Path-Reporting Distance Oracles with Linear Size

Ofer Neiman
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Idan Shabat
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Given an undirected weighted graph, an (approximate) distance oracle is a data structure that can (approximately) answer distance queries. A *Path-Reporting Distance Oracle*, or *PRDO*, is a distance oracle that must also return a path between the queried vertices. Given a graph on n vertices and an integer parameter $k \geq 1$, Thorup and Zwick [22] showed a PRDO with stretch $2k-1$, size $O(k \cdot n^{1+1/k})$ and query time $O(k)$ (for the query time of PRDOs, we omit the time needed to report the path itself). Subsequent works [20, 7, 8] improved the size to $O(n^{1+1/k})$ and the query time to $O(1)$. However, these improvements produce distance oracles which are not path-reporting. Several other works [12, 13] focused on small size PRDO for general graphs, but all known results on distance oracles with linear size suffer from polynomial stretch, polynomial query time, or not being path-reporting.

In this paper we devise the first linear size PRDO with poly-logarithmic stretch and low query time $O(\log \log n)$. More generally, for any integer $k \geq 1$, we obtain a PRDO with stretch at most $O(k^{4.82})$, size $O(n^{1+1/k})$, and query time $O(\log k)$. In addition, we can make the size of our PRDO as small as $n + o(n)$, at the cost of increasing the query time to poly-logarithmic. For unweighted graphs, we improve the stretch to $O(k^2)$.

We also consider *pairwise PRDO*, which is a PRDO that is only required to answer queries from a given set of pairs $\mathcal{P}$. An exact PRDO of size $O(n + |\mathcal{P}|^2)$ and constant query time was provided in [13]. In this work we dramatically improve the size, at the cost of slightly increasing the stretch. Specifically, given any $\epsilon > 0$, we devise a pairwise PRDO with stretch $1 + \epsilon$, constant query time, and near optimal size $n^{o(1)} \cdot (n + |\mathcal{P}|)$.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Graph Algorithms, Shortest Paths, Distance Oracles

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.36

Related Version *Full Version*: https://arxiv.org/abs/2405.14254

Funding Partially supported by the Lynn and William Frankel Center for Computer Sciences and ISF grant 970/21.

Acknowledgements The second-name author would like to thank Michael Elkin, who also suggested to the second-name author that the pairwise spanner of [19] might be extended to a pairwise PRDO, and that a PRDO for unweighted graphs with size $O\left(n^{1+\frac{1}{k}}\right)$ and stretch $O(k^2)$ can be achieved.

1 Introduction

Given an undirected weighted graph $G = (V, E)$ with n vertices and positive weights on the edges $w : E \to \mathbb{R}_+$, the distance $d_G(u, v)$ between two vertices $u, v \in V$ is the minimal weight of a path between them in G. For a parameter $\alpha \geq 1$, a *distance oracle* with stretch α is a data structure, that given a query for a pair of vertices (u, v), returns an estimated distance $\hat{d}(u, v)$ such that

$$d_G(u, v) \leq \hat{d}(u, v) \leq \alpha \cdot d_G(u, v) \ .$$

A *Path-Reporting Distance Oracle*, or *PRDO*, is a distance oracle that must also return a path in G of weight $\hat{d}(u, v)$ between the queried vertices u, v.

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).
Editor: Hans L. Bodlaender; Article No. 36; pp. 36:1–36:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Distance oracles have been the subject of extensive research in the last few decades. They are fundamental objects in Graph Algorithms, due to their both practical and theoretical usefulness. The main interest is in the triple tradeoff between the stretch of a distance oracle, its size, and its query time. In some cases, the preprocessing time, (that is, the time needed to construct the distance oracle) is also considered. Note that for every query, a PRDO must return a path P, thus the running time of the query algorithm is always in the general form of $O(q + |P|)$. We usually omit the term[1] $|P|$ from the query time and write only $O(q)$.

This work focuses on path-reporting distance oracles for general graphs. The path-reporting property is more appealing for certain applications that require navigation or routing [17, 11, 24]. See the survey [21] and the references therein for additional applications of distance oracles.

1.1 Linear Size Path-Reporting Distance Oracles

In [22], a PRDO with stretch $2k - 1$, size $O(kn^{1+1/k})$ and query time $O(k)$ was shown. Assuming the girth conjecture of Erdős [15], this result is best possible, up to the factor of k in the size and query time.[2] The query time was improved to $O(\log k)$ by [23]. Observe that $kn^{1/k} \geq \log n$ for any $k \geq 1$, so the PRDO of [22, 23] cannot be sparser than $\Theta(n \log n)$. Subsequent works [20, 7, 8] obtained distance oracles with stretch $2k - 1$, improved size $O(n^{1+1/k})$ and constant query time. However, these distance oracles are not path-reporting.

Of particular interest is trying to achieve a PRDO of linear size. The first such result [12] obtained a PRDO with size $O(tn)$, for any parameter $t \geq 1$, and $O(\log t)$ query time, but had a polynomial stretch $O(tn^{2/\sqrt{t}})$, and required that the aspect ratio of the weights is polynomially bounded. This result was improved by [13], who showed a PRDO with stretch $O(k)$ and size $O(n^{1+1/k})$, but at the cost of increasing the query time to $O(n^{1/k+\epsilon})$, where $\epsilon > 0$ is a *constant*. Note that the query time $O(n^{1/k+\epsilon})$ is prohibitively large - this term dominates the length of many of the output paths, so the PRDO suffers from large query time for these paths. For this reason, it is of special interest to construct linear size PRDOs with query time that is far less than polynomial in n, say polylogarithmic in n. Another variant of [13] does achieve a PRDO with query time $O(\log \log n)$ and stretch $\log^{O(1)} n$, however its size $O(n \log \log n)$ is no longer linear.

We conclude that in all previous results, every linear size distance oracle suffers from a polynomial stretch [12], has polynomial query time [13], or simply cannot report paths [8].

1.1.1 Our Results

In this work we devise the first linear size PRDO for general graphs with polylogarithmic stretch and low query time. Specifically, for any integer $k \geq 1$ our PRDO has stretch $O(k^{4.82})$, size $O(n^{1+1/k})$, and query time $O(\log k)$. Indeed, setting $k = \log n$ yields linear size, $\log^{O(1)} n$ stretch and $O(\log \log n)$ query time. Our main result is for the case $k = \log n$, where we get a linear size PRDO with low query time. In fact, for any $k > \frac{\log n}{\log \log \log n}$, our new PRDO improves all previous bounds.

Note that since the query time is low, in most cases it is dominated by the length of the reported path. Therefore, the strength of this result is not in the precise expression for the query time, but in the fact that the query time is far less than polynomial in n.

[1] Throughout this paper, $|P|$ denotes the number of *edges* in a path P.

[2] In fact, Erdős girth conjecture only implies that to achieve stretch $2k - 1$, any distance oracle must use $\Omega(n^{1+\frac{1}{k}})$ *bits*. However, in [8] a lower bound of $\Omega(n^{1+\frac{1}{k}})$ *words* (each of size $\log n$ bits) for PRDOs is proved.

We can refine our result to obtain an *ultra-compact* PRDO, whose size is as small as $n + o(n)$ (we measure the size by *words*, that is, the oracle uses storage of $n \log n + o(n \log n)$ bits), at the cost of increasing the query time to $\log^{O(1)} n$. In view of the lower bound of [8], this space usage is optimal, up to additive lower order terms. If the graph is unweighted, we offer a simpler construction with improved stretch $O(k^2)$.

1.2 Pairwise Path-Reporting Distance Oracles

A *pairwise distance oracle* is a distance oracle that is also given as input a set of pairs $\mathcal{P}$, and is required to answer queries only for pairs in $\mathcal{P}$. The problem of designing such oracles is related to the extensive research on *distance preservers*: these are subgraphs that preserve exactly all distances between pairs in $\mathcal{P}$. When allowing some stretch, these are sometimes called *pairwise spanners*.[3] Distance preservers were introduced in [9], and pairwise spanners have been studied in [10, 18, 4, 3, 6, 19].

In [13], an exact pairwise PRDO was shown with constant query time and size $O(n+|\mathcal{P}|^2)$. For distance preservers, [9] showed a lower bound of $\Omega(n^{2/3}|\mathcal{P}|^{2/3})$. Note that for $|\mathcal{P}| = n^{2-\delta}$, the lower bound implies that any distance preserver must have size $\Omega(|\mathcal{P}| \cdot n^{\delta/3})$, so there are no distance preservers with near-linear size (except for the trivial case when $\mathcal{P}$ contains a constant fraction of all pairs). In [5], it was proved that exact pairwise PRDOs suffer from the same lower bounds of exact preservers (see Theorem 14 in [5]). Thus, it is very natural to ask if the size of a pairwise PRDO can be reduced when allowing a small stretch. Specifically, we would like to obtain a very small stretch (e.g., $1 + \epsilon$ for any $\epsilon > 0$), and size that is proportional to $|\mathcal{P}| + n$ (which is the basic lower bound).

1.2.1 Our Results

In this work we devise a pairwise PRDO with near optimal size $n^{o(1)} \cdot (|\mathcal{P}| + n)$, constant query time, and stretch $1 + \epsilon$, for any $\epsilon > 0$ (the $o(1)$ term in the size depends logarithmically on ϵ). This result uses the techniques of [19] on hopsets and spanners, and extends them for pairwise path-reporting distance oracles.

1.3 Our Techniques

Our main result on linear size (and ultra-compact) PRDO uses a conceptually simple idea: we partition the graph into $O(n/k)$ clusters, and define the *cluster-graph* by contracting every cluster to a single vertex (keeping the lightest edge among parallel edges). Next, we apply the [22] distance oracle on this cluster-graph. In addition, we store a certain spanning tree for every cluster. Given a query (u, v), the algorithm first finds a path in the cluster-graph between the clusters containing u, v, and for each cluster in this path, it finds an inner-cluster path between the entry vertex and the exit vertex of the cluster, using the spanning tree.

In order to implement this framework, it is required to find a clustering so that the overhead created by going through the spanning tree of every cluster is small enough. For unweighted graphs, we apply a simple clustering with radius $O(k)$, and maintain a BFS tree for each cluster. However, for weighted graphs a more intricate clustering is required. Note that we cannot enforce a small diameter bound on all clusters, since by only restricting the diameter, each cluster can be very small, and we need to have at most $O(n/k)$ clusters. Instead, we use a variant of Borůvka clustering [1].

[3] In [19] pairwise spanners with small $1 + \epsilon$ stretch are called *near-exact preservers*.

In Borůvka's algorithm for minimum spanning tree, in each phase, every vertex adds its adjacent edge of minimal weight to a forest F (breaking ties consistently), and the connected components of F are contracted to yield the vertices of the next phase. If we truncate this process after t phases, we get a clustering, analyzed in [1]. Unfortunately, any phase in this clustering may produce long chains, in which case the stretch cannot be controlled. To rectify this, we delete certain edges in F, so that every cluster is a star, while ensuring that the number of non-singleton clusters is large enough. These stars are also the basis for the spanning tree of each cluster. The main technical part is analyzing the stretch induced by this clustering on the paths returned by calling the distance oracle on the cluster-graph.

1.4 Organization

After some preliminaries in Section 2, we show our PRDO for unweighted graph is in Section 3, and for weighted graphs in Section 4. Our result for pairwise PRDO appears in Section 5.

1.5 Bibliographic Note

Following this work, [14] showed (among other results) a PRDO of linear size with stretch $\tilde{O}(\log n)$ and query time $O(\log\log\log n)$.

2 Preliminaries

Let $G = (V, E)$ be an undirected weighted graph. In all that follows we assume that G is connected.

Spanners. For a parameter $\alpha \geq 1$, an *α-spanner* is a subgraph S of G, such that for every two vertices $u, v \in V$,

$$d_S(u,v) \leq \alpha \cdot d_G(u,v) \ . \qquad (1)$$

The spanner is called a *pairwise* spanner, if for a given a set of pairs $\mathcal{P}$, we only require (1) to hold for all $(u, v) \in \mathcal{P}$.

Trees. Let x, y be two vertices in a rooted tree. We denote by $p(x)$ the *parent* of x, which is the unique neighbor of x that lies on the path from x to the root, and by $h(x)$ its *height*, which is the number of edges on the path from x to the root. Denote by $lca(x, y)$ the lowest common ancestor of x, y, which is a vertex z such that x, y are both in its sub-tree, but not both in the sub-tree of any child of z. Note that the unique path between x, y in the tree is the concatenation of the unique paths from x to $lca(x, y)$, and from $lca(x, y)$ to y.

The following lemma let us easily find paths within a tree T.

▶ **Lemma 1.** *Let T be a rooted tree, and assume we are given $h(x)$ and $p(x)$ for every vertex x in T. There is an algorithm that given two distinct vertices a, b in a tree T, finds the unique path between a, b in T. The running time of this algorithm is proportional to the number of edges in the output path (or $O(1)$ if the path is empty).*

Proof. First, if $a = b$, then the desired path is empty and we return it in $O(1)$ time. Otherwise, if $h(a) > h(b)$, recursively find the unique path $P_{p(a),b}$ in T between $p(a)$ and b, and return $\{a, p(a)\} \circ P_{p(a),b}$. Symmetrically, if $h(b) \geq h(a)$, return $\{b, p(b)\} \circ P_{p(b),a}$.

The correctness of this algorithm follows from the fact that if, for example, $h(a) \geq h(b)$, and $a \neq b$, then b cannot be in the sub-tree of a in T, hence the unique path between a, b must pass through $p(a)$. In each recursive call we reduce the sum $h(a) + h(b)$ by 1, and therefore the algorithm ends when $a = b = lca(a, b)$. Therefore the running time of this algorithm is proportional to the length of the unique paths from a to $lca(a, b)$ and from b to $lca(a, b)$. The concatenation of these paths is exactly the returned path by our algorithm, which is the unique path in T between a, b. Hence, the running time is proportional to the number of edges in the output path, as desired. ◀

2.1 Thorup-Zwick PRDO

A main component of our new PRDO relies on a well-known construction by Thorup and Zwick [22]. Given a weighted graph G with n vertices and an integer parameter $k \geq 1$, they constructed a PRDO with stretch $2k - 1$, query time $O(k)$ and size $O(kn^{1+1/k})$.

A useful property of the Thorup-Zwick (TZ) PRDO is that for every query, it returns a path that is contained in a sub-graph S of G, such that $|S| = O(kn^{1+1/k})$. Notice that since the stretch of this PRDO is $2k - 1$, then S must be a $(2k - 1)$-spanner of G. We call S the *underlying spanner* of the PRDO. One can compute the underlying spanner S either during the PRDO construction, or after its construction by querying the PRDO on every pair of vertices, and computing the union of the resulting paths.

A result of [23] improved the query time of the TZ PRDO to $O(\log k)$ instead of $O(k)$, while returning the same path that the TZ PRDO returns. Indeed, when we use here the TZ PRDO, we consider its query time to be $O(\log k)$. The following theorem concludes this discussion.

▶ **Theorem 2** (By [22] and [23]). *Let G be an undirected weighted graph with n vertices, and let $k \geq 1$ be an integer parameter. There is a PRDO for G with stretch $2k - 1$, query time $O(\log k)$ and size $O(kn^{1+1/k})$, with an underlying spanner of the same size.*

3 Path-Reporting Distance Oracle for Unweighted Graphs

In this section we introduce a simple variant of our construction, tailored for unweighted graphs. We first apply a simple clustering, and store a BFS (Breadth First Search) tree for each cluster. We next apply the TZ PRDO on the resulting cluster-graph. Finally, each query (u, v) is answered by taking the path in the cluster-graph between the clusters containing u and v, and completing it to a path in G using the BFS trees.

3.1 Clustering

We start by dividing the graph into clusters, using the following lemma.

▶ **Lemma 3.** *Let $G = (V, E)$ be an undirected unweighted graph with n vertices. Let $k \in [1, n]$ be some integer. There is an algorithm that finds a partition $V = \bigcup_{i=1}^{q} C_i$, such that every C_i has a spanning tree $T_i = (C_i, E_i)$ with root r_i, where $E_i \subseteq E$ and for every $v \in C_i$, $d_{T_i}(v, r_i) \leq k$. In addition, the number of sets in this partition, q, is at most $\frac{n}{k}$.*

Proof. Fix some $r \in V$, and let $T = (V, E_T)$ be the BFS tree with r as a root. The tree T is actually the shortest paths tree from r in G, and so the path from every $v \in V$ to r in T is of length exactly $d_G(v, r)$, i.e., $d_T(v, r) = d_G(v, r)$. If every vertex $v \in V$ satisfies $d_G(v, r) \leq k$, then we can return the trivial partition $\{V\}$, with the spanning tree T and root r.

Otherwise, let v be the furthest leaf of T from r, that is, v maximizes the length $d_T(v, r)$. We know that $d_T(v, r) > k$, and since G is unweighted, there is a vertex r' on the path in T from v to r, with $d_T(v, r') = k$. Denote by T' the sub-tree of T rooted at r'.

Let $u \in V$ be a vertex in T'. Since r' is on the path from u to r, and on the path from v to r, we have

$$d_{T'}(u, r') = d_T(u, r) - d_T(r', r) \leq d_T(v, r) - d_T(r', r) = d_T(v, r') = k \ .$$

Therefore, if C is the set of vertices of T', we can return C as one of the sets in the desired partition, where its spanning tree is T' and its root is r'. We then delete C from G and continue recursively.

Note that the tree T' contains the path from v to r', which is of length k. Since G is unweighted, that means that T' contains at least k vertices, and so does C. Hence, the number of vertices in the graph, after the deletion of C, is at most $n - k$. Notice also that the tree T is still a tree, after the removal of T', thus the remaining graph is still connected. As a result, we can assume that our algorithm recursively partitions the remaining graph into at most $\frac{n-k}{k} = \frac{n}{k} - 1$ sets, with spanning trees and roots as desired. Together with the last set C, we obtain a partition into at most $\frac{n}{k}$ parts, with the wanted properties. ◀

Given the unweighted graph $G = (V, E)$ and the integer k, let $\mathcal{C}$ be a partition as in Lemma 3. For every $C \in \mathcal{C}$, let $T[C]$ and $r[C]$ be the spanning tree of C and its root. We define a new graph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ as follows.

▶ **Definition 4.** *The graph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ is defined as follows. The set $\mathcal{E}$ consists of all the pairs $\{C, C'\}$, where $C, C' \in \mathcal{C}$, such that there is an edge in G between C, C'.*

Given an edge $\{C, C'\} \in \mathcal{E}$, we denote by $e(C, C')$ the edge $\{x, y\}$ of G, where $\{x, y\}$ is some choice of an edge that satisfies $x \in C$, $y \in C'$.

We denote by F the forest that consists of the disjoint union of the trees $T[C]$, for every $C \in \mathcal{C}$. For a vertex $x \in V$, define $h(x)$ to be the height of x in the tree $T[C]$ such that $x \in C$, and $p(x)$ its parent in this tree.

3.2 Stretch Analysis

Fix any cluster C, let $T = T[C]$ be its spanning tree with root $r = r[C]$. For any two vertices $a, b \in T$, the unique path between them is a sub-path of the union between the two paths from a to r and from b to r. Both of these paths are of length at most k. Hence, the resulting path is of length at most $2k$, and this path is exactly the one that the algorithm from Lemma 1 returns.

▶ **Lemma 5.** *There is an algorithm that given two vertices $u, v \in V$, and a simple path $Q = (C_1, C_2, ..., C_t)$ in the graph $\mathcal{H}$, such that u is in C_1 and v is in C_t, returns a path P in G between u and v, with number of edges*

$$|P| \leq t \cdot (2k + 1) \ .$$

The running time of the algorithm is proportional to the number of edges in the output path. The required information for the algorithm is the set $\{h(x), p(x)\}_{x \in V}$, and the set $\{e(C_j, C_{j+1})\}_{j=1}^{t-1}$.

Proof. Given the edges $\{C_j, C_{j+1}\}$, the set $\{e(C_j, C_{j+1})\}_{j=1}^{t-1}$ can be used to find $x_j, y_j \in V$ (vertices of the original graph $G = (V, E)$), such that $x_j \in C_j$, $y_j \in C_{j+1}$ and $\{x_j, y_j\} \in E$. Define also $y_0 = u, x_t = v$. For every $j \in [1, t]$, using the set $\{h(x), p(x)\}_{x \in V}$, we can use Lemma 1 to find a path P_j in G between y_{j-1} and x_j, with length at most $2k$. Finding all of these paths takes time that is proportional to the sum of lengths of these paths.

The returned path by this algorithm is

$$P = P_1 \circ \{x_1, y_1\} \circ P_2 \circ \{x_2, y_2\} \circ \cdots \circ \{x_{t-1}, y_{t-1}\} \circ P_t \ .$$

The time needed to report this path is $O(\sum_{j=1}^{t} |P_j|) = O(|P|)$. The length of this path is

$$t - 1 + \sum_{j=1}^{t} |P_j| \le t - 1 + t \cdot 2k < t \cdot (2k + 1) \ .$$

This concludes the proof of the lemma. ◀

3.3 A PRDO for Unweighted Graphs

We are now ready to introduce the construction of our small size PRDO.

▶ **Theorem 6.** *Let $G = (V, E)$ be an undirected unweighted graph with n vertices, and let $k \in [1, \log n]$ be some integer parameter. There is a path-reporting distance oracle for G with stretch $2k(2k+1) = O(k^2)$, query time $O(\log k)$ and size $O(n^{1+\frac{1}{k}})$.*

Proof. Denote by TZ the PRDO from Theorem 2 with the parameter k, when constructed over the graph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ (the clustering $\mathcal{C}$ is constructed with k as the radius[4]). Let $S_{TZ} \subseteq \mathcal{E}$ be the set of the edges of the underlying spanner of TZ. In addition, for a given vertex $x \in V$, denote by $C(x)$ the vertex of $\mathcal{H}$ (i.e., cluster) that contains x. Recall also that $h(x)$ is the height of x in the tree spanning $C(x)$ and $p(x)$ denotes the parent of x in this tree.

We define our new PRDO for the undirected unweighted graph $G = (V, E)$. This PRDO contains the following information.

1. The TZ PRDO.
2. The set $\{e(C, C') \mid \{C, C'\} \in S_{TZ}\}$.
3. The variables $\{h(x), p(x)\}_{x \in V}$.
4. The variables $\{C(x)\}_{x \in V}$.

Given a query $(u, v) \in V^2$, our PRDO queries TZ on the vertices $C(u), C(v)$ of $\mathcal{H}$. Let $Q = (C(u) = C_1, C_2, ..., C_t = C(v))$ be the resulting path, and note that all of its edges are in S_{TZ}. Then, using the sets $\{e(C_j, C_{j+1})\}_{j=1}^{t-1} \subseteq \{e(C, C') \mid \{C, C'\} \in S_{TZ}\}$ and $\{h(x), p(x)\}_{x \in V}$, we find a path P in G between u, v using the algorithm from Lemma 5. The resulting path P has length of

$$|P| \le (|Q| + 1)(2k + 1) = t \cdot (2k + 1) \ ,$$

and it is returned as an output to the query.

Note that the path Q that TZ returned satisfies $|Q| = t - 1 \le (2k - 1)|R|$, where R is the shortest path in $\mathcal{H}$ between $C(u)$ and $C(v)$. Let $P_{u,v}$ be the actual shortest path in G between u and v. Suppose that the vertices of $\mathcal{H}$ that $P_{u,v}$ passes through, by the order that it passes through them, are $(T_1, T_2, ...T_q)$. By the definition of $\mathcal{H}$, there must be an edge $\{T_j, T_{j+1}\}$ in $\mathcal{H}$ for every $j \in [1, q-1]$. Hence, $R' = (T_1, T_2, ..., T_q)$ is a path in $\mathcal{H}$, between $T_1 = C(u)$ and $T_q = C(v)$, with length of at most $|P_{u,v}| = d_G(u, v)$. Since R is the shortest path in $\mathcal{H}$ between $C(u)$ and $C(v)$, we have $|R| \le d_G(u, v)$.

[4] Actually, by constructing the clustering $\mathcal{C}$ with radius $\frac{k}{8}$ instead of k, the stretch of our new PRDO decreases from $4k^2$ to k^2. In the same way, one can achieve an arbitrarily small leading constant in the stretch.

As a result,

$$\begin{aligned}|P| &\leq t\cdot(2k+1)\\ &\leq ((2k-1)|R|+1)(2k+1)\\ &\leq ((2k-1)d_G(u,v)+1)(2k+1)\\ &= (4k^2-1)d_G(u,v)+2k+1\\ &\leq (4k^2+2k)d_G(u,v)=2k(2k+1)d_G(u,v)\ .\end{aligned}$$

Thus, the stretch of our PRDO is at most $2k(2k+1)$.

The query time of our oracle consists of the time required for running a query of TZ, and of the time required for finding the path P. By Theorem 2 and Lemma 5, the total time for these two computations is $O(\log k+|P|)$ which is $O(\log k)$ by our conventional PRDO notations.

As for the size of our PRDO, note that the variables $\{h(x),p(x)\}_{x\in V}$ (item 3 in the description of the oracle) can be stored using only $O(n)$ space. The size of the set $\{e(C,C')\mid\{C,C'\}\in S_{TZ}\}$ equals to the size of S_{TZ}. Therefore, by Theorem 2, the size of TZ, as well as the size of this set (items 1 and 2), is

$$O(k|\mathcal{C}|^{1+\frac{1}{k}})\ .$$

Recall that by Lemma 3, the size of $\mathcal{C}$ is at most $\frac{n}{k}$. We conclude that the total size of our new PRDO is

$$O(n+k\cdot(\frac{n}{k})^{1+\frac{1}{k}})=O(n^{1+\frac{1}{k}})\ . \qquad ◀$$

An Ultra-Compact PRDO for Unweighted Graphs. We can modify our PRDO for unweighted graphs, and get a PRDO of size $n+o(n)$. Here, the required storage for our PRDO is measured by *words* - each of size at most $\log n$ bits. Decreasing the size of our PRDO is done at the cost of increasing the query time and (slightly) the stretch. The details are deferred to the full version of this paper.

4 Path-Reporting Distance Oracle for Weighted Graphs

In this section we devise our PRDO for weighted graphs. The basic framework is similar to the unweighted case: create a clustering of the graph, select a spanning tree for each cluster, and then apply the TZ PRDO over the cluster-graph. To answer a query (u,v), we use the path in the cluster-graph between the clusters containing u,v, and complete it inside each cluster via the spanning trees edges.

The main differences from the unweighted setting are: 1) we use a more intricate clustering, *Borůvka's clustering*, and 2) the trees spanning each cluster are not BFS trees, but are a subset of the MST (Minimum Spanning Tree) of the graph. These changes are needed in order to achieve the desired properties - that the number of clusters is small enough, while the stretch caused by going through the spanning trees of the clusters is controlled.

4.1 Clustering via Borůvka Forests

In this section we construct a clustering via a spanning forest of a graph. This construction is based on the well-known algorithm by Borůvka for finding a minimum spanning tree in a graph. Similar constructions can be found in [16, 1, 2].

▶ **Definition 7.** *Given an undirected weighted graph $G = (V, E)$, and a vertex $v \in V$, we denote by e_v the minimum-weight edge among the adjacent edges to v in the graph G. If there is more than one edge with this minimum weight, e_v is chosen to be the one that is the smallest lexicographically.*

▶ **Definition 8.** *Given an undirected weighted graph $G = (V, E)$, the* Borůvka Forest *of G is the sub-graph $G' = (V, E')$ of G, where*

$$E' = \{e_v \mid v \in V\} \ .$$

Each connected component T of G' is called a Borůvka Tree. *The* root *of T is chosen to be one of the adjacent vertices to the minimum-weight edge in T (if there are several such minimum-weight edges, we pick the smallest one lexicographically, and the choice between its two adjacent vertices is arbitrary).*

To justify the use of the words "forest" and "tree", we prove the following lemma.

▶ **Lemma 9.** *The graph G' is a forest. Moreover, if T is a tree in G', x is a vertex of T, and $p(x)$ is x's parent in T (that is, the next vertex on the unique path from x to the root of T), then $\{x, p(x)\} = e_x$.*

Proof. First, we prove that G' is a forest. Seeking contradiction, assume that G' contains a cycle C, and let $\{u, v\}$ be the heaviest edge in C (if there are several edges with the largest weight, choose the one that is largest lexicographically). Note that since u has at least one adjacent edge in C, that is lighter than $\{u, v\}$, then it cannot be that $e_u = \{u, v\}$ (recall that e_u is the lightest edge adjacent to u). Similarly, it cannot be that $e_v = \{u, v\}$. Hence, we get a contradiction to the fact that $\{u, v\}$ is an edge of G' - since every such edge must be the edge e_v of one of its endpoints v.

Next, Let T be a tree in G', denote its root by v, and let $x \neq v$ be a vertex of T. We prove by induction over the height of x, $h(x)$, which is the number of edges in the unique path between x and v in T.

When $h(x) = 1$, we have $p(x) = v$. We consider two cases. If $\{x, v\}$ is the minimum-weight edge in T, then by definition e_x must be this edge, i.e., $e_x = \{x, p(x)\}$. If $\{x, v\}$ is not the minimum-weight edge in T, then e_v must be some other adjacent edge to v, thus $e_v \neq \{x, v\}$. But then, the reason that $\{x, v\}$ is in E' must be that $\{x, p(x)\} = \{x, v\} = e_x$.

For $h(x) > 1$, notice that $h(p(x)) = h(x) - 1$, and therefore by the induction hypothesis, $\{p(x), p(p(x))\} = e_{p(x)}$. But then, the edge $\{x, p(x)\}$ cannot be equal to $e_{p(x)}$, so it must be equal to e_x. ◀

The following lemma bounds the number of connected components (i.e., trees) in G'.

▶ **Lemma 10.** *The number of connected components in G' is at most $\frac{1}{2}|V|$.*

Proof. Let $C = (V_C, E'_C)$ be a connected component of G', and let $x \in V_C$. The edge $e_x = \{x, y\}$ is in C, hence y is also a vertex of C. In particular, $|V_C| \geq 2$. Hence, if $\{C_i\}_{i=1}^t$ are the connected components of G', then $|V_{C_i}| \geq 2$ for every $i \in [1, t]$. Thus,

$$\frac{1}{2}|V| = \frac{1}{2}\sum_{i=1}^{t}|V_{C_i}| \geq \frac{1}{2} \cdot 2t = t \ . \qquad ◀$$

Next, we trim the trees in the Borůvka forest so that each of them will be a *star*, instead of a general tree. For this purpose, we will need the following definitions.

▶ **Definition 11.** *Let G' be the Borůvka forest of G. For a vertex $x \in V$, denote by $h(x)$ the height of x in the Borůvka tree containing it. Define*

$$E'_0 = \{\{a,b\} \in E' \mid \min\{h(a),h(b)\} = 0 \mod 2\} ,$$

$$E'_1 = \{\{a,b\} \in E' \mid \min\{h(a),h(b)\} = 1 \mod 2\} .$$

We denote by E'' the largest set among these two.

Given an undirected weighted graph $G=(V,E)$, the Partial Borůvka Forest *of G is the graph $G''=(V,E'')$.*

▶ **Definition 12.** *A* Star *is a rooted tree $S=(V_S,E_S)$ with root v such that for every $x \in V_S \setminus \{v\}$, $\{x,v\} \in E_S$.*

▶ **Lemma 13.** *The partial Borůvka forest $G''=(V,E'')$ is a forest, where every tree is a star. In addition, if S is a star in G'', x is its root and $z \neq x$ is some other vertex of S, then $\{z,x\} = e_z$.*

Proof. Notice that G'' is a sub-graph of the Borůvka forest G', hence G'' is also a forest. We assume that $E''=E'_0$, and the proof for the case where $E''=E'_1$ is symmetric.

Let T be a tree in G', with root r. Note that for any vertex $x \neq r$ we always have $h(x) = h(p(x))+1$, and thus $\min\{h(x),h(p(x))\} = h(p(x)) = h(x)-1$. We conclude that if $h(x)$ is even, then $\{x,p(x)\} \notin E''$, and if $h(x)$ is odd, then $\{x,p(x)\} \in E''$.

Now let S be a tree in G'', and let x be the vertex in S that has minimal $h(x)$. It cannot be that $h(x)$ is odd, otherwise $p(x)$ is connected to x in E'', thus $p(x)$ is also in S and has a smaller value of $h(p(x)) = h(x)-1$. Therefore, $h(x)$ is even. By the discussion above we know that all of the children of x in T (y's that satisfy $p(y)=x$) have an edge in E'' to x, but their children have no such edge. That is, S is a star with x as a root, where all the other vertices in S are the children of x.

The last part of the lemma follows from the fact that we just proved, that the only other vertices in a star S with a root x, are the children of x. By Lemma 9, for every such child z, the edge $\{z,x\} = \{z,p(z)\} = e_z$. ◀

The following lemma bounds the number of trees in the partial Borůvka forest of a graph.

▶ **Lemma 14.** *The number of stars in G'' is at most $\frac{3}{4}|V|$.*

Proof. Recall the Borůvka forest $G'=(V,E')$. In every spanning forest (V,F) of a graph $G=(V,E)$, the number of trees is exactly $|V|-|F|$. Thus, by Lemma 10, we get

$$|V| - |E'| \leq \frac{1}{2}|V| ,$$

and therefore $|E'| \geq \frac{1}{2}|V|$. By the definition of E'', it contains at least half of these edges (since it equals to the larger set among two sets that cover the entire set E'). We conclude that $|E''| \geq \frac{1}{4}|V|$, and the number of trees in G'', which are stars, is

$$|V| - |E''| \leq |V| - \frac{1}{4}|V| = \frac{3}{4}|V| .$$ ◀

4.1.1 A Hierarchy of Forests

Given an undirected weighted graph $G=(V,E)$, we construct a sequence of forests $\{F_i = (V,E_i)\}_{i=0}^{l}$, where the integer parameter $l \geq 0$ will be determined later. For $i=0$, define $E_0 = \emptyset$. Then, for every $i \in [0,l]$, define the cluster-graph $\mathcal{H}_i = (\mathcal{C}_i, \mathcal{E}_i)$ as follows.

The set $\mathcal{C}_i$ is defined to be the set of the disjoint trees of the forest F_i. For every $T, T' \in \mathcal{C}_i$, denote by $e(T, T')$ the minimum-weight edge in E among the edges between T and T'. If there is no such edge, denote $e(T, T') = \bot$. Then define

$$\mathcal{E}_i = \{\{T, T'\} \mid e(T, T') \neq \bot\} .$$

The weight of an edge $\{T, T'\} \in \mathcal{E}_i$ is defined to be the same as the weight of the edge $e(T, T') \in E$.

For any $i \in [0, l]$, given the graph $\mathcal{H}_i$, let $\mathcal{H}_i'' = (\mathcal{C}_i, \mathcal{E}_i'')$ be the partial Borůvka forest of $\mathcal{H}_i$. The graph $\mathcal{H}_i''$ is a disjoint union of stars. Let $\mathcal{S}$ be such star and let T_0 be its root.

Define the tree Z in G to be the tree that is formed by the union of the trees in $\mathcal{S}$ and the edges $e(T, T_0)$, for every $T \neq T_0$ in $\mathcal{S}$. The root of the tree Z is defined to be the root of T_0. Finally, for any $i \in [0, l-1]$, the forest $F_{i+1} = (V, E_{i+1})$ is defined to be the disjoint union of the rooted trees Z that are formed as was described, for all stars in $\mathcal{H}_i''$.

▶ **Lemma 15.** *For every $i \in [0, l]$, the forest F_i has at most $(\frac{3}{4})^i|V|$ trees.*

Proof. We prove the lemma by induction over $i \in [0, l]$. For $i = 0$, F_0 is defined to be the graph $(V, \emptyset)$, so the number of trees in F_0 is $|V|$ and the claim holds.

For $i > 0$, recall the graphs $\mathcal{H}_{i-1}$ and $\mathcal{H}_{i-1}''$ that were used in the definition of F_i. By the induction hypothesis, F_{i-1} consists of at most $(\frac{3}{4})^{i-1}|V|$ trees, which are exactly the vertices of $\mathcal{H}_{i-1}$. Then, by Lemma 14, the graph $\mathcal{H}_{i-1}''$ has at most $\frac{3}{4} \cdot (\frac{3}{4})^{i-1}|V| = (\frac{3}{4})^i|V|$ stars. The forest F_i consists of a single tree Z for every star $\mathcal{S}$ in $\mathcal{H}_{i-1}''$, thus the number of trees in F_i is at most $(\frac{3}{4})^i|V|$, as desired. ◀

4.2 Stretch Analysis

Due to space considerations, we only state here the main lemma that will be used for bounding the stretch of our PRDO, without proof. The proof of this lemma, as well as some other lemmata , appears in the full version of this paper.

In the next lemma, we use the notations $p_i(x)$ and $h_i(x)$ to denote the parent and the height of x in the tree of F_i that contains x.

▶ **Lemma 16.** *There is an algorithm that given two vertices $u, v \in V$, and a simple path $Q = (S_1, S_2, ..., S_t)$ in the graph $\mathcal{H}_i$, such that u is in S_1 and v is in S_t, returns a path P in G between u and v, with*

$$w(P) \leq 3^{i+1}(d_G(u, v) + w(Q)) .$$

The running time of the algorithm is proportional to the number of edges in the output path P. The required information for the algorithm is the set $\{h_i(x), p_i(x)\}_{x \in V}$, and the set $\{e(S_j, S_{j+1})\}_{j=1}^{t-1}$.

4.3 A PRDO for Weighted Graphs

We are now ready to introduce our small size path-reporting distance oracle.

▶ **Theorem 17.** *Let $G = (V, E)$ be an undirected weighted graph with n vertices, and let $k \geq 1$ be an integer parameter. There is a path-reporting distance oracle for G with stretch $k^{\log_{4/3} 4} < k^{4.82}$, query time $O(\log k)$ and size $O(n^{1+\frac{1}{k}})$.*

Proof. Given the graph $G = (V, E)$, we construct the hierarchy of forests $\{F_i\}_{i=0}^{l}$ from Section 4.1.1, where $l = \lfloor \log_{4/3} k \rfloor - 2$. Consider the graph $\mathcal{H}_l = (\mathcal{C}_l, \mathcal{E}_l)$ that is defined in Section 4.1.1. For every $x \in V$, denote by $h_l(x)$ the number of edges in the unique path from x to the root of the tree of F_l that x belongs to. Let $p_l(x)$ be the parent of x in that tree. Lastly, let $S(x)$ be the vertex of $\mathcal{H}_l$ (i.e., tree) that contains x.

Denote by TZ the PRDO from Theorem 2 with the parameter k, when constructed over the graph $\mathcal{H}_l$. Let $S_{TZ} \subseteq \mathcal{E}_l$ be the set of edges of the underlying spanner of TZ.

Our new PRDO D stores the following information.

1. The oracle TZ.
2. The set $\{e(T, T') \mid \{T, T'\} \in S_{TZ}\}$.
3. The variables $\{h_l(x), p_l(x)\}_{x \in V}$.
4. The variables $\{S(x)\}_{x \in V}$.

Given a query $(u, v) \in V^2$, the oracle D queries TZ on the vertices $S(u), S(v)$ of $\mathcal{H}_l$. Let $Q = (S(u) = S_1, S_2, ..., S_t = S(v))$ be the resulting path, and note that all of its edges are in S_{TZ}. Then, using the sets $\{e(S_j, S_{j+1})\}_{j=1}^{t-1} \subseteq \{e(T, T') \mid \{T, T'\} \in S_{TZ}\}$ and $\{h_l(x), p_l(x)\}_{x \in V}$, the oracle D uses the algorithm from Lemma 16 to find a path P in G between u, v with

$$w(P) \leq 3^{l+1}(d_G(u, v) + w(Q)) \ .$$

The path P is returned as an output to the query. Note that the path Q that TZ returned satisfies

$$w(Q) \leq (2k - 1)w(R) \ ,$$

where R is the shortest path in $\mathcal{H}_l$ between $S(u)$ and $S(v)$. Similarly to the proof of Theorem 17, it is easy to verify that $w(R) \leq d_G(u, v)$.

As a result,

$$\begin{aligned} w(P) &\leq 3^{l+1}(d_G(u, v) + w(Q)) \\ &\leq 3^{l+1}(d_G(u, v) + (2k - 1)w(R)) \\ &\leq 3^{l+1}(d_G(u, v) + (2k - 1)d_G(u, v)) \\ &= 2k \cdot 3^{l+1} d_G(u, v) \\ &\leq 2k \cdot 3^{\log_{4/3} k - 1} d_G(u, v) \\ &< k^{1+\log_{4/3} 3} d_G(u, v) = k^{\log_{4/3} 4} d_G(u, v) \ . \end{aligned}$$

Thus the stretch of our PRDO is smaller than $k^{\log_{4/3} 4}$.

The query time of our oracle consists of the time required for running a query of TZ, and of the time required for computing the resulting path P by Lemma 16. By Theorem 2 and Lemma 16, the total time for these two computations is $O(\log k + |P|)$, which is $O(\log k)$ by our conventional PRDO notations.

As for the size of the PRDO D, note that the variables $\{h_l(x), p_l(x), S(x)\}_{x \in V}$ (items 3 and 4 in the description of D) can be stored using only $O(n)$ space. The size of the set $\{e(T, T') \mid \{T, T'\} \in S_{TZ}\}$ equals to the size of S_{TZ}. Therefore, by Theorem 2, the size of TZ, as well as the size of this set (items 1 and 2 in the description of D), is

$$O(k|\mathcal{C}_l|^{1+\frac{1}{k}}) \ .$$

Recall that $\mathcal{C}_l$ is the set of vertices of $\mathcal{H}_l$. This set consists of the trees in the forest F_l. By Lemma 15, the number of these trees is at most

$$(\frac{3}{4})^l|V| = (\frac{3}{4})^{\lfloor \log_{4/3} k \rfloor - 2} n \le (\frac{3}{4})^{\log_{4/3} k - 3} n = \frac{64n}{27k} .$$

Hence, the total size of our PRDO is

$$O(n + k \cdot (\frac{64n}{27k})^{1+\frac{1}{k}}) = O(n + n^{1+\frac{1}{k}}) = O(n^{1+\frac{1}{k}}) . \qquad ◀$$

An Ultra-Compact PRDO for Weighted Graphs. As in the unweighted version, the PRDO presented above can be fine-tuned into an ultra-compact PRDO (with size $n + o(n)$), at the cost of increasing the stretch and the query time. The details are deferred to the full version of this paper.

5 Pairwise Path-Reporting Distance Oracle

Our construction of a pairwise PRDO relies on the pairwise spanner of Kogan and Parter, from their recent paper [19] (in which the pairwise spanner is called a "near-exact preserver"). One of their useful results, that they also relied on for constructing their pairwise spanners, is the following lemma on hopsets. We first recall the definition of hopsets.

Let $G = (V, E)$ be a weighted undirected graph. For vertices $u, v \in V$ and some positive integer β, $d_G^{(\beta)}(u, v)$, denotes the weight of the lightest path between u and v in G, among the paths that have at most β edges. An (α, β)-*hopset* is a set $H \subseteq \binom{V}{2}$, such that for every two vertices $u, v \in V$,

$$d_G(u, v) \le d_{G\cup H}^{(\beta)}(u, v) \le \alpha \cdot d_G(u, v) ,$$

where the weight of an edge $(x, y) \in H$ is defined to be $d_G(x, y)$.

The proof of the following lemma can be found in [19].

▶ **Lemma 18** (Lemma 4.4 from [19]). *Let $G = (V, E)$ be an undirected weighted graph on n vertices, and let $k, D \ge 1$ be integer parameters. For every $0 < \epsilon < 1$, there exists a $(1 + \epsilon, \beta)$-hopset H for G, where $\beta = O(\frac{\log k}{\epsilon})^{\log k} \cdot D$ and*

$$|H| = O\left(\left(\frac{n \log n}{D}\right)^{1+\frac{1}{k}}\right) .$$

Similarly to the constructions in [19], we now show how a pairwise PRDO can be produced, using the hopsets from Lemma 18. We will use the notation $\beta(\epsilon, k) = O(\frac{\log k}{\epsilon})^{\log k}$ for brevity.

▶ **Theorem 19.** *Let $G = (V, E)$ be an undirected weighted graph on n vertices and let $\mathcal{P} \subseteq V^2$ be a set of pairs of vertices. For every $\epsilon \in (0, 1)$, there exists a pairwise path-reporting distance oracle with stretch $1 + \epsilon$, query time $O(1)$ and size*

$$O\left(\frac{\log n \cdot (\log \log n)^2}{\epsilon}\right)^{\log \log n} \cdot \tilde{O}(|\mathcal{P}| + n) = n^{o_\epsilon(1)} \cdot O(n + |\mathcal{P}|) .$$

Proof. Let $n = D_0 > D_1 > \cdots > D_l = 2$ be some sequence of integer parameters that will be determined later. Denote $k = \log n$, and for a given $\epsilon \in (0, 1)$, denote $\epsilon' = \frac{\epsilon}{2(l+1)}$. Let $H_0, H_1, ..., H_l$ be the resulting hopsets when applying Lemma 18 on $\epsilon', k = \log n$ and $D_0, D_1, ..., D_l$ respectively. That is, H_i is a $(1+\epsilon', \beta_i)$-hopset with size $O((\frac{n \log n}{D_i})^{1+\frac{1}{k}})$, where $\beta_i = \beta(\epsilon', k) \cdot D_i$. For $i = 0$, note that $\beta_i \ge n$, thus we can simply assume that $H_0 = \emptyset$ (if it is not the case, we *define* H_0 to be $\emptyset$, which is a $(1, n)$-hopset).

We now define our oracle D to contain the following information. For every $i \in [1, l]$ and for every $(x, y) \in H_i$, let $Q_{x,y}$ be the shortest path in $G \cup H_{i-1}$ between x, y, among the paths that contain at most β_{i-1} edges. In addition, for every $(x, y) \in \mathcal{P}$, let $P_{x,y}$ be the shortest path in $G \cup H_l$ between x, y, among the paths with at most β_l edges. Our oracle D stores all of these paths: $\bigcup_{i=1}^{l}\{Q_{x,y}\}_{(x,y)\in H_i} \cup \{P_{x,y}\}_{(x,y)\in\mathcal{P}}$.

Given a query $(u, v) \in \mathcal{P}$, we find the path $P_l = P_{u,v} \subseteq G \cup H_l$ that is stored in D. Then, we replace every edge $(x, y) \in H_l$ on P_l by the corresponding path $Q_{x,y} \subseteq G \cup H_{l-1}$. The result is a path P_{l-1} between u, v in $G \cup H_{l-1}$. Every edge $(x, y) \in H_{l-1}$ on P_{l-1} is then replaced by the path $Q_{x,y} \subseteq G \cup H_{l-2}$, to get a path P_{l-2} between u, v in $G \cup H_{l-2}$. We continue in the same way, until finally reaching to a path P_0 between u, v in the graph $G \cup H_0 = G$. We return P_0 as an output to the query.

By the hopset property, we know that

$$w(P_l) = w(P_{u,v}) = d^{(\beta_l)}_{G\cup H_l}(u, v) \leq (1 + \epsilon')d_G(u, v) \ .$$

Similarly, every $(x, y) \in P_l$ that is also in H_l, is replaced with the path $Q_{x,y}$, that has a weight of

$$w(Q_{x,y}) = d^{(\beta_{l-1})}_{G\cup H_{l-1}}(x, y) \leq (1 + \epsilon')d_G(x, y) = (1 + \epsilon')w(x, y) \ .$$

Thus, the resulting path P_{l-1} has a weight of at most $1 + \epsilon'$ times the weight of P_l, that is

$$w(P_{l-1}) \leq (1 + \epsilon')w(P_l) \leq (1 + \epsilon')^2 d_G(u, v) \ .$$

Proceeding in the same way, we conclude that $w(P_0) \leq (1+\epsilon')^{l+1}d_G(u, v)$. Hence, the stretch of our distance oracle is

$$(1 + \epsilon')^{l+1} = (1 + \frac{\epsilon}{2(l + 1)})^{l+1} \leq e^{\frac{\epsilon}{2}} \leq 1 + \epsilon \ .$$

For analysing the query time of our distance oracle, we can think of the query algorithm as a single pass on the path P_l, where every time that an edge of H_l is reached, we replace it with the appropriate path $Q_{x,y}$, and continue inside $Q_{x,y}$ recursively. Since every step produces an edge that will appear in the output path, the query time is proportional to this output path. Observe, however, that the resulting path is actually a walk, and not necessarily a simple path. By our convention of writing the query time of PRDOs, this query time is $O(1)$.

Lastly, we analyse the size of our pairwise PRDO. Note that by their definitions, the paths $P_{x,y}$, for every $(x, y) \in \mathcal{P}$ are of length at most β_l. Similarly, the length of $Q_{x,y}$, for $(x, y) \in H_i$ is at most β_{i-1}. Therefore, the total space required for storing these paths is at most

$$\begin{aligned}
|\mathcal{P}| \cdot \beta_l + \sum_{i=1}^{l} |H_i| \cdot \beta_{i-1} &= |\mathcal{P}| \cdot \beta(\epsilon', k) \cdot D_l + \sum_{i=1}^{l} O\left(\left(\frac{n \log n}{D_i}\right)^{1+\frac{1}{k}}\right) \cdot \beta(\epsilon', k) \cdot D_{i-1} \\
&= \beta(\epsilon', k) \cdot \left(|\mathcal{P}| \cdot 2 + \sum_{i=1}^{l} O\left(\left(\frac{n \log n}{D_i}\right)^{1+\frac{1}{k}}\right) \cdot D_{i-1}\right) \\
&= \beta(\epsilon', k) \cdot O\big(|\mathcal{P}| + (n \log n)^{1+\frac{1}{k}} \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}) \\
&= O\left(\frac{\log k}{\epsilon/2l}\right)^{\log k} \cdot O\big(|\mathcal{P}| + (n \log n)^{1+\frac{1}{\log n}} \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}})
\end{aligned}$$

$$= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot O\big(|\mathcal{P}| + n \log n \cdot \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big)$$

$$= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + n \cdot \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big) \ .$$

For making the last term small, we choose $D_i = \left\lceil n^{(\frac{k}{k+1})^i} \right\rceil$, and thus $\frac{D_{i-1}}{D_i^{1+\frac{1}{k}}} \leq \frac{n^{(\frac{k}{k+1})^{i-1}}+1}{n^{(\frac{k}{k+1})^i \cdot (1+\frac{1}{k})}} = \frac{n^{(\frac{k}{k+1})^{i-1}}+1}{n^{(\frac{k}{k+1})^{i-1}}} \leq 2$. For this choice of D_i, since we want D_l to be 2, we must have $n^{(\frac{k}{k+1})^l} \leq 2$, that is, $l \geq \log_{\frac{k+1}{k}}(\log n)$. Notice that $\log_{\frac{k+1}{k}}(\log n) = \frac{\log\log n}{\log(1+\frac{1}{k})} \leq \frac{\log\log n}{\log(2^{\frac{1}{k}})} = k \log\log n$, thus we can choose $l = \lceil k \log\log n \rceil = \lceil \log n \cdot \log\log n \rceil$.

In conclusion, the size of our pairwise PRDO is at most

$$\begin{aligned} O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + n \cdot \sum_{i=1}^{l} \frac{D_{i-1}}{D_i^{1+\frac{1}{k}}}\big) &= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + n \cdot \sum_{i=1}^{l} 2\big) \\ &= O\left(\frac{l \cdot \log k}{\epsilon}\right)^{\log k} \cdot \tilde{O}\big(|\mathcal{P}| + l \cdot n\big) \\ &= O\left(\frac{\log n \cdot (\log\log n)^2}{\epsilon}\right)^{\log\log n} \cdot \tilde{O}\big(|\mathcal{P}| + n\big) \\ &= n^{o_\epsilon(1)} \cdot O\big(|\mathcal{P}| + n\big) \end{aligned}$$ ◀

References

1 MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity clustering: Hierarchical clustering at scale. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6864–6874, 2017. URL: `https://proceedings.neurips.cc/paper/2017/hash/2e1b24a664f5e9c18f407b2f9c73e821-Abstract.html`.

2 Amartya Shankha Biswas, Michal Dory, Mohsen Ghaffari, Slobodan Mitrović, and Yasamin Nazari. Massively parallel algorithms for distance approximation and spanners. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 118–128, 2021.

3 Greg Bodwin. New results on linear size distance preservers. *SIAM J. Comput.*, 50(2):662–673, 2021. `doi:10.1137/19M123662X`.

4 Greg Bodwin, Keerti Choudhary, Merav Parter, and Noa Shahar. New fault tolerant subset preservers. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ICALP.2020.15`.

5 Greg Bodwin, Gary Hoppenworth, and Ohad Trabelsi. Bridge girth: A unifying notion in network design. *arXiv preprint*, 2022. `arXiv:2212.11944`.

6 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. *ACM Trans. Algorithms*, 17(4):36:1–36:24, 2021. `doi:10.1145/3490147`.

7 Shiri Chechik. Approximate distance oracles with constant query time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 654–663. ACM, 2014. `doi:10.1145/2591796.2591801`.

8 Shiri Chechik. Approximate distance oracles with improved bounds. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10. ACM, 2015. doi:10.1145/2746539.2746562.

9 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.

10 Marek Cygan, Fabrizio Grandoni, and Telikepalli Kavitha. On pairwise spanners. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 209–220. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.209.

11 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina Anna Zweig, editors, *Algorithmics of Large and Complex Networks – Design, Analysis, and Simulation [DFG priority program 1126]*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009. doi:10.1007/978-3-642-02094-0_7.

12 Michael Elkin, Ofer Neiman, and Christian Wulff-Nilsen. Space-efficient path-reporting approximate distance oracles. *Theor. Comput. Sci.*, 651:1–10, 2016. doi:10.1016/j.tcs.2016.07.038.

13 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Trans. Algorithms*, 12(4):50:1–50:31, 2016. doi:10.1145/2888397.

14 Michael Elkin and Idan Shabat. Path-reporting distance oracles with near-logarithmic stretch and linear size. *CoRR*, abs/2304.04445, 2023. doi:10.48550/arXiv.2304.04445.

15 P. Erdős. Extremal problems in graph theory. In *Theory of Graphs and Applications (Proc. Sympos. Smolenice)*, pages 29–36, 1964.

16 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 202–219. SIAM, 2016.

17 Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical optimization of optimal path finding for transportation applications. In *CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, November 12–16, 1996, Rockville, Maryland, USA*, pages 261–268. ACM, 1996. doi:10.1145/238355.238550.

18 Telikepalli Kavitha. New pairwise spanners. *Theory Comput. Syst.*, 61(4):1011–1036, 2017. doi:10.1007/s00224-016-9736-7.

19 Shimon Kogan and Merav Parter. Having hope in hops: New spanners, preservers and lower bounds for hopsets. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 766–777. IEEE, 2022. doi:10.1109/FOCS54457.2022.00078.

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

21 Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46:45:1–31, 2014. doi:10.1145/2530531.

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

23 Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–549. SIAM, 2013.

24 Christos D. Zaroliagis. Engineering algorithms for large network applications. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms – 2008 Edition*. Springer, 2008. doi:10.1007/978-0-387-30162-4_125.