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

Michael Elkin [*] Seth Pettie [†]

October 6, 2014

Abstract

In a seminal paper [27] for any n-vertex undirected graph $G = (V, E)$ and a parameter $k = 1, 2, \ldots$, Thorup and Zwick constructed a distance oracle of size $O(kn^{1+1/k})$ which upon a query (u, v) 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 [27] is $O(k)$ (in addition to the length of the returned path), and it was subsequently improved to $O(1)$ [29, 11]. A major drawback of the oracle of [27] is that its space is $\Omega(n \cdot \log n)$. Mendel and Naor [18] 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 paper 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 $\epsilon > 0$. In particular, for $k = \log n$ our oracle provides logarithmic stretch using linear size. Another variant of our oracle has linear size, 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 $\beta()$, space $O(n^{1+1/k} \cdot \beta)$, 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 [16]. 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.

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 [27]. See their paper also for a very persuasive motivation of this natural notion. In their seminal paper Thorup and Zwick [27] devised a path-reporting distance oracle (henceforth, TZ oracle). The TZ oracle

[*]Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel, elkinm@cs.bgu.ac.il

This research has been supported by the Israeli Academy of Science, grant 593/11, and by the Binational Science Foundation, grant 2008390. This research has also been supported by the Lynn and William Frankel Center for Computer Science. A part of this research was performed while visiting the 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. 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.

[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 $O(|\Pi|)$.

with a parameter $k = 1, 2, \ldots$ has size $O(k \cdot n^{1+1/k})$, stretch $2k - 1$ and query time $O(k)$. As argued in [27], this tradeoff between size and stretch is essentially optimal for $k \le \frac{\log n}{\log\log n}$, as Erdos' girth conjecture implies that $\Omega(n^{1+1/k})$ space is required for any k. Note, however, that $k \cdot n^{1+1/k} = \Omega(n \cdot \log n)$, and Thorup and Zwick [27] left it open if one can obtain meaningful distance oracles of *linear* size (or, more generally, size $o(n \log n)$).

A partial answer to this question was provided by Mendel and Naor [18], who devised a distance oracle with size $O(n^{1+1/k})$, stretch $O(k)$ and query time $O(1)$. Alas, their distance oracle is *inherently not path-reporting*. Specifically, the oracle of [18] stores a collection of $O(k \cdot n^{1/k})$ hierarchically-separated trees (henceforth, HSTs; see [7] 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 [14, 2]). However, each of this 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, Neiman and Wulff-Nilsen [15]. For a parameter $t \ge 1$ their oracle uses space $O(n \cdot t)$ and provides stretch $O(\sqrt{t} \cdot n^{2/\sqrt{t}})$ for weighted graphs. 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 graphs their oracle exhibits roughly the same behavior. For a parameter $\epsilon > 0$ it uses space $O(n \cdot t/\epsilon)$ and provides stretch $O(t \cdot n^{1/t}(t + n^{\epsilon/t}))$.

The distance oracles of [15] 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 [27, 18]. 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 paper we answer this question in the affirmative. For any k, $\frac{\log n}{\log\log n} \le k \le \log n$, and any arbitrarily small constant $\epsilon > 0$, our path-reporting distance oracle has stretch $O(k)$, size $O(n^{1+1/k})$ and query time $O(n^{\epsilon})$. (When $\epsilon > 0$ is subconstant the stretch becomes $O(k) \cdot (1/\epsilon)^{O(1)}$.) Hence our oracle achieves an optimal up to constant factors tradeoff between size and stretch in the range $\frac{\log n}{\log\log n} \le k \le \log n$, 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 $\Omega(n \cdot \log n)$ [27, 29, 11] or have stretch $n^{\Omega(1)}$ [15]. (The query time of the TZ oracle was recently improved to $O(1)$ in [29, 11].) 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 $O(n^{1+1/k})$ to $O(n^{\epsilon})$, for an arbitrarily small constant $\epsilon > 0$.

We also can trade between the stretch and the query time. Specifically, a variant 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 $O(\log\log n)$. For a comparison, the path-reporting distance oracle of [15] with this stretch uses space $\Omega(n \cdot \frac{\log n}{\log\log n})$ and has query time $O(\log\log n \cdot \log_n w_{max})$.

We also remark that using a super-constant (but not trivial) query time is a common place by now in the distance oracles literature. In particular, this is the case in the oracles of Porat and Roditty [23], Agarwal and Godfrey [4] and of Agarwal et al. [5].

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 $\alpha \ge 1, \beta \ge 0$ if for any query (u, v) it constructs a path $\Pi(u, v)$ of length $\delta(u, v)$ which satisfies $d_G(u, v) \le \delta(u, v) \le \alpha \cdot d_G(u, v) + \beta$. 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 $d_H(u, v) \le \alpha \cdot d_G(u, v) + \beta$.

This notion was introduced in [16], 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

[2]For a parameter $t \ge 1$, $G' = (V, H)$ is a *t-spanner* of a graph $G = (V, E)$, $H \subseteq E$, if $d_H(u, v) \le t \cdot d_G(u, v)$.

similar properties were devised in [13, 28, 22].

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 [19]. Pătraşcu and Roditty [19] devised a distance oracle with stretch $(2, 1)$ and size $O(n^{5/3})$, and query time $O(1)$. Abraham and Gavoille [1] generalized the result of [19] to devise a distance oracle with stretch $(2k-2, 1)$ and space $\tilde{O}(n^{1+(2/(2k-1))})$. (The query time in [1] is unspecified.)

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 [16] in the context of *spanners*, where the multiplicative stretch becomes as small as $1 + \epsilon$, for an arbitrarily small $\epsilon > 0$.) In this paper we devise the first distance oracles that do achieve such a tradeoff. Specifically, our path-reporting distance oracle has stretch $(O(1), \beta(k))$, space $O(\beta(k) \cdot n^{1+1/k})$, $\beta(k) = k^{O(\log\log k)}$, 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 $(o(k), \beta(k))$, space $O(\beta(k) \cdot n^{1+1/k})$ 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, even if Erdős' girth conjecture is wrong, still it is known that (when the additive stretch is 0) the tradeoff between multiplicative stretch and size cannot be below the curve given by stretch k and space $O(n^{1+1/k})$. Our results, like the results of [16] 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.

1.3 Distance Oracles for Sparse Graphs

A central ingredient in all our distance oracles is a new path-reporting distance oracle for graphs with $O(n)$ edges. The most relevant result in this context is the paper by Agarwal et al. [5]. In this paper the authors devised a (not path-reporting)[3] linear-size distance oracle which given a parameter $k = 1, 2, \ldots$ provides distance estimates with stretch $4k - 1$, uses linear space and has time $O(n^{1/(k+1)})$. (Their result is, in fact, more general than this. We provide this form of their result to facilitate the comparison.) In this paper we present the first path-reporting linear-size distance oracle for this range of parameters. Specifically, our linear-size oracle (see Corollary 6.1) has stretch $O(k^{\log_{4/3} 7})$ and query time $O(n^{1/k})$, for any constant parameter k of the form $k = (4/3)^h$, $h = 1, 2, \ldots$.

1.4 Distance-Preserving Distance Oracle

In [12] 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 paper 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.

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. In addition to what we have already surveyed there are probe-complexity lower bounds by Sommer et al. [26]. There is an important line of work by Pătraşcu et al. [20, 19] on oracles with rational stretch. Finally, Baswana and Sen [10], Baswana and Kavitha [9] and Baswana et al. [8] improved the preprocessing time of the TZ oracle.

1.6 Structure of the Paper

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. [5] 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

[3]It was erroneously claimed in [5] that all their distance oracles are path-reporting. While their distance oracles with stretch smaller than 3 are path-reporting (albeit their space requirement is superlinear), this is not the case for their oracles with stretch $4k - 1$, $k \geq 1$ [3].

from [5]) is described in Section 5. Then (in Section 6) we devise a much more elaborate multi-level path-reporting oracle for sparse graphs. The oracle of [5] 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 7) 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 for general graphs with query time n^{ϵ}, and oracles with hybrid multiplicative-additive stretch for unweighted graphs.

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 $\tilde{O}(f(n)) = O(f(n) polylog(f(n)))$ and $\tilde{\Omega}(f(n) = \Omega(f(n)/polylog(f(n)))$.

3 A Distance-Preserving Path-Reporting Oracle

Consider a directed weighted n-vertex graph $G = (V, E, \omega)$. (The result given in this section applies to both directed and undirected graphs. However, our other distance oracles apply only to undirected graphs.) Let $Pairs \subseteq \binom{V}{2}$ be a subset of ordered pairs of vertices. We denote its cardinality by $P = |Pairs|$. 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 $Paths = \{\Pi_{u,v} \mid (u,v) \in Pairs\}$ 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 (i.e., e.g., both traverse first x and then y), then they necessarily share the entire subpath between x and y. It is argued in [12] that this property can be easily achieved.

We will need the following definitions from [12].

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 $pred_{\Pi}(u_i)$, is the vertex u_{i-1} (assuming that $i \ge 1$; otherwise it is defined as $NULL$), and the *successor* of u_i in Π, denoted $succ_{\Pi}(u_i)$, is the vertex u_{i+1} (again, assuming that $i \le h-1$; otherwise it is $NULL$).

Definition [12]: A *branching event* (Π, Π', x) is a triple with $\Pi, \Pi' \in Paths$ being two distinct paths and $x \in V(\Pi) \cap V(\Pi')$ be a vertex that belongs to both paths and such that $\{pred_{\Pi}(x), succ_{\Pi}(x)\} \ne \{pred_{\Pi'}(x), succ_{\Pi'}(x)\}$. 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 [12], 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 $|\mathcal{B}| \le |Paths|^2 = P^2$. Our oracle will keep $O(1)$ data for each vertex and $O(1)$ data for each branching event, i.e., $O(n + |\mathcal{B}| + P)$ 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.

The query algorithm proceeds as follows. Given a pair $(x,y) \in Pairs$, we find the first edge (x, x') of the path $\Pi_{x,y}$, and "move" to x'. Then 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 check 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 $\Pi(x') \ne \Pi(x,y)$) 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.)

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''.

In all cases the procedure then recurses with x''. 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 summarize this section with the following theorem.

Theorem 3.1. *Given a directed weighted graph $G = (V, E, \omega)$ and a collection Pairs $\subseteq \binom{V}{2}$ of pairs of vertices, our distance-preserving path-reporting oracle (shortly,* 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 $P = |Pairs|$.*

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 $\tilde{O}(m \cdot \min\{P, n\}) + O(n \cdot P^2)$.

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[4], the 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 $\lambda = \frac{m}{n}$.

[4]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 $c \geq 2$.

Our basic oracle employs just one level of sampled vertices, which we (following the terminology of [5]) 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 $\Pi(v)$ 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 [27] with a parameter k. (Henceforth we will call it the *TZ oracle*.) One can also use here Mendel-Naor's oracle [18], 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 [11, 29] one can have here 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 $O(k \cdot |L|^{1+1/k})$.) Fix an oracle $\mathcal{H}$ as above for $|H| = O(k \cdot |L|^{1+1/k})$. Whp such an $\mathcal{H}$ will 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 distance preserving oracle (shortly, DPPRO) from Section 3 on the graph G and set $Pairs = H$. 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 $\epsilon > 0$,

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

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 $Ball(v) = \{x \mid d_G(v, x) < d_G(v, \ell(v))\}$ 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 $u \in Ball(v)$ and if $v \in Ball(u)$. To test if $u \in Ball(v)$ 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 $u \in Ball(v)$, and return the (exact) shortest path between them. Otherwise we conclude that $u \notin Ball(v)$. Analogously, the algorithm tests if $v \in Ball(u)$.

Henceforth we assume that $u \notin Ball(v)$ and $v \notin Ball(u)$, 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

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

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

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

Next, we analyze the running time of the query algorithm. First, consider the step that tests if $v \in Ball(u)$ and if $u \in Ball(v)$. 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 $i \le j$. (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 all not sampled and the vertex u_t is sampled. Hence X is distributed geometrically with the parameter $p = \rho/n$. Hence

$$(4.1) \quad \mathbb{E}(X) \ = \ \sum_{t=1}^{n-1} (1-p)^t \cdot p \cdot t \ \le \ \frac{1}{p} \ = \ \frac{n}{\rho} \ .$$

Also, obviously for any positive constant c, $\mathbb{P}(X > \frac{n}{\rho}c\ln n) \le (1-\rho/n)^{(n/\rho)c\ln n} \le n^{-c}$, 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 $n' \le n$ vertices induces $O(n'\cdot\lambda)$ edges. Hence Dijkstra algorithm traverses expected $O(\frac{n}{\rho}\lambda)$ edges, 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[5] 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 $O(|\Pi'|+k)$. Finally, 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 $|\Pi'| \le |\hat{\Pi}|$. 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. (Observe that the additive term of $O(k)$ is dominated by $O(\frac{n}{\rho}\cdot\lambda)$. Specifically, we will be using $\rho \le n/\log n$, and $k \le O(\log n)$.) 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 one can build a DPPRO $\mathcal{H}$ for all pairs of landmarks. (In this case $\rho = n^{1/4}$, $|L| = O(\rho)$, $|Pairs| = |\binom{L}{2}| = O(\rho^2) = O(\sqrt{n})$, and so the size of the oracle $\mathcal{H}$ is $O(|Pairs|^2+n) = O(n)$.)

One can build the forest of SPTs rooted at the landmarks in $\tilde{O}(m)$ time. In additional $O(m\cdot\rho) = O(k\cdot m\cdot n^{1/2-\frac{1}{2k+2}})$ time one can construct the metric closure of L, i.e., the graph $\mathcal{L}$. This graph has $n'=\rho$ vertices and $m' \le \rho^2$ 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 7.1, our another application (Theorem 7.2) requires a more general result. We remark that our extension is based on the reduction of Agarwal et al. [5] of the distance oracle problem in general graphs to the same problem in bounded-degree graphs. Our reduction is somewhat more general than the one from [5], as it also applies to path-reporting distance oracles. We provide our extension for the sake of completeness.

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

[5]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.

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 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

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

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 $v \in Ball(u)$ and if $u \in Ball(v)$ 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 $y^{(2)}$, etc., 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

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

Hence

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

as well. Thus

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

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 $O(\log n)$.

Corollary 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.*

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. [5], 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 [22]. We will later (Section 7) build upon these oracles to construct oracles for dense graphs as well. 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 $p_i = \frac{\rho_i}{n}$, $\rho_1 > \rho_2 > \ldots > \rho_h$. 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 $r_i(v) = d_G(v, \ell_i(v))$. Following [22], let $\mathcal{B}_i^{1/3}(v) = \{u \mid d_G(v,u) < \frac{1}{3} r_i(v)\}$ denote the *ith one-third-ball* of v, and $Ball_i(v) = \{u \mid d_G(v,u) < r_i(v)\}$ denote the *ith ball* of v.

For each vertex $v \in V$ we keep a 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 $u^{(i)}$ we keep a shortest path between $u^{(i)}$ and its closest $(i+1)$st landmark $\ell_{i+1}(u^{(i)}) = u^{(i+1)}$. 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 $u^{(h)}, v^{(h)}$ of h-landmarks this oracle returns a shortest path $\Pi(u^{(h)}, v^{(h)})$ between them in time proportional to the number of edges in this path, i.e., $O(|\Pi(u^{(h)}, v^{(h)})|)$. 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 6.2). The size of the oracle is $O(n + |Branch_i|)$, 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 $|Branch_i| = O(n)$, 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 is given a pair $u = u^{(0)}, v = v^{(0)}$ of vertices. The algorithm starts with testing if $u \in Ball_1(v)$ and if $v \in Ball_1(u)$. 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).

Observe that by Equation (4.1), the expected size of $Ball_1(v)$ and of $Ball_1(u)$ 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 $v \in Ball_1(u)$ or that $u \in Ball_1(v)$ then it has found the shortest path between u and v. In this case the algorithm returns this path. 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.)

For $j < h$, the query algorithm of our oracle Λ_h then queries the hash table $\mathcal{H}_j$ whether the pair $(u^{(j)}, v^{(j)}) \in \mathcal{P}_j$. 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

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

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 prerecorded paths $\Pi(u^{(j)}, u^{(j+1)})$ and $\Pi(v^{(j)}, v^{(j+1)})$, and invokes itself recursively on the pair $(u^{(j+1)}, v^{(j+1)})$. (Recall that for each index j, $1 \le j \le h-1$, 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 $(u^{(h)}, v^{(h)})$. The query returns the shortest path between them in time $O(|\Pi(u^{(h)}, v^{(h)})|)$. 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

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

where $1 \le j \le h$ 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 $0 \le j \le h-1$ at least one of the subpaths $\Pi(u^{(j)}, u^{(j+1)}), \Pi(v^{(j)}, v^{(j+1)})$ is not empty then $h \le |\Pi(u, v)|$, 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 $u^{(j)} \ne u^{(j+1)}$ and $v^{(j)} \ne v^{(j+1)}$, for every j. To do this one just defines $u^{(j+1)} = \ell(u^{(j)})$ to be the closest $(j+1)$-level landmark to $u^{(j)}$, which is *different* from $u^{(j)}$. 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 6.2.)

6.3 The Stretch Analysis Recall that in the case that $v \in Ball_1(u)$ or $u \in Ball_1(v)$ our algorithm returns the exact shortest path between $u = u^{(0)}$ and $v = v^{(0)}$. Hence we next consider the situation when $v \notin Ball_1(u)$ and $u \notin Ball_1(v)$. 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 $d_G(u^{(0)}, u^{(1)}), d_G(v^{(0)}, v^{(1)}) \le d$, and so

$$\begin{aligned}d_G(u^{(1)}, v^{(1)}) &\le d_G(u^{(1)}, u^{(0)}) + d_G(u^{(0)}, v^{(0)}) \\ &+ d_G(v^{(0)}, v^{(1)}) \le 3 \cdot d.\end{aligned}$$

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

$$\begin{aligned}&d_G(u^{(0)}, u^{(1)}) + d_G(u^{(1)}, v^{(1)}) + d_G(v^{(1)}, v^{(0)}) \\ \le\ & d + 3 \cdot d + d \ = \ 5 \cdot d\ .\end{aligned}$$

More generally, suppose the query algorithm reached the j-level landmarks $u^{(j)}, v^{(j)}$, for some j, $1 \le j \le h-1$, and suppose that $(u^{(j)}, v^{(j)}) \notin \mathcal{P}_j$. This means that $v^{(j)} \notin \mathcal{B}^{1/3}_{j+1}(u^{(j)})$ and $u^{(j)} \notin \mathcal{B}^{1/3}_{j+1}(v^{(j)})$. By definition of the one-third-ball it follows that

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

and

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

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

Hence

$$\begin{aligned}d_G(u^{(j+1)}, v^{(j+1)}) &\le d_G(u^{(j+1)}, u^{(j)}) + d_G(u^{(j)}, v^{(j)}) \\ &+ d_G(v^{(j)}, v^{(j+1)}) \ \le\ 7 \cdot d_G(u^{(j)}, v^{(j)})\ .\end{aligned}$$

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

$$\begin{aligned}d_G(u^{(j)}, u^{(j+1)}) &\le 3 \cdot d_G(u^{(j)}, v^{(j)}) \ \le 3 \cdot d^{(j)} \\ &= 3^2 \cdot 7^{j-1} \cdot d\ .\end{aligned}$$

Hence the length of the path

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

returned by the algorithm is at most

$$d^{(0)} + 3 \cdot \left(\sum_{j=1}^{p-1} d^{(j)} \right) + d^{(p)} + 3 \cdot \left(\sum_{j=1}^{p-1} d^{(j)} \right) + d^{(0)} =$$

$$d \cdot \left(2 \cdot \left(1 + 3 \cdot \left(\sum_{j=1}^{p-1} 3 \cdot 7^{j-1} \right) \right) + 3 \cdot 7^{p-1} \right)$$

$$= \ d \cdot (6 \cdot 7^{p-1} - 1) \ .$$

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 $\rho_1 > \rho_2 > \ldots > \rho_h$ 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. (It is $\tilde{O}(\frac{n}{\rho_1}\lambda) + O(|\Pi|)$, where Π 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 $O(n + |Branch_i| + |\mathcal{P}_i|)$, where $Branch_i$ is the set of branching events for the set $\mathcal{P}_i$ of pairs of vertices. Next we argue that (whp) $|Branch_i| = O(n)$. Recall that the set $\mathcal{P}_i$ contains all pairs of i-landmarks $(u^{(i)}, v^{(i)})$ such that either $v^{(i)} \in \mathcal{B}_{i+1}^{1/3}(u^{(i)})$ or $u^{(i)} \in \mathcal{B}_{i+1}^{1/3}(v^{(i)})$.

The following two lemmas from [22] 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 $v \in \mathcal{B}_{i+1}^{1/3}(u)$. 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 $x, y \in Ball_{i+1}(u)$.*

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

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 $O\left(\frac{\rho_i^4}{\rho_{i+1}^3} \log^3 n\right) = O(n^{4\alpha_i - 3\alpha_{i+1}}) = O(n)$, and $O\left(\frac{\rho_i^2}{\rho_{i+1}}\right) = O(n^{2\alpha_i - \alpha_{i+1}}) = O(n^{1 - \frac{1}{2}(\frac{3}{4})^{h-i}})$. Hence by Lemma 6.2, for each $i \in [h-1]$, the oracle $\mathcal{D}_i$ requires expected space $O(n + |Branch_i| + |\mathcal{P}_i|) = O(n)$. 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 $\tilde{O}((n/\rho_1)\lambda) = \tilde{O}(n^{(3/4)^h} \cdot \lambda)$.

The reduction described in Section 5 enables us to extend these results to general m-edge n-vertex graphs. Specifically, the reduction reduces the distance oracle problem on such graphs to the same problem on graphs with bounded degree $\lambda = m/n$. Graphs with maximum degree λ have arboricity at most λ, and thus the reduction is applicable.

THEOREM 6.1. *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 graphs it is $O(n^{(3/4)^h} \cdot \frac{m}{n} \cdot \log^2 n)$.

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 $O(t^{\log_{4/3} 7}) \approx O(t^{6.76})$.

COROLLARY 6.1. *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 $O(t^{\log_{4/3} 7})$, 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 $v \in Ball_1(u)$ and if $u \in Ball_1(v)$. For unweighted graphs these tests require $O(\frac{n}{\rho_1}\lambda) = O(n^{(3/4)^h}\lambda)$ expected time.

COROLLARY 6.2. *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 $O(t^{\log_{4/3} 7})$, 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 $h = \log_{4/3}(\log n + 1)$. Then $\rho_1 = (2n)^{\alpha_1} = n$. 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 $v \in Ball_1(u)$ and if $u \in Ball_1(v)$. 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 $v \in Ball_1(u)$ and if $u \in Ball_1(v)$.) 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 6.1) $6 \cdot 7^{h-1} - 1 = O(\log^{\log_{4/3} 7} n)$.

THEOREM 6.2. *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 possibly 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 6.2 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 6.1 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 $O(|\mathcal{P}_i|) = O(\frac{\rho_i^2}{\rho_{i+1}} \cdot \log^2 n) = O(n)$. Then we produce all possible quadruples u, v, x, y with $v, x, y \in Ball_{i+1}(u) \cap L_i$, $u \in L_i$. 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 $\tilde{O}(m \cdot n)$.

7 Spanner-Based Oracles

While the query time of our oracle $\tilde{\Lambda}$ is essentially optimal (except maybe for an additive $\log\log n$ term), 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 [6]; a faster algorithm was given in [24]. For unweighted graphs a linear-time construction can be found in [21], and a linear-time construction with optimal stretch-space tradeoff can be found in [17].) Then we build the oracle Λ_h for the spanner G'. The space required by the oracle is (by Corollary 6.1) $O(n)$ (whp), 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., $O(t^{\log_{4/3} 7})$, 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 spanner has constant arboricity, and thus one does not really need the reduction described in Section 5 for this result.

Theorem 7.1. *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 6.1 on a linear-size $O(\log n)$-spanner is a path-reporting oracle with space $O(n)$ (whp), stretch $O(\log n)$, and query time $O(n^\epsilon)$.*

Generally, we can use an $O(k)$-spanner, $\frac{\log n}{\log \log n} \leq k \leq \log n$ with $O(n^{1+1/k})$ edges. As a result we obtain a path-reporting distance oracle with space $O(n^{1+1/k})$ (whp; in fact, the space is slightly better than that, i.e., $O(n^{1+1/k}) + S$, where $S = O(n)$ whp), stretch $O(k)$ and query time $O(n^{\epsilon + 1/k}) = O(n^{\epsilon + o(1)})$.

Observe that Theorem 7.1 exhibits an optimal (up to constant factors) tradeoff between the stretch and the oracle size in the range $\frac{\log n}{\log \log n} \leq k \leq \log n$. The only known oracle that exhibits this tradeoff is due to Mandel and Naor [18]. However, the oracle of [18] 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 $\tilde{O}(n^2)$ [25]) 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 6.1 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, $\beta = \beta(\delta, k) = \left(\frac{\log k}{\delta}\right)^{O(\log k)}$, 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 [16].) For the sake of the following application one can set $\delta = 1$. Invoke the distance oracle from Corollary 6.1 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 $n^{O\left(k^{-\frac{1}{\log_{4/3} 7 + \eta}}\right)}$, 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 $O(\beta \cdot n^{1+1/k})$, whp.

Theorem 7.2. *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 $n^{O\left(k^{-\frac{1}{\log_{4/3} 7 + \eta}}\right)}$ 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 [27], 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 [22] 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 [16] 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 $5 + \epsilon$. We remark that, 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 [16] 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 $O(\beta(k) \cdot n^{2+1/k}) = k^{O(\log \log k)} n^{2+1/k}$.

Acknowledgements

The first-named author wishes to thank Ofer Neiman and Christian Wulff-Nilsen for helpful discussions.

References

[1] I. Abraham and C. Gavoille. On approximate distance labels and routing schemes with affine stretch. In *DISC*, pages 404–415, 2011.

[2] I. Abraham and O. Neiman. Using petal-decompositions to build a low stretch spanning tree. In *STOC*, pages 395–406, 2012.

[3] R. Agarwal. Personal communication, 2014.

[4] R. Agarwal and P. B. Godfrey. Distance oracles for stretch less than 2. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 526–538, 2013.

[5] R. Agarwal, P. B. Godfrey, and S. Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM*, pages 1754–1762, 2011.

[6] I. Althöfer, G. Das, D. P. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In *SWAT*, pages 26–37, 1990.

[7] Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, pages 184–193, 1996.

[8] S. Baswana, A. Gaur, S. Sen, and J. Upadhyay. Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In *ICALP (1)*, pages 609–621, 2008.

[9] S. Baswana and T. Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *FOCS*, pages 591–602, 2006.

[10] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected $o(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.

[11] S. Chechik. Approximate distance oracles with constant query time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663, 2014.

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

[13] M. Elkin. Computing almost shortest paths. In *Proc. 20th ACM Symp. on Principles of Distributed Computing*, pages 53–62, 2001.

[14] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. In *STOC*, pages 494–503, 2005.

[15] M. Elkin, O. Neiman, and C. Wulff-Nilsen. Space-efficient path-reporting distance oracles. *CoRR*, abs/1410.0768, 2014.

[16] M. Elkin and D. Peleg. Spanner constructions for general graphs. In *Proc. of the 33th ACM Symp. on Theory of Computing*, pages 173–182, 2001.

[17] S. Halperin and U. Zwick. Inpublished manuscript, 2000.

[18] M. Mendel and A. Naor. Ramsey partitions and proximity data structures. In *FOCS*, pages 109–118, 2006.

[19] M. Pătraşcu and L. Roditty. Distance oracles beyond the Thorup-Zwick bound. In *FOCS*, pages 815–823, 2010.

[20] M. Pătraşcu , L. Roditty, and M. Thorup. A new infinity of distance oracles for sparse graphs. In *FOCS*, pages 738–747, 2012.

[21] D. Peleg and A. Schäffer. Graph spanners. *J. Graph Theory*, 13:99–116, 1989.

[22] S. Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1), 2009.

[23] E. Porat and L. Roditty. Preprocess, set, query! *Algorithmica*, 67(4):516–528, 2013.

[24] L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *ICALP*, pages 261–272, 2005.

[25] L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. In *Proc. 45th Annual IEEE Symp. on Foundations of Comp. Science*, pages 499–508, 2004.

[26] C. Sommer, E. Verbin, and W. Yu. Distance oracles for sparse graphs. In *FOCS*, pages 703–712, 2009.

[27] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. of the 33rd ACM Symp. on Theory of Computing*, pages 183–192, 2001.

[28] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. of Symp. on Discr. Algorithms*, pages 802–809, 2006.

[29] C. Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *SODA*, pages 202–208, 2012.

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 $x \notin Ball_{i+1}(u)$ or $y \notin Ball_{i+1}(u)$. 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 $x \in \mathcal{B}^{1/3}_{i+1}(x)$. 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 $x \notin Ball_{i+1}(u)$, and in the second we assume that $y \notin Ball_{i+1}(u)$. (Note that roles of x and y are not symmetric.) In both cases we reach a contradiction.

We start with the case $x \notin Ball_{i+1}(u)$. Observe that $d_G(x, z) \le d_G(x, y) < \frac{1}{3} \cdot r_{i+1}(x)$ and $d_G(u, z) \le d_G(u, v) < \frac{1}{3} \cdot r_{i+1}(u)$. 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 $r_{i+1}(x) \le d_G(x, u^{(i+1)}) \le \delta + \delta'$, and also (since $x \notin Ball_{i+1}(u)$) $\delta' = d_G(u, x) \ge \delta = r_{i+1}(u)$. Then

$$\begin{aligned} d_G(u, z) + d_G(z, x) &< \frac{1}{3} \cdot r_{i+1}(u) + \frac{1}{3} \cdot r_{i+1}(x) \\ &\le \frac{\delta}{3} + \frac{1}{3} \cdot (\delta + \delta') \\ &\le \delta' \ = \ d_G(u, x) \ . \end{aligned}$$

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 $x \in Ball_{i+1}(u)$, but $y \notin Ball_{i+1}(u)$. Then $d_G(y, z) \le d_G(x, y) < \frac{1}{3} \cdot r_{i+1}(x)$. Also, $d_G(u, z) \le d_G(u, v) < \frac{1}{3} \cdot r_{i+1}(u)$. In addition, $r_{i+1}(x) \le d_G(x, u^{(i+1)}) \le d_G(x, u) + r_{i+1}(u) \le 2\delta$. (Note that $d_G(x, u) \le \delta = r_{i+1}(u)$, because $x \in Ball_{i+1}(u)$.) Hence

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

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

Proof of Lemma 6.2: Recall that (see [12], 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 $Ball_{i+1}(u)$. 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 $v, x, y \in Ball_{i+1}(u)$. For a fixed i-landmark u, the number of vertices in its $(i+1)$st ball $Ball_{i+1}(u^{(i)})$ is, whp, $O\left(\frac{n}{\rho_{i+1}} \cdot \log n\right)$. (This random variable is distributed geometrically with the parameter $p = \frac{\rho_{i+1}}{n}$.) Each of the vertices in $Ball_{i+1}(u)$ has probability $\frac{\rho_i}{n}$ to belong to L_i, independently of other vertices. Hence, by Chernoff's bound, whp there are $\frac{\rho_i}{n} \cdot O\left(\frac{n}{\rho_{i+1}} \cdot \log n\right) = O\left(\frac{\rho_i}{\rho_{i+1}} \cdot \log n\right)$ i-landmarks in $Ball_{i+1}(u)$. (We select the constant c hidden by the O-notation in $O\left(\frac{n}{\rho_{i+1}} \cdot \log n\right)$ to be sufficiently large. Then the expectation is $c \cdot \frac{\rho_i}{\rho_{i+1}} \cdot \log n \ge c \cdot \log n$. Hence the Chernoff's bound applies with high probability.)

Hence the number of triples v, x, y of i-landmarks in $Ball_{i+1}(u)$ is, whp, $O\left(\frac{\rho_i^3}{\rho_{i+1}^3} \cdot \log^3 n\right)$. 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 $Ball_{i+1}(u)$ (whp, it is $O\left(\frac{\rho_i}{\rho_{i+1}} \cdot \log n\right)$), i.e., $|\mathcal{P}_i| = O\left(\frac{\rho_i^2}{\rho_{i+1}} \cdot \log n\right)$.

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

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 $Ball_{i+1}(u)$, 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 $v_{j_1}, v_{j_2}, v_{j_3}$, $1 \le j_1 < j_2 < j_3 \le k$ such that $v_{j_1}, v_{j_2}, v_{j_3} \in L_i$. Also, for each quadruple $1 \le j_1 < j_2 < j_3 < j_4 \le n-1$ 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, $1 \le j < j_4$, 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,

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

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

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

Denote $A = 10\frac{n}{\rho_{i+1}}$. For $k \le A$, we have $(1 - \frac{\rho_{i+1}}{n})^k = O(1)$, and so

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

Also,

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

The latter sum is at most

$$\begin{aligned}\int_A^{\infty} x^3 e^{-10x/A} dx &= (A/10)^4 \int_{10}^{\infty} y^3 e^{-y} dy \\ &= O(A^4) = O\left(\left(\frac{n}{\rho_{i+1}}\right)^4\right) .\end{aligned}$$

Hence

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

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 $v, x, y \in Ball_{i+1}(u)$ 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 $O\left(\frac{\rho_i^2}{\rho_{i+1}}\right)$ 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 $Ball_{i+1}(u)$. 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 $k \ge 1$, 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, $1 \le j_1 < j_2 \le n-1$, 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

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

and

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

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 the integral $\int_A^\infty x(1-(1/10A))^x dx = O(A^2)\int_1^\infty y \cdot e^{-y} dy = O(A^2) = O(n^2/\rho_{i+1}^2)$ 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 $\mathbb{E}(|\mathcal{P}_i|) \leq \sum_{u\in V} \mathbb{E}(X'(u)) = O(\rho_i^2/\rho_{i+1})$. ∎