve that pair of paths. Let $\mathcal{B}$ denote the set of branching events. The overall number of branching events for the set Paths is $|\mathcal{B}| \leq |\text{Paths}|^2 = P^2$. Our oracle will keep $O(1)$

data for each vertex, $O(1)$ data for each branching event, and $O(1)$ data for each path. Hence, the oracle stores $O(n + |\mathcal{B}| + P)$ data in total.

Specifically, in our oracle, for every vertex $v \in V$ we keep an identity of some path $\Pi \in$ Paths that contains v as an internal point, and two edges of Π incident on v. (If there is no path $\Pi \in$ Paths that contains v as an internal point, then our oracle stores nothing for v in this data structure.) The path Π stored for v will be referred to as the *home path* of v.

In addition, for every branching event (Π, Π', v) we keep the (at most four) edges of Π and Π' incident on v. Finally, for every pair $(x, y) \in$ Pairs we also store the first and the last edges of the path $\Pi_{x,y}$. Observe that the resulting space requirement is at most $O(n + |\mathcal{B}| + P) = O(n + P^2)$. We assume that the branching events are stored in a hash table of linear size, which allows membership queries in $O(1)$ time per query.

ALGORITHM 1: DPPRO Query(x, y)

1: Fetch the first edge (x, x') of $\Pi_{x,y}$
2: **if** $x' = y$ **then**
3: $\quad$ *Return*$((x, y))$
4: **else**
5: $\quad$ $Path' \leftarrow$ *Move_to*(x, y, x') {"Moving" to x', i.e., invoking the subroutine *Move_to* with x'}
6: $\quad$ *Return*$((x, x') \cdot Path')$
7: **end if**

ALGORITHM 2: Procedure *Move_to*(x, y, x')

1: **if** (x', y) is the last edge of $\Pi_{x,y}$ **then**
2: $\quad$ *Return*(x', y)
3: **else if** $(\Pi(x'), \Pi_{x,y}, x')$ is not a branching event **then**
4: $\quad$ Fetch the next edge (x', x'') of $\Pi(x')$
5: **else**
6: $\quad$ Fetch the next edge (x', x'') of $\Pi_{x,y}$
7: **end if**
8: $Path'' \leftarrow$ *Move_to*(x, y, x'') {A recursive invocation of Procedure *Move_to* with x''}
9: *Return*$((x', x'') \cdot Path'')$

The query algorithm proceeds as follows. See also Algorithm 1 for the pseudo-code. Given a pair $(x, y) \in$ Pairs, we find the first edge (x, x') of the path $\Pi_{x,y}$, and "move" to x'. This corresponds to the invocation of Procedure *Move_to* with the parameter x' on line 5 of Algorithm 1. The procedure itself is given in Algorithm 2. It accepts as input three parameters. The first two are the query vertices x, y, and the third one is a vertex $x' \in V(\Pi_{x,y})$.

Then, within Procedure *Move_to*, we check if (x', y) is the last edge of $\Pi_{x,y}$. If it is then we are done. Otherwise, let $\Pi(x')$ denote the home path of x'. Observe that since the vertex x' is an internal vertex in $\Pi_{x,y}$, it follows that there exists a home path $\Pi(x')$ for x'.

Next, we check if $\Pi(x') = \Pi_{x,y}$. This test is performed by comparing the identities of the two paths. If it is the case then we fetch the next edge (x', x'') of $\Pi(x')$, and move to x''. Otherwise (if $\Pi(x') \neq \Pi(x, y)$), then we check if the triple $(\Pi(x'), \Pi_{x,y}, x')$ is a branching event. This check is performed by querying the branching events' hash table.

If there is no branching event $(\Pi(x'), \Pi_{x,y}, x')$ then we again fetch the next edge (x', x'') of $\Pi(x')$, and move to x''. In fact, the algorithm does not need to separate between this

case and the case that $\Pi(x') = \Pi_{x,y}$. We distinguished between these cases here for clarity of presentation. See lines 3-4 of Algorithm 2.

Finally, if there is a branching event $(\Pi(x'), \Pi_{x,y}, x')$ then we fetch from our data structure all the information associated with this event. In particular, we fetch the next edge (x', x'') of $\Pi_{x,y}$, and move to x'' (line 6 of Algorithm 2).

In all cases, the procedure then recurses with x'' (line 8). It is easy to verify that, using appropriate hash tables, all queries can be implemented in $O(1)$ time per vertex, and in total $O(|\Pi_{x,y}|)$ time. We will write DPPRO as a shortcut for *distance-preserving path-reporting oracle*. The main result of this section is the following theorem.

THEOREM 3.2. *Given an undirected weighted graph $G = (V, E, \omega)$ and a collection* Pairs $\subseteq V \times V$ *of pairs of vertices, our DPPRO reports shortest paths $\Pi_{x,y}$ for query pairs $(x, y) \in$* Pairs *in $O(|\Pi_{x,y}|)$ time. The oracle employs $O(n + |\mathcal{B}| + P) = O(n + P^2)$ space, where $\mathcal{B}$ is the set of branching events for a fixed set of shortest paths between pairs of vertices from* Pairs*, and $P =$* |Pairs|.

Remark. In the preliminary version [Elkin and Pettie 2015] of this article, Theorem 3.2 was claimed for directed graphs. However, the proof argument given here (and in Elkin and Pettie [2015]) is valid only for undirected graphs.

One can construct the shortest paths in $\tilde{O}(m \cdot \min\{P, n\})$ time. Then, for each vertex v one keeps the list of paths that traverse v. For every such path, one keeps the two edges of this path which are incident on v. In overall $O(n \cdot P^2)$ additional time, one can use these lists to create the list of branching events. A hash table with them can be constructed in additional $O(P^2)$ time. Hence, the overall construction time of this oracle is $\tilde{O}(m \cdot \min\{P, n\}) + O(n \cdot P^2)$.

Observe that if one is given a set S, $|S| = O(n^{1/4})$, of terminals, then Theorem 3.2 provides a linear-size DPPRO (i.e., $O(1)$ words per vertex *on average*) which can report shortest paths between all pairs of terminals. It is well-known that any distance labeling scheme which is guaranteed to return exact distances between all pairs of $n^{1/4}$ terminals must use maximum label size $\Omega(n^{1/4})$ [Thorup and Zwick 2001a]. This is also the case for compact routing schemes [Thorup and Zwick 2001b]. In the latter case, the lower bound of $\Omega(n^{1/4})$ is on the maximum memory requirement of any individual vertex.

We remark that our DPPRO here employs $O(n + |\mathcal{B}| + P)$ space, whereas the underlying distance preserver has $O(n + \sqrt{n \cdot |\mathcal{B}|})$ edges [Coppersmith and Elkin 2005]. It is plausible that there exists a DPPRO of size $O(n + \sqrt{n \cdot |\mathcal{B}|})$. We leave this question open.

4. A BASIC DISTANCE ORACLE FOR GRAPHS WITH BOUNDED ARBORICITY

In this section we describe a basic variant of our path-reporting distance oracle for weighted undirected graphs $G = (V, E, \omega)$ of arboricity $\lambda(G) \leq \lambda$, for some parameter λ. (We will mostly use this oracle for constant or small values of λ. On the other hand, the result is meaningful for higher values of λ, as well.) Our oracle reports paths of stretch $O(k)$, for some positive integer parameter k. Unlike the partial oracle from Section 3, the oracle in this section is a full one, i.e., it reports paths for all possible queries $(u, v) \in \binom{V}{2}$. This is the case, also, for all our other oracles, which will be described in consequent sections. The expected query time of our oracle is $O(n^{1/2+\frac{1}{2k+2}} \cdot \lambda)$. (Whp[5], the

[5]Here and thereafter we use the shortcut "whp" for "with high probability". The meaning is that the probability is at least $1 - n^{-c}$, for some constant $c \geq 2$.

query time is $O(n^{1/2+\frac{1}{2k+2}} \cdot \log n \cdot \lambda)$.) The oracle requires $O(n)$ space, in addition to the space required to store the graph G, itself. Observe that for $\lambda = O(1)$, the query time is $O(n^{1/2+\epsilon})$, for an arbitrarily small constant $\epsilon > 0$, while the stretch is $O(\frac{1}{\epsilon}) = O(1)$. In Section 5, we extend this oracle to general m-edge n-vertex graphs with $\lambda = \frac{m}{n}$.

Our basic oracle employs just one level of sampled vertices, which we (following the terminology of Agarwal et al. [2011]) call *landmarks*. Each $v \in V$ is sampled independently at random with probability $\frac{\rho}{n}$, where ρ is a parameter which will be determined in the sequel. Denote by L the set of sampled vertices (landmarks). Note that $\mathbb{E}(|L|) = \rho$.

For every vertex $v \in V$, we keep the path $\Pi(v)$ from v to its closest landmark vertex $\ell(v)$, breaking ties arbitrarily. Denote by $D(v)$ the length $w(\Pi(v))$ of this path. This is a collection of vertex-disjoint shortest paths trees (shortly, SPTs) $\{T(u) \mid u \in L\}$, where each $T(u)$ is an SPT rooted at u for the subset $\{v \mid d_G(u, v) \leq d_G(u', v), \forall u' \neq u, u, u' \in L\}$. (Ties are broken arbitrarily.) This collection is a forest, and storing it requires $O(n)$ space.

The oracle also stores the original graph G. For the set of landmarks we compute the complete graph $\mathcal{L} = (L, \binom{L}{2}, d_G|L)$. Here, $d_G|L$ stands for the metric of G restricted to the point set L. (In other words, in the landmarks graph $\mathcal{L}$, for every pair $u, u' \in L$ of distinct landmarks, the weight $\omega_{\mathcal{L}}(u, u')$ of the edge (u, u') connecting them is defined by $\omega_{\mathcal{L}}(u, u') = d_G(u, u')$.)

Next, we invoke Thorup-Zwick's distance oracle [Thorup and Zwick 2001a] with a parameter k. (Henceforth, we will call it the *TZ oracle*.) One can also use here Mendel-Naor's oracle [Mendel and Naor 2006], but the resulting tradeoff will be somewhat inferior to the one that is obtained via the TZ oracle. Denote by $\mathcal{H}$ the TZ distance oracle for the landmarks graph $\mathcal{L}$. The oracle requires $O(k \cdot |L|^{1+1/k})$ space, and it provides $(2k-1)$-approximate paths $\Pi_{u,u'}$ in $\mathcal{L}$ for pairs of landmarks $u, u' \in L$. The query time is $O(k)$ (plus $O(|\Pi_{u,u'}|)$). Observe that some edges of $\Pi_{u,u'}$ may not belong to the original graph G. We note also that by using more recent oracles [Chechik 2014; Wulff-Nilsen 2012], one can have query time $O(1)$, but this improvement is immaterial for our purposes.

The TZ oracle $\mathcal{H}$ has a useful property that the union $H = \bigcup\{\Pi_{u,u'} \mid (u, u') \in \binom{L}{2}\}$ of all paths that the oracle returns forms a sparse $(2k-1)$-spanner. Specifically, $\mathbb{E}(|H|) = O(k \cdot |L|^{1+1/k})$. (This property holds for Mendel-Naor's oracle as well, but there, the stretch of the spanner is $O(k)$, where the constant hidden by the O-notation is greater than 2. On the other hand, their space requirement is $O(|L|^{1+1/k})$, rather than $O(k \cdot |L|^{1+1/k})$.) An invocation of the procedure of Thorup and Zwick that constructs an oracle for the landmarks' graph $\mathcal{L}$ returns a probability distribution of oracles $\mathcal{H}$, which, in turn, gives rise to a probability distribution of spanners H. Their expected size $\mathbb{E}(H)$ is $O(k \cdot |L|^{1+1/k})$. We fix a particular oracle $\mathcal{H}$ from this distribution that satisfies $|H| = O(k \cdot |L|^{1+1/k})$. Whp, such an $\mathcal{H}$ can be computed by running the procedure that computes the TZ oracle for $O(\log n)$ times. We will view the spanner H as a collection of pairs of vertices of our original graph G.

Finally, we invoke our DPPRO from Section 3 on the graph G and set the set Pairs to contain all edges of H. We will refer to this oracle as $\mathcal{D}(G, H)$. Its size is, with high probability, $O(n + |H|^2) = O(n + k^2 \cdot |L|^{2+2/k})$. Upon a query $(y, y') \in H$, this oracle returns a shortest path $\Pi_{y,y'}$ between y and y' in G in time $O(|\Pi_{y,y'}|)$.

Observe that $|L|$ is the sum of identical independent indicator random variables $|L| = \sum_{v \in V} I_v$, where I_v is the indicator random variable of the event $\{v \in L\}$. Hence, by Chernoff's inequality, for any constant $\epsilon > 0$,

$$\mathbb{P}(|L| > (1+\epsilon)\mathbb{E}(|L|)) = \mathbb{P}(|L| > (1+\epsilon) \cdot \rho) < \exp(-\Omega(\rho)).$$

We will set the parameter ρ to be at least $c \log n$, for a sufficiently large constant c. This will ensure that, whp, $|L| = O(\rho)$, and so $|L|^{2+2/k} = O(\rho^{2+2/k})$. Set ρ so that $k^2 \cdot \rho^{2+2/k} = \Theta(n)$, i.e., $\rho = n^{\frac{k}{2k+2}} \cdot \frac{1}{k}$. This guarantees that, aside from the storage needed for the original graph, the total space used by our oracle is $O(n)$.

This completes the construction algorithm of our oracle. Next, we describe its query algorithm. We need the following definition. For a vertex $v \in V$, let $\text{Ball}(v) = \{x \mid d_G(v, x) < d_G(v, \ell(v))\}$ denote the set of all vertices x which are closer to v than the closest landmark vertex $\ell(v)$ to v.

Given a pair u, v of vertices of G, our oracle starts with testing if $u \in \text{Ball}(v)$ and if $v \in \text{Ball}(u)$. To test if $u \in \text{Ball}(v)$ we just conduct a Dijkstra exploration rooted at v in the graph G, until we discover either u or $\ell(v)$. (Recall that G is stored in our oracle.) If u is discovered before $\ell(v)$, we conclude that $u \in \text{Ball}(v)$, and return the (exact) shortest path between them. Otherwise, we conclude that $u \notin \text{Ball}(v)$. Analogously, the algorithm tests if $v \in \mathit{Ball}(u)$.

Henceforth, we assume that $u \notin \text{Ball}(v)$ and $v \notin \text{Ball}(u)$, and therefore, the two searches returned $u' = \ell(u)$, $v' = \ell(v)$, and the shortest paths $\Pi(u)$ and $\Pi(v)$ between u and u' and between v and v', respectively. (In fact, using the forest of SPTs rooted at landmarks that our oracle stores, the query algorithm can compute shortest paths between u and u' and between v and v' in time proportional to the lengths of these paths.) Observe that $d_G(u', v') \le d_G(u', u) + d_G(u, v) + d_G(v, v')$, and $d_G(u', u), d_G(v, v') \le d_G(u, v)$. Hence, $d_G(u', v') \le 3 \cdot d_G(u, v)$.

Then, the query algorithm invokes the query algorithm of the oracle $\mathcal{H}$ for the landmarks graph $\mathcal{L}$. The latter algorithm returns a path $\Pi' = (u' = z_0, z_1, \ldots, z_h = v')$ in $\mathcal{L}$ between u' and v'. The length $\omega_{\mathcal{L}}(\Pi')$ of this path is at most $(2k-1) \cdot d_G(u', v') \le (6k-3) \cdot d_G(u, v)$. The time required for this computation is $O(k+h)$, where $|\Pi'| = h$. For each edge $(z_i, z_{i+1}) \in \Pi'$, $i \in [0, h-1]$, we invoke the query algorithm of the DPPRO $\mathcal{D}(G, H)$. (The edges (z_i, z_{i+1}) of the path Π' are typically not edges of the original graph. H is a $(2k-1)$-spanner of $\mathcal{L}$ produced by the oracle $\mathcal{H}$. Observe that $\Pi' \subseteq H$, and so $(z_i, z_{i+1}) \in H$, for every index $i \in [0, h-1]$.) The oracle $\mathcal{D}(G, H)$ returns a path $\tilde{\Pi}_i$ between z_i and z_{i+1} in G of length $\omega_{\mathcal{L}}(z_i, z_{i+1}) = d_G(z_i, z_{i+1})$. Let $\tilde{\Pi} = \tilde{\Pi}_0 \cdot \tilde{\Pi}_1 \cdot \ldots \cdot \tilde{\Pi}_{h-1}$ be the concatenation of these paths. Observe that $\tilde{\Pi}$ is a path in G between $z_0 = u'$ and $z_h = v'$, and

$$\omega(\tilde{\Pi}) = \sum_{i=0}^{h-1} \omega(\tilde{\Pi}_i) = \sum_{i=0}^{h-1} d_G(z_i, z_{i+1}) = \sum_{i=0}^{h-1} \omega_{\mathcal{L}}(z_i, z_{i+1}) = \omega_{\mathcal{L}}(\Pi') \le (6k-3) \cdot d_G(u, v).$$

Finally, the query algorithm returns the concatenated path $\hat{\Pi} = \Pi(u) \cdot \tilde{\Pi} \cdot \Pi(v)$ as the approximate path for the pair u, v. This completes the description of the query algorithm of our basic oracle. Observe that

$$\omega(\hat{\Pi}) = \omega(\Pi(u)) + \omega(\tilde{\Pi}) + \omega(\Pi(v)) \le d_G(u, v) + (6k-3) \cdot d_G(u, v) + d_G(u, v) = (6k-1) \cdot d_G(u, v).$$

Next, we analyze the running time of the query algorithm. First, consider the step that tests if $v \in \text{Ball}(u)$ and if $u \in \text{Ball}(v)$. Denote by X the random variable that counts the number of vertices discovered by some fixed Dijkstra exploration originated at u before the landmark $\ell(u)$ is discovered. We order all graph vertices by their distance from u in a non-decreasing order, i.e., $u = u_0, u_1, \ldots, u_{n-1}$, such that $d_G(u, u_i) \le d_G(u, u_j)$ for $i \le j$. Note that this is the order in which the aforementioned Dijkstra exploration originated at u discovers them. For an integer value $1 \le t \le n-1$, the probability that $X = t$ is equal to the probability that the vertices $u_0, u_1, \ldots, u_{t-1}$ are not all sampled and the vertex u_t is sampled. Hence, X is distributed geometrically with the parameter

$p = \rho/n$. Hence,

$$\mathbb{E}(X) = \sum_{t=1}^{n-1}(1-p)^t \cdot p \cdot t \leq \frac{1}{p} = \frac{n}{\rho}. \tag{1}$$

Also, obviously for any positive constant c, $\mathbb{P}(X > \frac{n}{\rho} c \ln n) \leq (1-\rho/n)^{(n/\rho)c \ln n} \leq n^{-c}$, i.e., whp, $X = O(\frac{n}{\rho}\log n)$.

Recall that the graph G has arboricity at most λ, and thus, any set of $n' \leq n$ vertices induces $O(n' \cdot \lambda)$ edges. Hence, the expected number of traversed edges by the Dijkstra algorithm is $O(\frac{n}{\rho}\lambda)$, and, whp, $O(\frac{n}{\rho}\lambda \log n)$ edges. In an unweighted graph, such exploration requires time linear in the number of edges, and in weighted[6] graphs, the required time is $O(\frac{n}{\rho}(\lambda + \log n))$ in expectation, and $O(\frac{n}{\rho}\lambda \cdot \log n)$, whp. (Recall that Dijkstra algorithm that scans a subgraph (V', E') requires time $O(|E'| + |V'|\log|V'|)$.)

The second step of our query algorithm queries the distance oracle $\mathcal{H}$ for the landmarks graph $\mathcal{L}$. The query is (u', v'), $u' = \ell(u)$, $v' = \ell(v)$. This query returns a path Π' between u' and v' in $\mathcal{L}$ in time $O(|\Pi'| + k)$. Next, for each of the $h = |\Pi'|$ edges (z_i, z_{i+1}), $i = 0, 1, \ldots, h-1$ of the path Π', the query algorithm invokes our DPPRO $\mathcal{D}(G, H)$ with the query (z_i, z_{i+1}). This oracle returns the shortest path $\tilde{\Pi}_i$ between z_i and z_{i+1} in G within time $O(|\tilde{\Pi}_i|)$. Finally, the algorithm returns the concatenated path $\hat{\Pi} = \Pi(u) \cdot \tilde{\Pi}_0 \cdot \tilde{\Pi}_1 \cdot \ldots \cdot \tilde{\Pi}_{h-1} \cdot \Pi(v)$. The running time required for producing the path $\tilde{\Pi}_0 \cdot \ldots \cdot \tilde{\Pi}_{h-1}$ is $O(\sum_{i=0}^{h-1}|\tilde{\Pi}_i|) = O(|\hat{\Pi}|)$, and $|\Pi'| \leq |\hat{\Pi}|$. Hence, the overall expected running time of the algorithm is $O(\frac{n}{\rho} \cdot \lambda + |\hat{\Pi}|)$ for unweighted graphs, and is $O(\frac{n}{\rho} \cdot (\lambda + \log n) + |\hat{\Pi}|)$ for weighted ones. We remark that the additive term of $O(k)$ is dominated by $O(\frac{n}{\rho} \cdot \lambda)$. To ensure this, we will be using $\rho \leq n/\log n$, and $k \leq O(\log n)$. For the high-probability bounds, one needs to multiply the first term of the running time by an additional $O(\log n)$ factor in both the unweighted and the weighted cases.

Now, we substitute $\rho = \frac{1}{k} \cdot n^{\frac{k}{2k+2}}$. The resulting expected query time becomes $O(k \cdot n^{\frac{1}{2}+\frac{1}{2k+2}} \cdot \lambda) + O(|\hat{\Pi}|)$. We summarize the properties of our basic oracle in the following theorem.

THEOREM 4.1. *For an undirected n-vertex graph G of arboricity λ and a positive integer parameter $k = 1, 2, \ldots$, there exists a path-reporting distance oracle of size (whp) $O(n)$ (in addition to the size required to store the input graph G) that returns $(6k-1)$-approximate shortest paths $\hat{\Pi}$. The expected query time is $O(n^{\frac{1}{2}+\frac{1}{2k+2}} \cdot k \cdot \lambda)$ in unweighted graphs and $O(n^{\frac{1}{2}+\frac{1}{2k+2}} \cdot k \cdot (\lambda + \log n))$ in weighted ones. (The same bounds on the query time apply, whp, if one multiplies them by $O(\log n)$. In addition, in all cases, the query time contains the additive term $O(|\hat{\Pi}|)$.)*

In particular, Theorem 4.1 implies that for any constant $\epsilon > 0$ one can have a path-reporting oracle with query time $O(n^{1/2+\epsilon}\lambda)$, which provides $O(1)$-approximate shortest paths for weighted undirected graphs. Observe, also, that for $k = 1$, we obtain a 5-approximate path-reporting oracle with query time $\tilde{O}(n^{3/4}\lambda)$. We remark that to get the latter oracle, one does not need to use the TZ oracle for the landmarks graph $\mathcal{L}$. Rather,

[6]One subtlety: we have to avoid scanning too many edges with just one endpoint in Ball(u). We store the edges incident to each vertex x in increasing order of their weights and relax them in that order when x is scanned. As soon as an edge (x, y) is relaxed such that the tentative distance to y is greater than $d_G(u, \ell(u))$, we can dispense with relaxing the remaining edges. Alternatively, a modification of the sampling rule which we describe in Section 5 also resolves this issue.

one can build a DPPRO $\mathcal{H}$ for all pairs of landmarks. (In this case $\rho = n^{1/4}$, $|L| = O(\rho)$, $|\text{Pairs}| = |\binom{L}{2}| = O(\rho^2) = O(\sqrt{n})$, and so the size of the oracle $\mathcal{H}$ is $O(|\text{Pairs}|^2 + n) = O(n)$.)

One can build the forest of SPTs rooted at the landmarks in $\tilde{O}(m)$ time. Within additional $O(m \cdot \rho + \rho \cdot n \cdot \log n) = O(k \cdot m \cdot n^{1/2-\frac{1}{2k+2}} + n^{\frac{3}{2}-\frac{1}{2k+2}} \cdot \log n)$ time one can construct the metric closure of L, i.e., the graph $\mathcal{L}$. This graph has $n' = \rho$ vertices and $m' \leq \rho^2$ edges. In $O(km' \cdot n'^{1/k}) = O(k\rho^{2+1/k}) = \tilde{O}(k \cdot n^{\frac{2k+1}{2k+2}})$ time, one can construct the TZ oracle for it. To construct the DPPRO with $P = O(k \cdot \rho^{1+1/k}) = O(k \cdot n^{1/2})$ pairs, one needs $O(n \cdot P^2) + \tilde{O}(k \cdot m \cdot n^{1/2-\frac{1}{2k+2}}) = O(k^2 \cdot n^2) + \tilde{O}(k \cdot m \cdot n^{1/2-\frac{1}{2k+2}})$ time. Hence, the overall construction time of this oracle is $O(k^2 \cdot n^2) + \tilde{O}(k \cdot m \cdot n^{1/2-\frac{1}{2k+2}})$.

In Section 5, we show (see Corollary 5.1) that Theorem 4.1 extends to general graphs with $m = \lambda \cdot n$ edges.

5. AN EXTENSION TO GENERAL GRAPHS

In this section, we argue that Theorem 4.1 can be extended to general n-vertex graphs $G = (V, E, \omega)$ with $m = \lambda n$ edges. In its current form, the theorem only applies to graphs of arboricity at most λ. While this is sufficient for our main application, i.e., for Theorem 6.9, our other application (Theorem 6.10) requires a more general result. Our extension is based on the reduction of Agarwal et al. [2011] of the distance oracle problem in general graphs to the same problem in bounded-degree graphs. Our argument is somewhat more general than the one from Agarwal et al. [2011], as it also applies to path-reporting distance oracles. We provide our extension for the sake of completeness.

THEOREM 5.1. *Up to constant factors, the result of Theorem 4.1 holds for general undirected unweighted m-edge n-vertex graphs with $m = \lambda n$. For undirected weighted graphs, the expected query time becomes $O(n^{1/2+\frac{1}{2k+2}} \cdot k \cdot \lambda \cdot \log n) = O(n^{1/2+\frac{1}{2k+2}} \cdot k \cdot \frac{m}{n} \cdot \log n)$, and the same bound applies, whp, if one multiplies it by another* $\log n$ *factor.*

PROOF. Given an m-edge n-vertex graph G with $\lambda = m/n$, we split each vertex u_i into $d(u) = \lceil \frac{\deg(u)}{\lambda} \rceil$ copies $u^{(1)}, u^{(2)}, \ldots, u^{(d(u))}$. Each copy is now selected independently at random with probability ρ/n, for a parameter ρ determined in the same way as in Section 4. The original vertex u is selected to the landmarks' set if and only if at least one of its copies (which will also be called *virtual nodes*) is selected. Observe that the rule that we have described is up to a constant factor equivalent to selecting u with probability $d(u) \cdot \frac{\rho}{n} = \lceil \frac{\deg(u)}{\lambda} \rceil \cdot \frac{\rho}{n}$.

The expected number of selected virtual nodes is

$$\sum_{v \in V} d(v) \cdot \frac{\rho}{n} = \frac{\rho}{n} \cdot \sum_{v \in V} \left\lceil \frac{\deg(u)}{\lambda} \right\rceil \leq \frac{\rho}{n} \sum_{v \in V} \left(\frac{\deg(v)}{\lambda} + 1 \right) = \rho + \frac{\rho}{\lambda n} \sum_{v \in V} \deg(v) = 3\rho.$$

The number $|L|$ of landmarks is at most the number of selected virtual nodes, and so $\mathbb{E}(|L|) \leq 3\rho$. By Chernoff's bound, the number of selected virtual nodes is, whp, $O(\rho)$, and so, whp, $|L|^{2+2/k} = O(\rho^{2+2/k})$, as well. Hence, the size of our oracle remains $O(n)$.

The rest of the construction algorithm for our distance oracle is identical to that of Section 4. (The only change is the distribution of selecting landmarks.) The query algorithm is identical to the query algorithm from Section 4. In particular, note that the virtual nodes have no effect on the computation, i.e., the returned paths contain only original vertices.

Next, we argue that the expected query time of the modified oracle is still at most $O(\frac{n}{\rho} \cdot \lambda)$ in unweighted graphs, and $O(\frac{n}{\rho} \cdot \lambda \log n)$ in weighted ones. (As usual, we omit the additive term of the number of edges of the returned path.) Specifically, we argue

that the tests, if $v \in \mathit{Ball}(u)$ and if $u \in \mathit{Ball}(v)$, can be carried out within the above expected time.

Let $u = u_0, u_1, \ldots, u_{n-1}$ be all graph vertices ordered by a Dijkstra exploration originated from u, and replace each vertex u_i by its $d(u_i)$ copies $u_i^{(1)}, \ldots, u_i^{(d(u_i))}$. The copies appear in an arbitrary order. Since each virtual node has probability $\frac{\rho}{n}$ to be selected independently of other vertices, it follows by a previous argument that the number N of virtual nodes that the algorithm encounters before seeing a selected virtual node is $O(\frac{n}{\rho})$. (The algorithm actually explores only original vertices. For the sake of this argument, we imagine that when the algorithm reaches a vertex y it reaches its first copy $y^{(1)}$. Right after that, it reaches the next copy $y^{(2)}$, and so on, and then reaches $y^{(d(y))}$. After "reaching" all these copies, the algorithm continues to the next original vertex.)

Denote the original vertices explored by the algorithm $u_1, u_2, \ldots, u_{i-1}, u_i$, and let u_i^h be a selected copy of u_i. (We assume that all copies of u_j, for $j < i$, are not selected, and all copies $u_i^{h'}$, $h' < h$, are also not selected.) It follows that $N = \sum_{j=1}^{i-1} d(u_j) + h$. Hence,

$$\mathbb{E}\left(\sum_{j=1}^{i-1} d(u_j)\right) \le \mathbb{E}(N) = O\left(\frac{n}{\rho}\right).$$

Hence,

$$\mathbb{E}\left(\sum_{j=1}^{i-1} \left\lceil \frac{\deg(u_j)}{\lambda} \right\rceil\right) = O\left(\frac{n}{\rho}\right)$$

as well. Thus,

$$\mathbb{E}\left(\sum_{j=1}^{i-1} \deg(u_j)\right) = O\left(\frac{\lambda n}{\rho}\right) = O\left(\frac{m}{\rho}\right).$$

Observe that the number of edges explored by the algorithm before reaching u_i is at most $\sum_{j=1}^{i-1} \deg(u_j)$. (The only edges incident on u_i explored by the algorithm are edges (u_j, u_i), for $j < i$. These edges are accounted for in the above sum of degrees.) Hence, the expected number of edges explored by the algorithm is $O(\frac{m}{\rho})$. Hence, its expected running time is $O(\frac{m}{\rho})$ (respectively, $O(\frac{m}{\rho} \cdot \log n)$) in unweighted (resp., weighted) graphs. The bounds that hold with high probability are higher, by a factor of $O(\log n)$. □

Since $\mathbb{E}(|L|) = O(\rho)$, the construction time of the oracle is, up to constant factors, the same as in Section 4.

This result provides a path-reporting analogue of the result of Agarwal et al. [2011], which provides stretch $O(k)$ and query time $(n\lambda)^{O(1/k)}$. Their oracle is not path-reporting. Our oracle is path-reporting, but its query time is significantly higher, specifically, it is $n^{1/2+O(1/k)} \cdot k \cdot \lambda$.

6. ORACLES WITH SMALLER QUERY TIME

In this section, we devise two path-reporting oracles with improved query time. The first oracle has size $O(m + n)$ (it stores the original graph), and query time $\lambda \cdot n^{\epsilon}$, for an arbitrarily small $\epsilon > 0$. The stretch parameter of this oracle grows polynomially with ϵ^{-1}. For the time being, we will focus on graphs of arboricity at most λ. The argument extends to general graphs with $m = \lambda n$ in the same way as was described in Section 5. Our second oracle has size $O(n \log\log n)$ (independent of the size of the original graph)

and reports stretch-$O(\log^{\log_{4/3} 7} n)$ paths in $O(\log \log n)$ time. Both draw on techniques used in sublinear additive spanner constructions of Pettie [2009]. We will later build upon the first oracle to construct additional oracles that work for dense graphs, as well. Like the second oracle, these later oracles will not have to store the input graph.

6.1. Construction of an Oracle with Time $O(\lambda \cdot n^{\epsilon})$

In this section, we describe the construction algorithm of our oracle. It will use a hierarchy of landmarks' sets $L_1, L_2, \ldots, L_h$, for a positive integer parameter h that will be determined later. For each index $i \in [h]$, every vertex v is selected into L_i independently at random with probability $p_i = \frac{\rho_i}{n}$, $\rho_1 > \rho_2 > \cdots > \rho_h$. The sequence $\rho_1, \rho_2, \ldots, \rho_h$ will be determined in the sequel. The vertices of L_i will be called the *i-level landmarks*, or shortly, the *i-landmarks*. For convenience of notation, we also denote $L_0 = V$.

For each vertex $v \in V$ and index $i \in [h]$, let $\ell_i(v)$ denote the closest i-landmark to v, where ties are broken in an arbitrary consistent way. Denote $r_i(v) = d_G(v, \ell_i(v))$ the distance between v and its closest i-landmark $\ell_i(v)$. Following [Pettie 2009], for a real number $0 < c \leq 1$, let $\mathcal{B}_i^c(v) = \{u \mid d_G(v,u) < c \cdot r_i(v)\}$ denote the *ith c-fraction-ball* of v. In our analysis, c will be set to either 1/3 or 1. Specifically, let $\mathcal{B}_i^{1/3}(v)$ denote the *one-third-ball* of v, and $\mathrm{Ball}_i(v) = \mathcal{B}_i^1(v) = \{u \mid d_G(v,u) < r_i(v)\}$ denote the *ith ball* of v.

For each vertex $v \in V$, we keep a shortest path between v and $\ell_1(v)$. (This is a forest of vertex-disjoint SPTs rooted at 1-landmarks. For each 1-landmark u', its SPT spans all vertices $v \in V$, which are closer to u' than to any other 1-landmark.) Similarly, for each $i \in [h-1]$ and every i-landmark u, we keep a shortest path between u and its closest $(i+1)$st landmark $\ell_{i+1}(u) = u^{(i+1)}$. Again, this entails storing a forest of vertex-disjoint SPTs rooted at $(i+1)$-landmarks, for each each index $i \in [h-1]$. Overall, this part of the oracle requires $O(n \cdot h)$ space.

For the hth-level landmarks' set L_h, we build a DPPRO $\mathcal{L}_h$ described in Section 3. Given a pair u, v of h-landmarks, this oracle returns a shortest path $\Pi(u,v)$ between them in time proportional to the number of edges in this path, i.e., $O(|\Pi(u,v)|)$. The space requirement of the oracle $\mathcal{L}_h$ is $O(n + |L_h|^4)$, and thus, we will select ρ_h to ensure that $|L_h|^4 = O(n)$, i.e., ρ_h will be roughly $n^{1/4}$. Denote also $\mathcal{P}_h = \binom{L_h}{2}$ be the set of all pairs of h-landmarks.

For each index $i \in [h-1]$, we also build a DPPRO $\mathcal{D}_i$ for the following set $\mathcal{P}_i$ of pairs of i-landmarks. Each pair of i-landmarks u, v, such that either $v \in \mathcal{B}_{i+1}^{1/3}(u)$ or $u \in \mathcal{B}_{i+1}^{1/3}(v)$ is inserted into $\mathcal{P}_i$.

Similarly to the DPPRO $\mathcal{L}_h$, given a pair $(u,v) \in \mathcal{P}_i$ for some $i \in [h-1]$, the oracle $\mathcal{D}_i$ returns a shortest path $\Pi(u,v)$ between u and v in time $O(|\Pi(u,v)|)$. Our oracle also stores the graph G itself. We will later show a variant of this oracle that does not store G (Theorem 6.6). The size of the oracle $\mathcal{D}_i$ is $O(n + |\mathrm{Branch}_i|)$, where Branch_i is the set of branching events for the set $\mathcal{P}_i$. Since we aim at a linear size bound, we will ensure that $|\mathrm{Branch}_i| = O(n)$, for every $i \in [h-1]$. We will also construct a hash table $\mathcal{H}_i$ for $\mathcal{P}_i$ of size $O(|\mathcal{P}_i|)$ that supports membership queries to $\mathcal{P}_i$ in $O(1)$ time per query. The resulting h-level oracle will be denoted Λ_h.

6.2. The Query Algorithm

Next, we describe the query algorithm of our oracle Λ_h. The query algorithm (see Algorithm 3 for the pseudo-code) is given a pair $u = u^{(0)}, v = v^{(0)}$ of vertices. The algorithm starts (line 1 of Algorithm 3) with testing if $u \in \mathrm{Ball}_1(v)$ and if $v \in \mathrm{Ball}_1(u)$. For this test, the algorithm just conducts a Dijkstra search from v until it discovers either $v^{(1)}$ or u (and, symmetrically, also conducts a search from u).

ALGORITHM 3: Hierarchical Query(u, v)

1: **if** $u \in \text{Ball}_1(v)$ or $v \in \text{Ball}_1(u)$ **then**
2: *Return*(shortest $u - v$ path) {The test is conducted via Dijkstra explorations. These explorations return either a shortest $u - v$ path, or shortest $u - u^{(1)}$ and $v - v^{(1)}$ paths.}
3: **else**
4: $Path \leftarrow Connect(u^{(1)}, v^{(1)}, 1)$
5: $Return(\Pi(u, u^{(1)}) \cdot Path \cdot \Pi(v^{(1)}, v))$
6: **end if**

ALGORITHM 4: Procedure $Connect(u^{(j)}, v^{(j)}, j)$

1: **if** $j = h$ **then**
2: $Return(\mathcal{L}_h(u^{(h)}, v^{(h)}))$ {Query the DPPRO $\mathcal{L}_h$ with the pair $(u^{(h)}, v^{(h)})$, and return the path that $\mathcal{L}_h$ returns.}
3: **else if** $(u^{(j)}, v^{(j)}) \in \mathcal{P}_j$ **then**
4: $Return(\Pi(u^{(j)}, v^{(j)}))$ {The condition is tested via the hash table $\mathcal{H}_j$, and the path is computed by the oracle $\mathcal{D}_j$.}
5: **else**
6: Fetch $\Pi(u^{(j)}, u^{(j+1)})$ and $\Pi(v^{(j)}, v^{(j+1)})$
7: $Path \leftarrow \Pi(u^{(j)}, u^{(j+1)}) \cdot Connect(u^{(j+1)}, v^{(j+1)}, j + 1) \cdot \Pi(v^{(j+1)}, v^{(j)})$
8: $Return(Path)$
9: **end if**

Observe that by Equation (1), the expected size of $\text{Ball}_1(v)$ and of $\text{Ball}_1(u)$ is $O(\frac{n}{\rho_1})$, and, whp, both these sets have size $O(\frac{n}{\rho_1} \cdot \log n)$. Hence, the running time of this step is, whp, $\tilde{O}(\frac{n}{\rho_1} \cdot \lambda)$. (Specifically, it is $O(\frac{n}{\rho_1} \cdot \lambda \cdot \log n)$ in unweighted graphs, and $O(\frac{n}{\rho_1} \cdot \log n \cdot (\lambda + \log n))$ in weighted ones. The expected running time of this step is smaller, by a factor of $\log n$, than the above bound.)

If the algorithm discovers that $v \in \text{Ball}_1(u)$ or that $u \in \text{Ball}_1(v)$, then it has found the shortest path between u and v. In this case, the algorithm returns this path (line 2 of Algorithm 3). Otherwise, it has found $u^{(1)} = \ell_1(u^{(0)})$ and $v^{(1)} = \ell_1(v^{(0)})$.

In general, consider a situation when, for some index j, $1 \le j \le h$, the algorithm has already computed $u^{(j)}$ and $v^{(j)}$. In this case, inductively, the algorithm has already computed shortest paths $\Pi(u^{(0)}, u^{(1)}), \Pi(u^{(1)}, u^{(2)}), \ldots, \Pi(u^{(j-1)}, u^{(j)})$ and $\Pi(v^{(0)}, v^{(1)}), \Pi(v^{(1)}, v^{(2)}), \ldots, \Pi(v^{(j-1)}, v^{(j)})$ between $u^{(0)}$ and $u^{(1)}$, $u^{(1)}$ and $u^{(2)}, \ldots, u^{(j-1)}$ and $u^{(j)}$, $v^{(0)}$ and $v^{(1)}$, $v^{(1)}$ and $v^{(2)}, \ldots, v^{(j-1)}$ and $v^{(j)}$, respectively. Note that the base case $j = 1$ has been just argued. The algorithm, then, invokes Procedure *Connect* (Algorithm 4) with parameters $u^{(j)}$, $v^{(j)}$ and j. This procedure accepts as input an index j, $1 \le j \le h$, and a pair of j-landmarks $u^{(j)}, v^{(j)} \in L_j$. It returns an approximately shortest path between them.

For $j < h$, the query algorithm of our oracle Λ_h then queries the hash table $\mathcal{H}_j$ whether the pair $(u^{(j)}, v^{(j)}) \in \mathcal{P}_j$ (line 3 of Algorithm 4). If it is the case, then the algorithm queries the oracle $\mathcal{D}_j$, which, in turn, returns the shortest path $\Pi(u^{(j)}, v^{(j)})$ between $u^{(j)}$ and $v^{(j)}$ in time $O(|\Pi(u^{(j)}, v^{(j)})|)$. The algorithm then reports the concatenated path

$$\begin{aligned}\Pi(u, v) \;=\; & \Pi(u^{(0)}, u^{(1)}) \cdot \Pi(u^{(1)}, u^{(2)}) \cdot \ldots \Pi(u^{(j-1)}, u^{(j)}) \cdot \Pi(u^{(j)}, v^{(j)}) \\ & \cdot \Pi(v^{(j)}, v^{(j-1)}) \cdot \ldots \cdot \Pi(v^{(2)}, v^{(1)}) \cdot \Pi(v^{(1)}, v^{(0)}).\end{aligned}$$

This is done on line 4 of Algorithm 4. Computing this concatenation requires $O(j) \le O(|\Pi(u, v)|)$ time.

In the complementary case when $(u^{(j)}, v^{(j)}) \notin \mathcal{P}_j$, the algorithm fetches the pre-recorded paths $\Pi(u^{(j)}, u^{(j+1)})$ and $\Pi(v^{(j)}, v^{(j+1)})$, and invokes itself recursively on the pair $(u^{(j+1)}, v^{(j+1)})$. This is done on line 7 of Algorithm 4. Recall that for each index j, $1 \leq j \leq h-1$, the algorithm stores a forest of vertex-disjoint SPTs rooted at $(j+1)$-landmarks L_{j+1}. These SPTs enable us to compute the paths $\Pi(u^{(j)}, u^{(j+1)})$, $\Pi(v^{(j)}, v^{(j+1)})$ for all $j \in [h-1]$, in time proportional to the number of edges in these paths.

Finally, if $j = h$, then we query the DPPRO $\mathcal{L}_h$ of the graph L_h with the query $(u^{(h)}, v^{(h)})$. Note that it is not necessary to query if $(u^{(h)}, v^{(h)})$ is in the DPPRO $\mathcal{L}_h$, since, by construction, all such pairs are there. The query returns the shortest path between them in time $O(|\Pi(u^{(h)}, v^{(h)})|)$. This is done on lines 1-2 of Algorithm 4. It follows that the overall running time of the query algorithm is dominated by the time required to compute $\Pi(u^{(0)}, u^{(1)})$ and $\Pi(v^{(0)}, v^{(1)})$. Specifically, it is

$$\tilde{O}\left(\frac{n}{\rho_1} \cdot \lambda\right) + \sum_{i=0}^{j-1} \left(|\Pi(u^{(i)}, u^{(i+1)})| + |\Pi(v^{(i)}, v^{(i+1)})|\right) + |\Pi(u^{(j)}, v^{(j)})|,$$

where $1 \leq j \leq h$ is the smallest index such that $(u^{(j)}, v^{(j)}) \in \mathcal{P}_j$. (Recall that for $j = h$, $\mathcal{P}_h = \binom{L_h}{2}$, i.e., all pairs of h-landmarks belong to $\mathcal{P}_h$.) Hence, the overall query time is $\tilde{O}(\frac{n}{\rho_1} \cdot \lambda) + O(|\Pi(u, v)| + h)$, where $\Pi(u, v)$ is the path that the algorithm ultimately returns.

Remark. If for each index $0 \leq j \leq h-1$ at least one of the subpaths $\Pi(u^{(j)}, u^{(j+1)})$, $\Pi(v^{(j)}, v^{(j+1)})$ is not empty, then $h \leq |\Pi(u, v)|$, and the resulting query time is $\tilde{O}(\frac{n}{\rho_1}\lambda) + O(|\Pi(u, v)|)$. One can artificially guarantee that all these subpaths will not be empty, i.e., that $u^{(j)} \neq u^{(j+1)}$ and $v^{(j)} \neq v^{(j+1)}$, for every j. To do this, one can modify the construction slightly so that the set of i-landmarks and the set of j-landmarks will be disjoint for all $i \neq j$. Under this modification of the algorithm, the query time is $\tilde{O}(\frac{n}{\rho_1} \cdot \lambda) + O(|\Pi(u, v)|)$, while the stretch guarantee of the oracle (which will be analyzed in Section 6.3) stays the same. This modification can make oracle's performance only worse than it is without this modification, but the bounds on the query time of the modified oracle, in terms of the number of edges in the returned path, become somewhat nicer. (See Theorem 6.6.)

6.3. The Stretch Analysis

Recall that in the case that $v \in \text{Ball}_1(u)$ or $u \in \text{Ball}_1(v)$, our algorithm returns the exact shortest path between $u = u^{(0)}$ and $v = v^{(0)}$. Hence, we next consider the situation when $v \notin \text{Ball}_1(u)$ and $u \notin \text{Ball}_1(v)$. For brevity, let $d = d^{(0)} = d_G(u, v)$. At this point, the algorithm also has already computed $u^{(1)}$ and $v^{(1)}$, along with the shortest paths $\Pi(u^{(0)}, u^{(1)})$ and $\Pi(v^{(0)}, v^{(1)})$ between $u^{(0)}$ and $u^{(1)}$ and between $v^{(0)}$ and $v^{(1)}$, respectively. Observe that in this scenario, we have $d_G(u^{(0)}, u^{(1)}), d_G(v^{(0)}, v^{(1)}) \leq d$, and so

$$d_G(u^{(1)}, v^{(1)}) \leq d_G(u^{(1)}, u^{(0)}) + d_G(u^{(0)}, v^{(0)}) + d_G(v^{(0)}, v^{(1)}) \leq 3 \cdot d.$$

Hence, if $(u^{(1)}, v^{(1)}) \in \mathcal{P}_1$, then the path $\Pi(u^{(0)}, u^{(1)}) \cdot \Pi(u^{(1)}, v^{(1)}) \cdot \Pi(v^{(1)}, v^{(0)})$ returned by the algorithm is a 5-approximate path between u and v. Indeed, its length is at most

$$d_G(u^{(0)}, u^{(1)}) + d_G(u^{(1)}, v^{(1)}) + d_G(v^{(1)}, v^{(0)}) \leq d + 3 \cdot d + d = 5 \cdot d.$$

More generally, suppose the query algorithm reached the j-level landmarks $u^{(j)}, v^{(j)}$, for some j, $1 \leq j \leq h-1$, and suppose that $(u^{(j)}, v^{(j)}) \notin \mathcal{P}_j$. This means that $v^{(j)} \notin \mathcal{B}_{j+1}^{1/3}(u^{(j)})$

and $u^{(j)} \notin \mathcal{B}^{1/3}_{j+1}(v^{(j)})$. By definition of the one-third-ball, it follows that

$$d_G(u^{(j)}, v^{(j)}) \geq \frac{1}{3} \cdot d_G(u^{(j)}, u^{(j+1)}) = \frac{1}{3} \cdot r_{j+1}(u^{(j)}),$$

and

$$d_G(u^{(j)}, v^{(j)}) \geq \frac{1}{3} \cdot d_G(v^{(j)}, v^{(j+1)}) = \frac{1}{3} \cdot r_{j+1}(v^{(j)}),$$

where $u^{(j+1)}$ (respectively, $v^{(j+1)}$) is the $(j+1)$-landmark closest to $u^{(j)}$ (resp., $v^{(j)}$).

Hence,

$$d_G(u^{(j+1)}, v^{(j+1)}) \leq d_G(u^{(j+1)}, u^{(j)}) + d_G(u^{(j)}, v^{(j)}) + d_G(v^{(j)}, v^{(j+1)}) \leq 7 \cdot d_G(u^{(j)}, v^{(j)}).$$

Denote by p, $1 \leq p \leq h$, the index for which the algorithm discovers that $(u^{(p)}, v^{(p)}) \in \mathcal{P}_p$. (Since $(u^{(h)}, v^{(h)}) \in \mathcal{P}_h$ for every pair $(u^{(h)}, v^{(h)})$ of h-landmarks, it follows that the index p is well-defined.)

We have seen that $d_G(u^{(1)}, v^{(1)}) \leq 3d$, and for every index j, $1 \leq j \leq p-1$, $d_G(u^{(j+1)}, v^{(j+1)}) \leq 7 \cdot d_G(u^{(j)}, v^{(j)})$. Hence, for every j, $1 \leq j \leq p$, it holds that $d_G(u^{(j)}, v^{(j)}) \leq 3 \cdot 7^{j-1} \cdot d$. Denote $d^{(j)} = 3 \cdot 7^{j-1} \cdot d$, for $0 \leq j \leq p$. Also, $d_G(u^{(0)}, u^{(1)}), d_G(v^{(0)}, v^{(1)}) \leq d = d^{(0)}$, and for every index j, $1 \leq j \leq p-1$,

$$d_G(u^{(j)}, u^{(j+1)}) \leq 3 \cdot d_G(u^{(j)}, v^{(j)}) \leq 3 \cdot d^{(j)} = 3^2 \cdot 7^{j-1} \cdot d.$$

Hence, the length of the path

$$\Pi(u^{(0)}, u^{(1)}) \cdot \ldots \cdot \Pi(u^{(p-1)}, u^{(p)}) \cdot \Pi(u^{(p)}, v^{(p)}) \cdot \Pi(v^{(p)}, v^{(p-1)}) \cdot \ldots \Pi(v^{(1)}, v^{(0)})$$

returned by the algorithm is at most

$$\begin{aligned} &d^{(0)} + 3 \cdot \left(\sum_{j=1}^{p-1} d^{(j)} \right) + d^{(p)} + 3 \cdot \left(\sum_{j=1}^{p-1} d^{(j)} \right) + d^{(0)} \\ &= d \cdot \left(2 \cdot \left(1 + 3 \cdot \left(\sum_{j=1}^{p-1} 3 \cdot 7^{j-1} \right) \right) + 3 \cdot 7^{p-1} \right) = d \cdot (6 \cdot 7^{p-1} - 1). \end{aligned}$$

Since $p \leq h$, we conclude that the oracle has stretch at most $6 \cdot 7^{h-1} - 1$.

6.4. The Size of the Oracle

For each index $i \in [h]$, our oracle stores a forest of (vertex-disjoint) SPTs rooted at i-landmarks. Each of these forests requires $O(n)$ space, i.e., together these h forests require $O(n \cdot h)$ space.

We next set the values $\rho_1 > \rho_2 > \cdots > \rho_h$, so that each of the auxiliary oracles $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{h-1}, \mathcal{L}_h$ requires $O(n)$ space. Each of the hash tables $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_h$ associated with these oracles requires less space than its respective oracle. Recall that the parameter ρ_1 also determines the query time. Specifically, it is $\tilde{O}(\frac{n}{\rho_1}\lambda) + O(|\Pi|)$, where Π is the path returned by the algorithm. In the sequel, we will often skip the additive term of $O(|\Pi|)$ when stating the query time.

For each $i \in [h]$, we write $\rho_i = n^{\alpha_i}$, where $\alpha_i = 1 - (3/4)^{h-i+1}$. Observe that $\alpha_h = 1/4$, i.e., $\rho_h = n^{1/4}$. Hence, $\mathbb{E}(|L_h|) = \rho_h = n^{1/4}$, and by Chernoff's bound, whp, $|L_h| = O(n^{1/4})$. (Recall that $|L_h|$ is a Binomial random variable.) Hence, the DPPRO $\mathcal{L}_h$ for $\mathcal{P}_h = \binom{L_h}{2}$ requires space $O(|L_h|^4 + n) = O(n)$, whp.

Next, we analyze the space requirements of the oracles $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{h-1}$. Fix an index $i \in [h-1]$, and recall that the space requirement of the DPPRO $\mathcal{D}_i$ is $O(n + |\text{Branch}_i| + |\mathcal{P}_i|)$, where Branch_i is the set of branching events for the set $\mathcal{P}_i$ of pairs of vertices. Next, we argue that (whp) $|\text{Branch}_i| = O(n)$. Recall that the set $\mathcal{P}_i$ contains all pairs of i-landmarks $(u^{(i)}, v^{(i)})$ such that either $v^{(i)} \in \mathcal{B}^{1/3}_{i+1}(u^{(i)})$ or $u^{(i)} \in \mathcal{B}^{1/3}_{i+1}(v^{(i)})$.

The following two lemmas from Pettie [2009] are the key to the analysis of the oracle's size. The first says that with our definition of $\mathcal{P}_{i+1}$ all branching events are confined to $(i+1)$st level balls. The second bounds the expected number of branching events in terms of the sampling probabilities. For completeness, the proofs of these lemmas are provided in Appendix A.

LEMMA 6.1. *Suppose that $v \in \mathcal{B}^{1/3}_{i+1}(u)$. Then, if $(x, y) \in \mathcal{P}_{i+1}$ and there is a branching event between the pairs (u, v) and (x, y), then necessarily $x, y \in \text{Ball}_{i+1}(u)$.*

LEMMA 6.2. *Whp, $|\text{Branch}_i| = O(\frac{\rho_i^4}{\rho_{i+1}^3} \cdot \log^3 n)$, and $\mathbb{E}(|\text{Branch}_i|) = O(\frac{\rho_i^4}{\rho_{i+1}^3})$. Moreover, whp, $|\mathcal{P}_i| = O(\frac{\rho_i^2}{\rho_{i+1}} \cdot \log n)$, and $\mathbb{E}(|\mathcal{P}_i|) = O(\frac{\rho_i^2}{\rho_{i+1}})$.*

Observe that with our choice of ρ_i ($\rho_i = n^{\alpha_i}$, $\alpha_i = 1 - (3/4)^{h-i+1}$, for every $i \in [h]$), it holds for every $i \in [h-1]$ that $O(\frac{\rho_i^4}{\rho_{i+1}^3}) = O(n^{4\alpha_i - 3\alpha_{i+1}}) = O(n)$, and $O(\frac{\rho_i^2}{\rho_{i+1}}) = O(n^{2\alpha_i - \alpha_{i+1}}) = O(n^{1-\frac{1}{2}(\frac{3}{4})^{h-i}})$. Hence, by Lemma 6.2, for each $i \in [h-1]$, the oracle $\mathcal{D}_i$ requires expected space $O(n + |\text{Branch}_i| + |\mathcal{P}_i|) = O(n)$. Thus, the overall expected space required by our h-level oracle oracle Λ_h (in addition to the space required to store the original graph G) is $O(n \cdot h)$. Recall that the query time is (whp) $\tilde{O}((n/\rho_1)\lambda) = \tilde{O}(n^{(3/4)^h} \cdot \lambda)$.

The argument described in Section 5 enables us to extend these results to general m-edge n-vertex graphs.

THEOREM 6.3. *For any parameter $h = 1, 2, \ldots$ and any n-vertex undirected possibly weighted graph G with arboricity λ, the path-reporting distance oracle Λ_h uses expected space $O(n \cdot h)$, in addition to the space required to store G. Its stretch is $(6 \cdot 7^{h-1} - 1)$, and its query time is (whp) $\tilde{O}(n^{(3/4)^h}\lambda)$. The same result applies for any m-edge n-vertex graph with $\lambda = m/n$.*

Specifically, in unweighted graphs with arboricity λ, the query time is $O((n/\rho_1) \cdot \lambda \cdot \log n) = O(n^{(3/4)^h} \cdot \lambda \cdot \log n)$, while in weighted graphs it is $O(n^{(3/4)^h} \cdot (\lambda + \log n) \log n)$. In unweighted m-edge n-vertex graphs, the query time is $O(n^{(3/4)^h} \cdot \frac{m}{n} \cdot \log n)$, while in m-edge n-vertex weighted graphs it is $O(n^{(3/4)^h} \cdot \frac{m}{n} \cdot \log^2 n)$. The expected query time in unweighted (respectively, weighted) graphs is $O(n^{(3/4)^h} \cdot \frac{m}{n} + h)$ (resp., $O(n^{(3/4)^h} \cdot (\frac{m}{n} + \log n) + h)$).

By introducing a parameter $t = (4/3)^h$, we get query time $\tilde{O}(n^{1/t}\lambda)$, space $O(n \cdot \log t)$, and stretch at most $t^{\log_{4/3} 7}$. (The exponent is ≈ 6.76.)

COROLLARY 6.4. *For any constant t of the form $t = (4/3)^h$, for a positive integer h, and an n-vertex graph G with arboricity λ, our path-reporting distance oracle Λ_h uses expected space $O(n)$ in addition to the space needed to store G. It provides stretch at most $t^{\log_{4/3} 7}$, and its query time is (whp) $\tilde{O}(n^{1/t}\lambda)$. (For a non-constant t, the space requirement becomes $O(n \cdot \log t)$.) The same result applies for any m-edge n-vertex graph with $\lambda = m/n$.*

Yet better bounds can be obtained if one is interested in small *expected* query time. The expected query time is dominated by the time required to test if $v \in \text{Ball}_1(u)$ and

if $u \in \text{Ball}_1(v)$. For unweighted graphs these tests require $O(\frac{n}{\rho_1}\lambda) = O(n^{(3/4)^h}\lambda)$ expected time.

COROLLARY 6.5. *For any t of the form $t = (4/3)^h$, for a positive integer h, and an n-vertex m-edge graph G, our path-reporting oracle Λ_h uses expected $O(n \cdot h)$ space in addition to the space required to store G. It provides stretch at most $t^{\log_{4/3} 7}$, and its expected query time is $O(n^{1/t} \cdot (m/n) + \log t)$ for unweighted graphs. In the case of weighted graphs, the expected query time is $O(n^{1/t}(m/n) \cdot \log n)$.*

Consider now the oracle Λ_h for a superconstant number of levels $h = \lceil \log_{4/3}(\log n + 1) \rceil$. Then, $\rho_1 = (2n)^{\alpha_1} = (2n)^{1-(3/4)^h} \geq n$. In other words, all vertices V of G are now defined as the first level landmarks (1-landmarks), i.e., $L_1 = V$. (For levels $i = 2, 3, \ldots, h$, landmarks L_i are still selected at random from V with probability $\rho_i/n < 1$, independently. For level 1 this probability is 1.) Recall that our oracle starts with testing if $v \in \text{Ball}_1(u)$ and if $u \in \text{Ball}_1(v)$. Now both these balls are empty sets, because all vertices belong to L_1. Thus, with this setting of parameters, the oracle Λ_h no longer needs to conduct this time-consuming test. Rather, it proceeds directly to querying the oracle $\mathcal{D}_1$. Remarkably, this variant of our oracle does not require storing the graph G. (Recall that the graph was only used by the query algorithm for testing if $v \in \text{Ball}_1(u)$ and if $u \in \text{Ball}_1(v)$.) The query time of the new oracle is now dominated by the h queries to the oracles $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{h-1}, \mathcal{L}_h$, i.e., $O(h) = O(\log \log n)$. Recall that, by the remark at the end of Section 6.2, one can always make our oracle to return paths with at least h edges, and thus, the $O(h) = O(\log \log n)$ additive term in the query time can be swallowed by $O(|\Pi|)$, where Π is the path that our oracle returns.

Denote by $\tilde{\Lambda}$ the oracle which was just described. The stretch of $\tilde{\Lambda}$ is (by Theorem 6.3) $6 \cdot 7^{h-1} - 1 = O(\log^{\log_{4/3} 7} n)$.

THEOREM 6.6. *The oracle $\tilde{\Lambda}$ is a path-reporting oracle with expected space $O(n \log \log n)$, where n is the number of vertices of its input undirected weighted graph G. Its stretch is $O(\log^{\log_{4/3} 7} n)$ and its query time is $O(\log \log n)$. (It can be made $O(1)$, but the paths returned by the oracle will then contain $\Omega(\log \log n)$ edges.)*

Note that by Markov's inequality, Theorem 6.6 implies that one can produce a path-reporting oracle with space $O(n \log \log n)$, query time $O(\log \log n)$, and polylogarithmic stretch by just repeating the above oracle-constructing algorithm for $O(\log n)$ times. Whp, in one of the executions, the oracle's space will be $O(n \log \log n)$. Similarly, by the same Markov's argument, Corollary 6.4 implies that, whp, one can have the space of the oracle Λ_h bounded by $O(n)$ (in addition to the space required to store the input graph).

Next, we analyze the construction time of our oracle. The h forests rooted at landmarks can be constructed in $\tilde{O}(m \cdot h)$ time. We also spend $\tilde{O}(m \cdot n) = \tilde{O}(n^2\lambda)$ time to compute all-pairs-shortest-paths (henceforth, APSP). Then, for each ball $B_{i+1}(u)$, $u \in L_i$, we store all i-landmarks that belong to it. They can be fetched from the APSP structure in $O(1)$ time per i-landmark. The expected size of this data structure is $O(|\mathcal{P}_i|) = O(\frac{\rho_i^2}{\rho_{i+1}}) = O(n)$. Then, we produce all possible quadruples u, v, x, y with $v, x, y \in \text{Ball}_{i+1}(u) \cap L_i$, $u \in L_i$. By the proof of Lemma 6.2, there are expected $O(\frac{\rho_i^4}{\rho_{i+1}^3}) = O(n)$ such quadruples. For each of these quadruples, we check if the involved shortest paths intersect, and compute the corresponding branching events. Since the length of each such path is, whp, $O(\frac{n}{\rho_{i+1}} \cdot \log n)$, it follows that the entire computation can be carried out in $\tilde{O}(\frac{n^2}{\rho_{i+1}})$ expected time. Recall that $\rho_{i+1} = \tilde{\Omega}(n^{1/4})$, and, thus,

this running time is $\tilde{O}(n^{7/4})$. In $O(n \cdot P^2) = \tilde{O}(n^2)$ additional time, we construct the DPPRO $\mathcal{L}_h$ for the set of all pairs of h-landmarks. The total expected construction time is therefore dominated by the APSP computation, i.e., it is $\tilde{O}(m \cdot n)$.

Theorem 6.6 can be generalized to provide a tradeoff between oracle's parameters. Specifically, when aiming at size $O(n^{1+1/k} \cdot \log k)$, one can set $\rho_0 = n^{\frac{1}{4} \cdot \frac{k+1}{k}}$, $\frac{\rho_{i+1}^4}{\rho_i^3} = n^{1+1/k}$, i.e.,

$$\rho_i = n^{(1-(3/4)^{i+1})\frac{k+1}{k}}.$$

As a result, we have $\rho_h = n$ for $h \geq \log_{4/3}(k+1) - 1$. Hence, for $h = \lceil \log_{4/3}(k+1) \rceil - 1$, all vertices are h-level landmarks. The stretch becomes $O(7^{\log_{4/3}(k+1)}) = O(k^{\log_{4/3} 7}) \approx O(k^{6.76})$. Each of the $O(\log k)$ DPPROs requires now size $O(n^{1+1/k})$, i.e., the overall size is $O(n^{1+1/k} \cdot \log k)$, and the query time is $O(\log k)$.

Corollary 6.7. *Our path-reporting oracle, parameterized by $k = 1, 2, \ldots, O(\log n)$, uses expected space $O(n^{1+1/k} \cdot \log k)$, provides stretch $O(k^{\log_{4/3} 7})$, and has query time $O(\log k)$.*

This oracle improves previous bounds for all points in the size range between $\Omega(n \log \log n)$ and $o(n \log n)$. Recall that the oracle of Thorup and Zwick [2001a] always has size $\Omega(n \log n)$, and the oracle of Mendel and Naor [2006] is not path-reporting. The only previously existing path-reporting oracle that can be that sparse is the oracle of Elkin et al. [2014], which, for a parameter t, has size $O(n \cdot t \cdot \log_n w_{max})$, stretch $O(\sqrt{t} \cdot n^{\frac{2}{\sqrt{t}}})$, and query time $O(\log t + \log \log_n w_{max})$. Next, we compare our oracle with that of Elkin et al. [2014] in a number of points on the tradeoff curve.

For size $O(n \log \log n)$, our oracle has stretch $O(\log^{6.76} n)$ and query time $O(\log \log n)$, while the oracle of Elkin et al. [2014] with size $O(n(\log \log n + \log \log_n w_{max}))$ has stretch $2^{O(\frac{\log n}{\sqrt{\log \log n}})}$ and query time $O(\log^{(3)} n + \log \log_n w_{max})$. For size $O(n\sqrt{\log n})$, our oracle provides stretch $O((\frac{\log n}{\log \log n})^{6.76})$ and has query time $O(\log \log n)$. The oracle of Elkin et al. [2014] can have size $O(n(\sqrt{\log n} + \log \log_n w_{max}))$, stretch $2^{O(\log^{3/4} n)}$, and query time $O(\log \log n + \log \log_n w_{max})$. In other words, for size $O(n \log^{\delta} n)$, for any constant $\delta > 0$, our oracle is strictly better than that of Elkin et al. [2014], and its stretch is exponentially better than the stretch of Elkin et al. [2014].

Finally, we generalize the path-reporting oracle for sparse graphs (Corollary 6.4) so that it will provide a tradeoff between size and stretch.

Theorem 6.8. *For any parameters $k = 1, 2, \ldots$ and $\zeta > 0$, our path-reporting oracle for sparse graphs provides stretch $O(k^{\log_{4/3} 7})$, has expected size $O(n^{1+\zeta} \cdot \log k)$ in addition to the size of the input graph G, and has query time $\tilde{O}(\frac{m}{n} \cdot n^{\frac{1-(k-1)\zeta}{k}})$.*

Proof. We set $\rho_0 = n^{\frac{1}{4}(1+\zeta)}$, $\frac{\rho_{i+1}^4}{\rho_i^3} = n^{1+\zeta}$, i.e.,

$$\rho_i = n^{(1-(3/4)^{i+1})(1+\zeta)}.$$

To get query time $\tilde{O}(\frac{m}{n} \cdot n^{\epsilon})$, for some $\epsilon > 0$, we set the number of levels h to be the smallest integer that satisfies

$$\rho_h = n^{(1-(3/4)^{h+1})(1+\zeta)} \geq n^{1-\epsilon}.$$

This gives rise to $h \geq \log_{4/3} \frac{1+\zeta}{\zeta+\epsilon} - 1$. Hence, we set $h = \lceil \log_{4/3} \frac{1+\zeta}{\zeta+\epsilon} \rceil - 1$. (Observe that for $\zeta = 1/k$ and $\epsilon = 0$ we get $h = \lceil \log_{4/3}(k+1) \rceil - 1$, exactly as in Corollary 6.7. Indeed,

when $\epsilon = 0$, this oracle does not need to store the input graph G, and thus, it applies to general, and not only to sparse, graphs.)

Hence, the resulting oracle has expected size $O(n^{1+\zeta} \cdot \log \frac{1+\zeta}{\zeta+\epsilon})$, provides stretch $O((\frac{1+\zeta}{\zeta+\epsilon})^{\log_{4/3} 7})$, and has query time $\tilde{O}(\frac{m}{n} \cdot n^{\epsilon})$. We write $k = \frac{1+\zeta}{\zeta+\epsilon}$ and obtain $\epsilon = \frac{1-(k-1)\zeta}{k}$. In terms of ζ and k, the oracle has stretch $O(k^{\log_{4/3} 7})$, expected size $O(n^{1+\zeta} \cdot \log k)$, and query time $\tilde{O}(\frac{m}{n} \cdot n^{\frac{1-(k-1)\zeta}{k}})$. □

6.5. Spanner-Based Oracles

While the query time of our oracle $\tilde{\Lambda}$ is close to optimal (there is an additive slack of $O(\log \log n)$), its space requirement $O(n \log \log n)$ is slightly suboptimal, and also its stretch requirement is $O(\log^{\log_{4/3} 7} n)$, instead of the desired $O(\log n)$. Next, we argue that one can get an optimal space $O(n)$ and optimal stretch $O(\log n)$, at the expense of increasing the query time to $O(n^{\epsilon})$, for an arbitrarily small constant $\epsilon > 0$.

Given an n-vertex weighted graph $G = (V, E, \omega)$, we start with constructing an $O(\log n)$-spanner $G' = (V, H, \omega)$ of G with $O(n)$ edges. (See Althöfer et al. [1990]; a faster algorithm was given in Roditty et al. [2005]. For unweighted graphs a linear-time construction can be found in Peleg and Schäffer [1989], and a linear-time construction with optimal stretch-space tradeoff can be found in Halperin and Zwick [2000].) Then, we build the oracle Λ_h for the spanner G'. The space required by the oracle is (by Corollary 6.4) $O(n)$, plus the space required to store the spanner G', i.e., also $O(n)$. Hence, the total space required for this spanner-based oracle is $O(n)$. Its stretch is the product of the stretch of the oracle, i.e., at most $t^{\log_{4/3} 7}$, with $t = (4/3)^h$ for an integer h, and the stretch of the spanner, i.e., $O(\log n)$. Hence, the oracle's stretch is $O(t^{\log_{4/3} 7} \cdot \log n)$. The oracle reports paths in $G' = (V, H)$, but since $H \subseteq E$, these paths belong to G as well. Observe also that the query time of the spanner-based oracle is $\tilde{O}(n^{1/t} \cdot \frac{m'}{n})$, where $m' = |H|$ is the number of edges in the spanner. Since $m' = O(n)$, it follows that the query time is, whp, $\tilde{O}(n^{1/t})$. We remark also that the spanners produced by Althöfer et al. [1990] and Roditty et al. [2005] have constant arboricity, and thus, one does not really need the reduction described in Section 5 for this result.

THEOREM 6.9. *For any constant $\epsilon > 0$, the oracle obtained by invoking the oracle Λ_h with $h = \lceil \log_{4/3} \epsilon^{-1} \rceil$ from Corollary 6.4 on a linear-size $O(\log n)$-spanner is a path-reporting oracle with space $O(n)$, stretch $O(\log n)$, and query time $O(n^{\epsilon})$.*

Generally, we can use an $O(k)$-spanner, $\frac{\log n}{\log \log n} \le k \le \log n$ with $O(n^{1+1/k})$ edges. As a result, we obtain a path-reporting distance oracle with space $O(n^{1+1/k})$, stretch $O(k)$, and query time $O(n^{\epsilon+1/k}) = O(n^{\epsilon+o(1)})$.

Observe that Theorem 6.9 exhibits an optimal (up to constant factors) tradeoff between the stretch and the oracle size in the range $\frac{\log n}{\log \log n} \le k \le \log n$. The only known oracle that exhibits this tradeoff is due to Mendel and Naor [2006]. However, the oracle of Mendel and Naor [2006] is not path-reporting, while our oracle is.

The construction time of this oracle consists of the time required to build the $O(\log n)$-spanner (which is $\tilde{O}(n^2)$ [Roditty et al. 2005]) and the construction time of the oracle Λ_h in G' (which is also $\tilde{O}(n^2)$, because G' has $O(n)$ edges). Hence, its overall construction time is $\tilde{O}(n^2)$.

In the context of unweighted graphs, the same idea of invoking our oracle from Corollary 6.4 on a spanner can be used in conjunction with $(1+\epsilon, \beta)$-spanners. Given an unweighted n-vertex graph $G = (V, E)$, let $G' = (V, H)$ be its $(1+\delta, \beta)$-spanner, $\beta = \beta(\delta, k) = (\frac{\log k}{\delta})^{O(\log k)}$, with $|H| = O(\beta \cdot n^{1+1/k})$ edges, for a pair of parameters

$\delta > 0$, $k = 1, 2, \ldots$. (Such a construction was devised in Elkin and Peleg [2001].) For the sake of the following application, one can set $\delta = 1$. Invoke the distance oracle from Corollary 6.4 with a parameter t on top of this spanner. We obtain a path-reporting distance oracle with space $O(\beta n^{1+1/k})$ (whp). Its stretch is $(O(t^{\log_{4/3} 7}), \beta = \beta(t,k))$, $\beta(t,k) = O(t^{\log_{4/3} 7} \cdot \beta(1,k)) = t^{\log_{4/3} 7} \cdot k^{O(\log\log k)}$, and its query time is $\tilde{O}(n^{1/t+1/k})$, whp. As long as $t = o(k^{\frac{1}{\log_{4/3} 7}})$, the multiplicative stretch is $o(k)$, the additive stretch is still $\beta(k) = k^{O(\log\log k)}$, while the space is $O(\beta n^{1+1/k})$. In particular, one can have query time $n^{O(k^{-\frac{1}{\log_{4/3} 7+\eta}})}$, for an arbitrarily small constant $\eta > 0$, stretch $(o(k), k^{O(\log\log k)})$, and space $O(k^{O(\log\log k)} n^{1+1/k})$.

Another variant of this construction has a higher query time $O(n^{\epsilon})$, for some arbitrarily small constant $\epsilon > 0$, but its multiplicative stretch is $O(1)$. We just set t to be a large fixed constant and consider $k \gg t^{\log_{4/3} 7}$. Then, the query time is $O(n^{\epsilon})$, whp, $(\epsilon = t^{-1})$, stretch is $(O(1), poly(1/\epsilon) \cdot k^{O(\log\log k)})$, and space $O(\beta \cdot n^{1+1/k})$.

THEOREM 6.10. *For any unweighted undirected n-vertex graph G, any arbitrarily small constant $\epsilon > 0$ and any parameter $k = 1, 2, \ldots$, our path-reporting distance oracle has query time $O(n^{\epsilon})$ (whp), stretch $(O(1), \beta(k))$ and space $O(\beta(k) \cdot n^{1+1/k})$ (whp), where $\beta(k) = k^{O(\log\log k)}$. Another variant of this oracle has query time $n^{O(k^{-\frac{1}{\log_{4/3} 7+\eta}})}$ whp, for an arbitrarily small constant $\eta > 0$, stretch $(o(k), k^{O(\log\log k)})$, and space $O(k^{O(\log\log k)} \cdot n^{1+1/k})$, whp.*

To our knowledge, these are the first distance oracles whose tradeoff between multiplicative stretch and space is better than the classical tradeoff, i.e., $2k-1$ versus $O(n^{1+1/k})$. Naturally, we pay by having an additive stretch. By lower bounds from Thorup and Zwick [2001a], an additive stretch of $\Omega(k)$ is inevitable for such distance oracles.

One can also use a $(5+\epsilon, k^{O(1)})$-spanner with $O(n^{1+1/k})$ edges from Pettie [2009] instead of $(1+\epsilon, (\frac{\log k}{\epsilon})^{O(\log k)})$-spanner with $(\frac{\log k}{\epsilon})^{O(\log k)} n^{1+1/k}$ edges from Elkin and Peleg [2001] for our distance oracle. As a result, the oracle's space bound decreases to $O(n^{1+1/k})$, its additive stretch becomes polynomial in k, but the multiplicative stretch grows by a factor of $5+\epsilon$. In general, any construction of (α, β)-spanners with size $O(S \cdot n)$ can be plugged in our oracle. The resulting oracle will have stretch $(t^{\log_{4/3} 7} \cdot \alpha, t^{\log_{4/3} 7} \cdot \beta)$, size $O(Sn + n \cdot \log t)$, and query time $O(S \cdot n^{1/t})$.

The construction time of this oracle is the time needed to construct the $(1+\epsilon, \beta)$-spanner G', plus the construction of Λ_h on G'. The construction time of Elkin and Peleg [2001] is $O(n^{2+1/k})$. The construction time of the oracle Λ_h on G' is $\tilde{O}(m' \cdot n')$, where $m' = O(\beta \cdot n^{1+1/k})$ is the number of edges in G', and $n' = n$ is the number of vertices in G'. Hence, the overall construction time in this case is $O(\beta(k) \cdot n^{2+1/k}) = k^{O(\log\log k)} n^{2+1/k}$.

7. LOWER BOUNDS

In this section, we argue that one cannot expect to obtain distance labeling or routing schemes (see Section 2 for their definitions) with properties analogous to those of our distance oracles (given by Theorem 6.10 and Corollary 6.5). We also employ lower bounds of Sommer et al. [2009] to show that a distance oracle with stretch $(O(1), \beta(k))$ and space $O(\beta(k) \cdot n^{1+1/k})$ for unweighted n-vertex graphs (like the distance oracle given by Theorem 6.10) must have query time $\Omega(k)$.

7.1. Distance Labeling and Routing

We start with discussing distance labeling schemes.

THEOREM 7.1. *Any distance labeling scheme for general unweighted graphs that provides stretch $(t, t \cdot \beta(k))$, for a pair of parameters t, k, and a fixed function $\beta(\cdot)$, requires labels of size $\Omega(n^{\frac{1}{2t+5}})$.*

PROOF. Suppose for contradiction that there were a distance labeling scheme $\mathcal{D}$ for unweighted n-vertex graphs with maximum label size $O(n^{\frac{1}{2t+5}})$ and stretch $(t, t \cdot \beta(k))$, for some fixed function $\beta(\cdot)$, and any parameter k. Consider an infinite family of n-vertex unweighted graphs $G_n = (V, E_n)$ with girth at least $t+2$ and $|E_n| = \Theta(n^{1+\frac{1}{t+2}})$. (Such a family can be easily constructed by probabilistic method; see, e.g., Bollobas [1998], Theorem 3.7(a). Denser extremal graphs can be found in Lubotsky et al. [1988] and Lazebnik and Ustimenko [1995].) There are $2^{\Theta(n^{1+\frac{1}{t+2}})}$ different subgraphs of each G_n. To achieve stretch t, one would need $2^{\Theta(n^{1+\frac{1}{t+2}})}$ distinct encodings for these graphs, i.e., the total label size for this task is $\Omega(n^{1+\frac{1}{t+2}})$, and the maximum individual label size is $\Omega(n^{\frac{1}{t+2}})$. (See e.g., Thorup and Zwick [2001a], Chapter 5, for this lower bound.)

Replace every edge of $G = G_n$ by a path of length $10t \cdot \beta(k)$, consisting of new vertices. The new graph G'_n has $N = O(n^{1+\frac{1}{t+2}} \cdot t \cdot \beta(k))$ vertices. Invoke the distance labeling scheme $\mathcal{D}$ on G'_n. For a pair of original vertices u, v (vertices of G_n), the distance between them in G'_n is $d'(u, v) = 10t\beta(k) \cdot d_G(u, v)$. Given their labels $\varphi(u)$ and $\varphi(v)$, the labeling scheme $\mathcal{D}$ provides us with an estimate $\delta(\varphi(u), \varphi(v))$ of the distance between them in G'_n which satisfies:

$$\delta(\varphi(u), \varphi(v)) \leq t \cdot d'(u, v) + t \cdot \beta(k) = (10t\beta(k) \cdot d_G(u, v)) \cdot t + t \cdot \beta(k).$$

On the other hand, a path of length $d_G(u, v) \cdot t + 1$ in G between u and v translates into a path of length at most

$$10t \cdot \beta(k)(d_G(u, v) \cdot t + 1) = 10t^2\beta(k)d_G(u, v) + 10t\beta(k)$$

between them in G'_n. Hence, the estimate provided by $\mathcal{D}$ corresponds to a path between u and v of length at most $d_G(u, v) \cdot t$ in G_n, i.e., via $\mathcal{D}$ we obtain a t-approximate distance labeling scheme for G_n.

The maximum label size used by $\mathcal{D}$ is

$$O(N^{\frac{1}{2t+5}}) = O((n^{\frac{t+3}{t+2}} \cdot t \cdot \beta(k))^{\frac{1}{2t+5}}) = O(n^{\frac{t+3}{(t+2)(2t+5)}} \cdot (\beta(k))^{\frac{1}{2t+5}}).$$

However, by the above argument, this label size must be $\Omega(n^{\frac{1}{t+2}})$. Note that

$$n^{\frac{t+3}{(t+2)(2t+5)}}(\beta(k))^{\frac{1}{2t+5}} < n^{\frac{1}{t+2}},$$

as long as $\beta(k) < n$. This condition holds for any $k = O(\log n)$ and subexponential function $\beta(\cdot)$. (Recall that in all relevant upper bounds for spanners/distance oracles/distance labeling/routing schemes, it is always the case that $k = O(\log n)$ and $\beta(\cdot)$ is at most a quasi-polynomial function of k. Moreover, an additive stretch of $\Omega(t \cdot n)$ is obviously meaningless in the context of unweighted graphs.) Hence, this is a contradiction, and there can be no distance labeling scheme for unweighted graphs with label size $O(n^{\frac{1}{2t+5}})$ and stretch $(t, t \cdot \beta(k))$, for any parameter k. □

The same argument clearly applies to routing schemes as well. The only difference is that one needs to use lower bounds on the tradeoff between space and multiplicative stretch for routing due to Peleg and Upfal [1989], Thorup and Zwick [2001b], and Abraham et al. [2006], instead of analogous lower bounds of Thorup and Zwick [2001a] for distance labeling.

To summarize, while Theorem 6.10 provides a distance oracle with stretch $(t, t \cdot \beta(k))$ and *average* space per vertex of $O(\beta(k) \cdot n^{1/k})$ for $k \gg t^{\log_{4/3} 7}$, for distance labeling or routing one needs at least $n^{\Omega(1/t)}$ space per vertex to achieve the same stretch guarantee.

Similarly, one cannot have a distance labeling scheme for sparse graphs (graphs $G = (V, E)$ with $O(n^{1+1/k})$ edges, for some $k \geq 1$) with maximum label size $O(n^{1/k})$ and stretch $O(t)$, for a parameter $t \ll k$.[7] A distance labeling scheme, as above, requires maximum label size of $n^{\Omega(1/t)}$, as otherwise one would get a distance labeling with stretch $(t, t \cdot \text{poly}(k))$ for general graphs with maximum label size $n^{o(1/t)}$, contradiction.

7.2. Distance Oracles, Cell-Probe Model

Next, we argue that in the cell-probe model of computation (cf., Milterson [1999]), any distance oracle with size and stretch like in Theorem 6.10 (i.e., size $O(n^{1+1/k})$ and stretch $(O(1), \beta(k))$, for a fixed function $\beta(\cdot)$) must have query time $\Omega(k)$. We rely on the following lower bound of Sommer et al. [2009].

THEOREM 7.2 (SOMMER ET AL. [2009]). *A distance oracle with stretch t using query time q requires space $\mathcal{S} \geq n^{1+\frac{c}{t \cdot q}} / \log n$ in the cell-probe model with w-bit cells, even on unweighted undirected graphs with maximum degree at most $(t \cdot q \cdot w)^{O(1)}$, where $t = o(\frac{\log n}{\log w + \log\log n})$, and c is a positive constant.*

Suppose, for a contradiction, that there exists a distance oracle with stretch $(t, t \cdot \beta(k))$, for a pair of parameters $t \ll k$ and a fixed function $\beta(\cdot)$, with space at most $n^{1+\frac{c/2}{t \cdot q}} / \log n$ (and query time q) for general unweighted graphs.

Let $G = (V, E)$ be an n-vertex unweighted graph with maximum degree at most $(t \cdot q \cdot w)^{O(1)}$, and let G' be the graph obtained from G by replacing each edge of G by a path of length $10t \cdot \beta(k)$. The graph G' has $N \leq (t \cdot q \cdot w)^{O(1)} \cdot \beta(k) \cdot n$ vertices, and an oracle with stretch $(t, t \cdot \beta(k))$ for G' can be used also as a stretch-t oracle for G. The size of this oracle is, by our assumption, at most

$$\frac{(n \cdot (t \cdot q \cdot w)^{O(1)} \cdot \beta(k))^{1+\frac{c/2}{t \cdot q}}}{\log N} < \frac{n^{1+\frac{c/2}{t \cdot q}}}{\log n} \cdot ((t \cdot q \cdot w)^{O(1)} \beta(k))^{1+\frac{c/2}{t \cdot q}}.$$

As long as $((t \cdot q \cdot w)^{O(1)} \cdot \beta(k))^{1+\frac{c/2}{t \cdot q}} < n^{\frac{c/2}{t \cdot q}}$, i.e., as long as

$$((t \cdot q \cdot w)^{O(1)} \cdot \beta(k))^{\frac{2}{c} t \cdot q + 1} < n, \tag{2}$$

we have a contradiction to Theorem 7.2. (As the oracle uses less than $n^{1+\frac{c}{t \cdot q}} / \log n$ space and has stretch t and query time q.)

For k being at most a mildly growing function of n (specifically, $k \leq \log^{\zeta} n$, $\zeta < 1/2$), $t = o(k)$, $q \leq k$, $w = O(\log n)$, and $\beta(\cdot)$ being a polynomial (or even a quasi-polynomial) function, the condition (2) holds. Hence, in this range of parameters, any distance oracle for unweighted graphs with stretch $(t, t \cdot \beta(k))$ and query time q requires space $\mathcal{S} \geq n^{1+\frac{c/2}{t \cdot q}} / \log n$ in the cell-probe model with w-bit cells, assuming $t = o(\frac{\log n}{\log w + \log\log n})$.

So, if this oracle uses $\mathcal{S} = O(n^{1+1/k} \cdot \beta(k))$ space, then it holds that $n^{1+1/k} \cdot \log n \cdot \beta(k) \geq n^{1+\frac{c/2}{t \cdot q}}$, i.e.,

$$1 + 1/k + \frac{\log\log n + \log \beta(k)}{\log n} \geq 1 + \frac{c/2}{t \cdot q},$$

and so $q = \Omega(k/t)$.

[7] Recall that by Corollary 6.5, a path-reporting distance oracle of total size $O(n^{1+1/k})$ with stretch $O(t)$ and query time $O(n^{\frac{1}{t^c}+\frac{1}{k}} + |\Pi(u, v)|)$ (for a query u, v; the constant c is given by $c = \log_7 4/3$) does exist.

We summarize this lower bound in the next theorem.

THEOREM 7.3. *Let $k \leq \log^{\zeta} n$, for any constant $\zeta < 1/2$, $t = o(k)$, $w = O(\log n)$, and $\beta(\cdot)$ being a polynomial or a quasi-polynomial function. In the cell-probe model with w-bit cells, any distance oracle for general unweighted undirected n-vertex graphs with space $O(\beta(k) \cdot n^{1+1/k})$ and stretch $(t, t \cdot \beta(k))$ has query time $q = \Omega(k/t) = \Omega(k)$.*

Theorem 7.3 states that in contrast to distance oracles with multiplicative stretch which can have constant query time (see Mendel and Naor [2006] and Chechik [2014]), a distance oracle with stretch $(O(1), \beta(k))$ (like the one given by our Theorem 6.10) must have query time $\Omega(k)$.

7.3. Distance Oracles, A Conditional Lower Bound

In this section, we argue that even relatively mild improvement of Theorem 6.10 would give rise to improved distance oracles for sparse unweighted graphs.

Recall that Agarwal et al. [2011] devised a not path-reporting distance oracle for sparse graphs, which for parameters $t = 1, 2, \ldots$ and $\epsilon > 0$, provides stretch $4t - 1$, has expected size $O(t \cdot n^{(1+1/t)(1-\epsilon)}$, in addition to the size of the input graph, and has query time $\tilde{O}(\frac{m}{n} \cdot n^{\epsilon})$. Set $\zeta = \frac{t+1}{t}(1-\epsilon) - 1$, i.e., $\epsilon = \frac{1-\zeta t}{t+1}$. The size becomes $O(t \cdot n^{1+\zeta})$, and the query time is $\tilde{O}(\frac{m}{n} \cdot n^{\frac{1-\zeta t}{t+1}})$. In the next theorem, we consider a special case of this oracle for graphs with $m = O(n)$ edges.

THEOREM 7.4 (AGARWAL ET AL. [2011]). *For any parameters $t = 1, 2, \ldots, \zeta > 0$, the not path-reporting distance oracle of Agarwal et al. [2011] for sparse graphs ($m = O(n)$) has stretch $4t - 1$, expected size $O(t \cdot n^{1+\zeta})$, and query time $\tilde{O}(n^{\frac{1-\zeta t}{t+1}})$.*

See also our Theorem 6.8 for a path-reporting counterpart of this result.

We will now argue that a path-reporting distance oracle for unweighted general graphs, which provides a mixed multiplicative-additive stretch, and has significantly better parameters than those given Theorem 6.10, can be used to devise a path-reporting oracle for sparse unweighted graphs that outperforms the (not path-reporting) oracle of Agarwal et al. [2011] from Theorem 7.4. (Note, however, that Theorem 7.4 applies to weighted graphs as well. On the other hand, no better bound than the one given by Theorem 7.4 for sparse unweighted graphs is known.) We view this as an indication that obtaining a path-reporting oracle with mixed stretch for general unweighted graphs with parameters similar to those given in Theorem 6.10, but with *polylogarithmic* query time, might be hard.

Specifically, our Theorem 6.10 provides a distance oracle with stretch $(O(t^{\log_{4/3} 7}), O(t^{\log_{4/3} 7} \cdot \beta(k)))$, size $O(n^{1+1/k})$, and query $O(n^{1/t})$ for general unweighted graphs, where $\beta(\cdot)$ is a polynomial function.

THEOREM 7.5. *We are given a positive integer n, a pair of positive integer parameters t, k, and an at most exponential function $\beta(\cdot)$, such that $t \leq \frac{k}{10 \cdot \log(\beta(\log n))}$, $k \leq \log n$. Suppose that there exists a path-reporting oracle D for general unweighted n-vertex graphs with stretch $(t, t \cdot \beta(k))$, size $O(n^{1+1/k})$, and query time $O(n^{1/8t})$. Then, there exists a path-reporting oracle D' for sparse ($m = O(n)$) unweighted n-vertex graphs, which outperforms the not path-reporting oracle of Agarwal et al. [2011] given in Theorem 7.4.*

PROOF. Consider an unweighted sparse graph $G = (V, E)$, $m = |E| = O(n)$. Replace every edge $e \in E$ with a path of length $10t \cdot \beta(k)$. We get a sparse unweighted G' with $N = 10t \cdot \beta(k) \cdot n$ vertices. Invoke the oracle D on G'. It has size $O(N^{1+1/k}) = O((t \cdot \beta(k))^{1+1/k} \cdot n^{1+1/k}) = O(t \cdot \beta(k) \cdot n^{1+1/k})$. Also, it provides pure multiplicative stretch at most t for G. Since the additive term $\beta(k) \leq O(n)$, the query time is $O((t \cdot \beta(k))^{1/8t} \cdot n^{1/8t}) = O(\beta(k)^{1/8t} \cdot n^{1/8t}) = O(n^{1/4t})$. The size of the oracle is $O(n^{1+\zeta})$, for $\zeta = 1/k + \log_n(t \cdot \beta(k))$.

So the resulting path-reporting oracle D' for sparse unweighted graphs has size $O(n^{1+\zeta})$, for ζ as above, stretch t, and query time $O(n^{1/4t})$. On the other hand, the not path-reporting oracle $\tilde{D}$ of Agarwal et al. [2011], Theorem 7.4, gives (for $\zeta = 1/k + \log_n(t \cdot \beta(k))$) size $O(n^{1+\zeta})$, stretch $4t-1$, and query time $\tilde{O}(n^{\frac{1-\zeta t}{t+1}})$. Since

$$1-\zeta t \;\geq\; 1-(1/k+\log_n(t\beta(k)))t \geq 1-t/k-2\cdot\frac{\log(\beta(\log n))}{\log n}\cdot t$$
$$\geq\; 9/10-2\cdot\frac{\log(\beta(\log n))}{\log n}\cdot\frac{k}{10\cdot\log(\beta(\log n))}\geq 7/10,$$

we have that this query time is at least $\Omega(n^{\frac{7}{10}\frac{1}{t+1}})$. Hence, D' strictly outperforms $\tilde{D}$. □

In particular, if the query time of D is polylogarithmic in n, then the query time of D' is polylogarithmic in n as well, in a sharp contrast to the polynomial in n query time in Theorem 7.4.

APPENDIX

A. MISSING PROOFS

In this section, we provide proofs of Lemmas 6.1 and 6.2.

Proof of Lemma 6.1. Suppose for contradiction that there exists a pair $(x,y)\in\mathcal{P}_{i+1}$ such that the pairs $(u,v),(x,y)$ participate in a branching event β, and such that either $x\notin \mathrm{Ball}_{i+1}(u)$ or $y\notin \mathrm{Ball}_{i+1}(u)$. Then, $\beta=(\Pi(u,v),\Pi(x,y),z)$, where $\Pi(u,v)$ (respectively, $\Pi(x,y)$) is a shortest path between u and v (respectively, between x and y), and z is a node at which these two paths branch. Since $(x,y)\in\mathcal{P}_{i+1}$, it follows that either $y\in\mathcal{B}^{1/3}_{i+1}(x)$ or $x\in\mathcal{B}^{1/3}_{i+1}(y)$. Without loss of generality, suppose that $y\in\mathcal{B}^{1/3}_{i+1}(x)$.

The proof splits into two cases. In the first case, we assume that $x\notin\mathrm{Ball}_{i+1}(u)$, and in the second, we assume that $y\notin\mathrm{Ball}_{i+1}(u)$. (Note that roles of x and y are not symmetric.) In both cases, we reach a contradiction.

We start with the case $x\notin\mathrm{Ball}_{i+1}(u)$. Observe that $d_G(x,z)\leq d_G(x,y)<\frac{1}{3}\cdot r_{i+1}(x)$ and $d_G(u,z)\leq d_G(u,v)<\frac{1}{3}\cdot r_{i+1}(u)$. Denote $\delta=d_G(u,u^{(i+1)})=r_{i+1}(u)$, where $u^{(i+1)}=\ell_{i+1}(u)$. Denote also $\delta'=d_G(u,x)$. Observe that $r_{i+1}(x)\leq d_G(x,u^{(i+1)})\leq\delta+\delta'$, and also (since $x\notin\mathrm{Ball}_{i+1}(u)$) $\delta'=d_G(u,x)\geq\delta=r_{i+1}(u)$. Then,

$$d_G(u,z)+d_G(z,x)<\frac{1}{3}\cdot r_{i+1}(u)+\frac{1}{3}\cdot r_{i+1}(x)\leq\frac{\delta}{3}+\frac{1}{3}\cdot(\delta+\delta')\leq\delta'=d_G(u,x).$$

Hence, $d_G(u,z)+d_G(z,x)<d_G(u,x)$, contradicting the triangle inequality.

We are now left with the case that $x\in\mathrm{Ball}_{i+1}(u)$, but $y\notin\mathrm{Ball}_{i+1}(u)$. Then, $d_G(y,z)\leq d_G(x,y)<\frac{1}{3}\cdot r_{i+1}(x)$. Also, $d_G(u,z)\leq d_G(u,v)<\frac{1}{3}\cdot r_{i+1}(u)$. In addition, $r_{i+1}(x)\leq d_G(x,u^{(i+1)})\leq d_G(x,u)+r_{i+1}(u)\leq 2\delta$. (Note that $d_G(x,u)\leq\delta=r_{i+1}(u)$, because $x\in\mathrm{Ball}_{i+1}(u)$.) Hence,

$$d_G(u,z)+d_G(z,y)<\frac{1}{3}\cdot(r_{i+1}(u)+r_{i+1}(x))\leq\frac{1}{3}\cdot(\delta+2\delta)=\delta\leq d_G(u,y).$$

(The last inequality is because, by an assumption, $y\notin\mathrm{Ball}_{i+1}(u)$.) This is, however, again a contradiction to the triangle inequality. □

Proof of Lemma 6.2. Recall that (see Coppersmith and Elkin [2005], Lemma 7.5) each pair $(u,v),(x,y)$ may produce at most two branching events. Hence, next we focus on providing an upper bound on the number of intersecting pairs of paths $\Pi(u,v)$, $\Pi(x,y)$ for $(u,v),(x,y)\in\mathcal{P}_i$.

By the previous lemma, for a pair (u, v), (x, y) to create a branching event, there must be one of these four vertices (without loss of generality we call it u) such that the three other vertices belong to $\mathrm{Ball}_{i+1}(u)$. Hence, the number of intersecting pairs as above is at most (a constant factor multiplied by) the number of quadruples (u, v, x, y) with $v, x, y \in \mathrm{Ball}_{i+1}(u)$. For a fixed i-landmark u, the number of vertices in its $(i+1)$st ball $\mathrm{Ball}_{i+1}(u^{(i)})$ is, whp, $O(\frac{n}{\rho_{i+1}} \cdot \log n)$. (This random variable is distributed geometrically with the parameter $p = \frac{\rho_{i+1}}{n}$.) Each of the vertices in $\mathrm{Ball}_{i+1}(u)$ has probability $\frac{\rho_i}{n}$ to belong to L_i, independently of other vertices. Hence, by Chernoff's bound, whp, there are $\frac{\rho_i}{n} \cdot O(\frac{n}{\rho_{i+1}} \cdot \log n) = O(\frac{\rho_i}{\rho_{i+1}} \cdot \log n)$ i-landmarks in $\mathrm{Ball}_{i+1}(u)$. (We select the constant c hidden by the O-notation in $O(\frac{n}{\rho_{i+1}} \cdot \log n)$ to be sufficiently large. Then, the expectation is $c \cdot \frac{\rho_i}{\rho_{i+1}} \cdot \log n \geq c \cdot \log n$. Hence, the Chernoff's bound applies with high probability.)

Hence, the number of triples v, x, y of i-landmarks in $\mathrm{Ball}_{i+1}(u)$ is, whp, $O(\frac{\rho_i^3}{\rho_{i+1}^3} \cdot \log^3 n)$. The number of i-landmarks u is, by the Chernoff's bound, whp, $O(\rho_i)$. Hence, the number of quadruples as above is, whp, at most

$$O(\rho_i) \cdot O\left(\frac{\rho_i^3}{\rho_{i+1}^3} \cdot \log^3 n\right) = O\left(\frac{\rho_i^4}{\rho_{i+1}^3} \cdot \log^3 n\right).$$

Also, the number of pairs $|\mathcal{P}_i|$ is at most the number of i-landmarks (whp, it is $O(\rho_i)$) multiplied by the maximum number of i-landmarks in an $(i+1)$-level ball $\mathrm{Ball}_{i+1}(u)$ (whp, it is $O(\frac{\rho_i}{\rho_{i+1}} \cdot \log n)$), i.e., $|\mathcal{P}_i| = O(\frac{\rho_i^2}{\rho_{i+1}} \cdot \log n)$.

Next, we argue that the expected number of quadruples (u, v, x, y) of i-landmarks such that $v, x, y \in \mathrm{Ball}_{i+1}(u)$ is $O(\frac{\rho_i^4}{\rho_{i+1}^3})$ and that $\mathbb{E}(|\mathcal{P}_i|) = O(\frac{\rho_i^2}{\rho_{i+1}})$.

For a fixed vertex u, write $X(u) = I(\{u \in L_i\}) \cdot Y(u)$, where $Y(u)$ is the number of triples of distinct i-landmarks different from u which belong to $\mathrm{Ball}_{i+1}(u)$, and $I(\{u \in L_i\})$ is the indicator random variable of the event $\{u \in L_i\}$. (Note that the ball is defined even if $u \notin L_i$.) Observe that the random variables $I(\{u \in L_i\})$ and $Y(u)$ are independent, and thus,

$$\mathbb{E}(X(u)) = \mathbb{E}(I(\{u \in L_i\})) \cdot \mathbb{E}(Y(u)) = \frac{\rho_i}{n} \cdot \mathbb{E}(Y(u)).$$

Let $\sigma = (v_1, v_2, \ldots, v_{n-1})$ be the sequence of vertices ordered by the non-decreasing distance from u. (They appear in the order in which the Dijkstra algorithm initiated at u discovers them.) For $k = 3, 4, \ldots, n-1$, denote by $\mathcal{J}_k$ the random variable which is equal to 0 if v_{k+1} is not the first vertex in σ which belongs to L_{i+1}. If v_{k+1} is the first vertex as above then $\mathcal{J}_k$ is equal to the number of triples $v_{j_1}, v_{j_2}, v_{j_3}$, $1 \leq j_1 < j_2 < j_3 \leq k$ such that $v_{j_1}, v_{j_2}, v_{j_3} \in L_i$. Also, for each quadruple $1 \leq j_1 < j_2 < j_3 < j_4 \leq n-1$ of indices, define $J(j_1, j_2, j_3, j_4)$ to be the indicator random variable of the event that $v_{j_1}, v_{j_2}, v_{j_3} \in L_i$, $v_{j_4} \in L_{i+1}$, and for each j, $1 \leq j < j_4$, the vertex v_j is not an $(i+1)$-landmark. Observe that

$$\mathbb{E}(J(j_1, j_2, j_3, j_4)) = \left(\frac{\rho_i}{n}\right)^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^{j_4 - 1} \cdot \frac{\rho_{i+1}}{n}.$$

Also,

$$\mathbb{E}(\mathcal{J}_k) = \sum_{1 \leq j_1 < j_2 < j_3 \leq k} \mathbb{E}(J(j_1, j_2, j_3, k+1)) = \binom{k}{3} \left(\frac{\rho_i}{n}\right)^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k \cdot \frac{\rho_{i+1}}{n}.$$

Note that $Y(u) = \sum_{k=3}^{n-2} \mathcal{J}_k$, and so

$$\mathbb{E}(Y(u)) \leq \sum_{k=3}^{\infty} \binom{k}{3} \left(\frac{\rho_i}{n}\right)^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k \cdot \frac{\rho_{i+1}}{n}.$$

Denote $A = 10\frac{n}{\rho_{i+1}}$. For $k \leq A$, since $(1 - \frac{\rho_{i+1}}{n})^k = O(1)$, it follows that

$$\sum_{k=3}^{A} \binom{k}{3} \left(\frac{\rho_i}{n}\right)^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k \cdot \frac{\rho_{i+1}}{n} = O\left(\frac{\rho_i^3 \cdot \rho_{i+1}}{n^4}\right) \sum_{k=3}^{A} k^3 = O\left(\frac{\rho_i^3}{\rho_{i+1}^3}\right).$$

Also,

$$\sum_{k=A+1}^{\infty} \binom{k}{3} \left(\frac{\rho_i}{n}\right)^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k \cdot \frac{\rho_{i+1}}{n} \leq O\left(\frac{\rho_i^3 \cdot \rho_{i+1}}{n^4}\right) \cdot \sum_{k=A+1}^{\infty} k^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k.$$

Denote $\gamma = 1 - \rho_{i+1}/n$. Then,

$$\sum_{k=A+1}^{\infty} k^3 \gamma^k \leq \frac{\mathrm{d}^3}{\mathrm{d}\gamma^3} \sum_{k=A+1}^{\infty} \gamma^{k+3} \leq \frac{\mathrm{d}^3}{\mathrm{d}\gamma^3} \frac{1}{1-\gamma} = \frac{6}{(1-\gamma)^4} = O\left(\left(\frac{n}{\rho_{i+1}}\right)^4\right).$$

Hence,

$$\sum_{k=A+1}^{\infty} \binom{k}{3} \left(\frac{\rho_i}{n}\right)^3 \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k \cdot \frac{\rho_{i+1}}{n} = O\left(\frac{\rho_i^3 \cdot \rho_{i+1}}{n^4}\right) \cdot O\left(\left(\frac{n}{\rho_{i+1}}\right)^4\right) = O\left(\frac{\rho_i^3}{\rho_{i+1}^3}\right),$$

and so $\mathbb{E}(Y(u)) = O(\frac{\rho_i^3}{\rho_{i+1}^3})$. Hence, $\mathbb{E}(X(u)) = \frac{\rho_i}{n} \cdot \mathbb{E}(Y(u)) = O(\frac{\rho_i^4}{\rho_{i+1}^3} \cdot \frac{1}{n})$.

Finally, the overall expected number of quadruples (u, v, x, y) of i-landmarks such that $v, x, y \in \mathrm{Ball}_{i+1}(u)$ is, by linearity of expectation, at most $\sum_{v \in V} \mathbb{E}(X(u)) = O(\frac{\rho_i^4}{\rho_{i+1}^3})$.

A similar argument provides an upper bound of $O(\frac{\rho_i^2}{\rho_{i+1}})$ on the expected number of pairs $|\mathcal{P}_i|$. We shortly sketch it below.

For a vertex u, let $X'(u) = I(\{u \in L_i\}) \cdot Y'(u)$, where $Y'(u)$ is the number of i-landmarks which belong to $\mathrm{Ball}_{i+1}(u)$. Clearly, $\mathbb{E}(I(\{u \in L_i\})) = \rho_i/n$, and the two random variables ($I(\{u \in L_i\})$ and $Y'(u)$) are independent. For every integer $k \geq 1$, let $\mathcal{J}'_k$ be a random variable which is equal to 0 if v_{k+1} is not the first vertex in σ which belongs to L_{i+1}. Otherwise, it is the number of i-landmarks among $v_1, v_2, \ldots, v_k$. For integer j_1, j_2, $1 \leq j_1 < j_2 \leq n-1$, let $J'(j_1, j_2)$ be the indicator random variable of the event that $v_{j_1} \in L_i$, $v_{j_2} \in L_{i+1}$, and for every $j < j_2$, it holds that $v_j \notin L_{i+1}$. Then,

$$\mathbb{E}(J'(j_1, j_2)) = \frac{\rho_i}{n} \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^{j_2 - 1} \cdot \frac{\rho_{i+1}}{n}.$$

Hence,

$$\mathbb{E}(\mathcal{J}'_k) = \sum_{1 \leq j_1 \leq k} \mathbb{E}(J'(j_1, k+1)) = \frac{\rho_i \cdot \rho_{i+1}}{n^2} \cdot k \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k,$$

and

$$\mathbb{E}(Y'(u)) \leq \sum_{k=1}^{\infty} \mathbb{E}(\mathcal{J}'_k) = \frac{\rho_i \cdot \rho_{i+1}}{n^2} \cdot \sum_{k=1}^{\infty} k \cdot \left(1 - \frac{\rho_{i+1}}{n}\right)^k.$$

Write $A = 10\frac{n}{\rho_{i+1}}$, and

$$\sum_{k=1}^{\infty} k\left(1-\frac{\rho_{i+1}}{n}\right)^k = \sum_{k=1}^{A} k\left(1-\frac{\rho_{i+1}}{n}\right)^k + \sum_{k>A} k\left(1-\frac{\rho_{i+1}}{n}\right)^k.$$

Each term of the first sum is $O(1)$, and thus, the first sum is at most $O(A^2) = O(n^2/\rho_{i+1}^2)$. The second sum is at most $\frac{d}{d\gamma}\sum_{k>A}\gamma^{k+1} \le \frac{d}{d\gamma}\frac{1}{1-\gamma} = O(n^2/\rho_{i+1}^2)$ as well. Hence,

$$\mathbb{E}(Y'(u)) = \frac{\rho_i \cdot \rho_{i+1}}{n^2} \cdot O\left(\frac{n^2}{\rho_{i+1}^2}\right) = O\left(\frac{\rho_i}{\rho_{i+1}}\right).$$

Hence, $\mathbb{E}(X'(u)) = O(\rho_i^2/(\rho_{i+1}n))$, and by linearity of expectation we conclude that $\mathbb{E}(|\mathcal{P}_i|) \le \sum_{u\in V}\mathbb{E}(X'(u)) = O(\rho_i^2/\rho_{i+1})$. □

ACKNOWLEDGMENTS

The first-named author wishes to thank Christian Wulff-Nilsen for posing the problem of devising path-reporting oracles of size $o(n \log n)$ for general graphs. A conversation with him triggered the research on Elkin et al. [2014], and this line of research was continued in this article. The first-named author is also grateful Ofer Neiman for helpful discussions, Elad Verbin for explaining to him the lower bounds from Sommer et al. [2009], and an anonymous referee for helping to improve the presentation of this article.

REFERENCES

Ittai Abraham and Cyril Gavoille. 2011. On approximate distance labels and routing schemes with affine stretch. In *DISC*. 404–415.

Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. 2006. On space-stretch trade-offs: Lower bounds. In *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures*. 207–216. DOI:http://dx.doi.org/10.1145/1148109.1148143

Ittai Abraham and Ofer Neiman. 2012. Using petal-decompositions to build a low stretch spanning tree. In *STOC*. 395–406.

R. Agarwal. 2014a. Personal communication. (2014).

Rachit Agarwal. 2014b. The space-stretch-time tradeoff in distance oracles. In *Algorithms - ESA 2014 (Lecture Notes in Computer Science)*, Andreas S. Schulz and Dorothea Wagner (Eds.), Vol. 8737. Springer Berlin, 49–60. DOI:http://dx.doi.org/10.1007/978-3-662-44777-2_5

Rachit Agarwal and Philip Brighten Godfrey. 2013. Distance oracles for stretch less than 2. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*. 526–538. DOI:http://dx.doi.org/10.1137/1.9781611973105.38

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

Ingo Althöfer, Gautam Das, David P. Dobkin, and Deborah Joseph. 1990. Generating sparse spanners for weighted graphs. In *SWAT*. 26–37.

Yair Bartal. 1996. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*. 184–193.

Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. 2008. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *ICALP (1)*. 609–621.

Surender Baswana and Telikepalli Kavitha. 2006. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS*. 591–602.

Surender Baswana and Sandeep Sen. 2006. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms* 2, 4 (2006), 557–577.

B. Bollobas. 1998. *Extremal Graph Theory*. Springer-Verlag.

Shiri Chechik. 2014. Approximate distance oracles with constant query time. In *Proceedings of the Symposium on Theory of Computing (STOC'14)*. 654–663. DOI:http://dx.doi.org/10.1145/2591796.2591801

D. Coppersmith and M. Elkin. 2005. Sparse source-wise and pair-wise distance preservers. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*. 660–669.

M. Elkin. 2001. Computing almost shortest paths. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing*. 53–62.

Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. 2005. Lower-stretch spanning trees. In *STOC*. 494–503.

Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. 2014. Space-efficient path-reporting distance oracles. *CoRR* abs/1410.0768 (2014). http://arxiv.org/abs/1410.0768.

M. Elkin and D. Peleg. 2001. Spanner constructions for general graphs. In *Proceedings of the 33th ACM Symposium on Theory of Computing*. 173–182.

Michael Elkin and Seth Pettie. 2015. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 805–821. DOI:http://dx.doi.org/10.1137/1.9781611973730.55

S. Halperin and U. Zwick. 2000. Unpublished manuscript. (2000).

F. Lazebnik and V. A. Ustimenko. 1995. Explicit construction of graphs with an arbitrary large girth and of large size. *Discrete Applied Mathematics* 60, 1–3 (1995), 275–284.

A. Lubotsky, R. Phillips, and P. Sarnak. 1988. Ramanujan graphs. *Combinatorica* 8 (1988), 261–277.

J. Matousek. 1996. On the distortion required for embedding finite metric spaces into normed spaces. *Israeli J. Math.* 93 (1996), 333–344.

Manor Mendel and Assaf Naor. 2006. Ramsey partitions and proximity data structures. In *FOCS*. 109–118.

Peter Bro Milterson. 1999. Cell probe complexity - A survey. In *Invited Talk and Paper in Advances in Data Structures (Preconference Workshop of FSTTCS)*.

Mihai Pătraşcu and Liam Roditty. 2010. Distance oracles beyond the Thorup-Zwick bound. In *FOCS*. 815–823.

Mihai Pătraşcu, Liam Roditty, and Mikkel Thorup. 2012. A new infinity of distance oracles for sparse graphs. In *FOCS*. 738–747.

David Peleg. 2000. Proximity-preserving labeling schemes. *Journal of Graph Theory* 33, 3 (2000), 167–176. DOI:http://dx.doi.org/10.1002/(SICI)1097-0118(200003)33:3<167::AID-JGT7>3.0.CO;2-5

D. Peleg and A. Schäffer. 1989. Graph spanners. *Journal of Graph Theory* 13 (1989), 99–116.

D. Peleg and E. Upfal. 1989. A tradeoff between size and efficiency for routing tables. *Journal of the ACM* 36 (1989), 510–530.

Seth Pettie. 2009. Low distortion spanners. *ACM Transactions on Algorithms* 6, 1 (2009).

Ely Porat and Liam Roditty. 2013. Preprocess, set, query! *Algorithmica* 67, 4 (2013), 516–528.

Liam Roditty. 2015. Distance oracles for sparse graphs. In *Encyclopedia of Algorithms*. DOI:http://dx.doi.org/10.1007/978-3-642-27848-8_571-1

Liam Roditty, Mikkel Thorup, and Uri Zwick. 2005. Deterministic constructions of approximate distance oracles and spanners. In *ICALP*. 261–272.

Christian Sommer, Elad Verbin, and Wei Yu. 2009. Distance oracles for sparse graphs. In *FOCS*. 703–712.

M. Thorup and U. Zwick. 2001a. Approximate distance oracles. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*. 183–192.

M. Thorup and U. Zwick. 2001b. Compact routing schemes. In *Proceedings of the 13th Symposium on Parallelism in Algorithms and Architectures*. 1–10.

M. Thorup and U. Zwick. 2006. Spanners and emulators with sublinear distance errors. In *Proceedings of the Symposium on Discrete Algorithms*. 802–809.

Christian Wulff-Nilsen. 2012. Approximate distance oracles with improved preprocessing time. In *SODA*. 202–208.

Received July 2015; revised January 2016; accepted January 2016