Algorithmica (2013) 67:516–528
DOI 10.1007/s00453-013-9825-9

Preprocess, Set, *Query!*

Ely Porat · Liam Roditty

Received: 18 December 2011 / Accepted: 9 August 2013 / Published online: 28 August 2013

Abstract Thorup and Zwick (J. ACM 52(1):1–24, 2005 and STOC'01) in their seminal work introduced the notion of *distance oracles*. Given an n-vertex weighted undirected graph with m edges, they show that for any *integer* $k \geq 1$ it is possible to preprocess the graph in $\tilde{O}(mn^{1/k})$ time and generate a compact data structure of size $O(kn^{1+1/k})$. For each pair of vertices, it is then possible to retrieve an estimated distance with multiplicative stretch $2k-1$ in $O(k)$ time. For $k=2$ this gives an oracle of $O(n^{1.5})$ size that produces in constant time estimated distances with stretch 3. Recently, Pătraşcu and Roditty (In: Proc. of 51st FOCS, 2010) broke the theoretical status-quo in the field of distance oracles and obtained a distance oracle for sparse unweighted graphs of $O(n^{5/3})$ size that produces in constant time estimated distances with stretch 2.

In this paper we show that it is possible to break the stretch 2 barrier at the price of non-constant query time in unweighted undirected graphs. We present a data structure that produces estimated distances with $1+\varepsilon$ stretch. The size of the data structure is $O(nm^{1-\varepsilon'})$ and the query time is $\tilde{O}(m^{1-\varepsilon'})$. Using it for sparse unweighted graphs we can get a data structure of size $O(n^{1.87})$ that can supply in $O(n^{0.87})$ time estimated distances with multiplicative stretch 1.75.

Keywords Graph algorithms · Distance oracles · Data structures

L. Roditty's work is supported by Israel Science Foundation (grant no. 822/10).

E. Porat · L. Roditty (✉)
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel
e-mail: liamr@macs.biu.ac.il

E. Porat
e-mail: porately@macs.biu.ac.il

1 Introduction

Thorup and Zwick [15] introduced the notion of *distance oracles*, that is, data structures capable of representing almost shortest paths efficiently, both in terms of space requirement and query time. Given an n-vertex weighted undirected graph with m edges, they show that for any integer $k \geq 1$ it is possible to preprocess the graph in $\tilde{O}(mn^{1/k})$ time and generate a compact data structure of size $O(kn^{1+1/k})$. For each pair of vertices, it is then possible to retrieve an estimated distance with multiplicative stretch $2k-1$ in $O(k)$ time. An estimated distance has multiplicative (additive) stretch c if for two vertices at distance Δ it is at least Δ and at most $c\Delta$ ($\Delta + c$). We use (α, β) to denote a combination of multiplicative stretch of α and additive stretch of β, that is, the estimated distance is at most $\alpha\Delta + \beta$. Thorup and Zwick called such data structures *distance oracles* as their query time is constant for any constant stretch. Thorup and Zwick [15] showed also that their data structure is optimal for dense graphs. Based on the girth conjecture of Erdős they showed that there exist dense enough graphs which cannot be represented by a data structure of size less than $\Omega(n^{1+1/k})$ without increasing the stretch above $2k-1$ for any integral k.

Sommer, Verbin, and Yu [14] proved a three-way tradeoff between space, stretch and query time of approximate distance oracles. They show that any distance oracle that can give stretch α answers to distance queries in time $O(t)$ must use $n^{1+\Omega(1/(t\alpha))}/\log n$ space. Their result is obtained by a reduction from lopsided set disjointness to distance oracles, using the framework introduced by Pătraşcu [10]. Any improvement to this lower bound requires a major breakthrough in lower bounds techniques. In particular, it does not imply anything even for slightly non-constant query time as $\Omega(\log n)$ and slightly non-linear space as $n^{1.01}$.

This suggests that upper bounds are currently the only realistic way to attack the Thorup and Zwick space-stretch-query tradeoff. There are several possible ways to obtain a progress:

1. To consider sparse graphs. In particular, graphs with less than $n^{1+1/k}$ edges.
2. To consider also additive error, that is, to get below $2k-1$ multiplicative stretch with an additional additive stretch.
3. Non-constant query time.

The first way and the second one are closely related. We cannot gain from introducing also additive stretch without getting an improved multiplicative stretch for sparse graphs (i.e., $m = O(n)$). A data structure with size $S(m,n)$ and stretch (α, β) implies a data structure with size $S((\beta+1)m, n+\beta m)$ and multiplicative stretch of α, as if we divide every edge into $\beta+1$ edges then all distances become a multiple of $\beta+1$ and additive stretch of β is useless. For graphs with $m = O(n)$ the size of the data structure is asymptotically the same.

Recently, Pătraşcu and Roditty [11] broke the theoretical status-quo in the field of distance oracles. They obtained a distance oracle for sparse unweighted graphs of size $O(n^{5/3})$ that can supply in $O(1)$ time an estimated distance with multiplicative stretch 2. For dense graphs the distance oracle has the same size and stretch $(2,1)$. Pătraşcu and Roditty [11] showed also a conditional lower bound for distance oracle that is based on a conjecture on the hardness of the set intersection problem. They showed that a distance oracle for unweighted graphs with $m = \tilde{O}(n)$ edges, which can distinguish between distances of 2 and 4 in constant time (as multiplicative stretch strictly less than 2 implies) requires $\tilde{\Omega}(n^2)$ space, assuming the conjecture holds. Thus, non-constant query time is essential to get stretch smaller than 2.

In this paper we show that one can gain by allowing non-constant query time. We break the barrier of 2 for multiplicative stretch at the price of non-constant query time. Surprisingly, we show that it is possible to get an arbitrary small multiplicative stretch.

We prove the following:

Theorem 1 *For any unweighted graph and any $\varepsilon > 0$, we can construct in $O(mn)$ time a data structure of size $O(nm^{1-\frac{\varepsilon}{4+2\varepsilon}})$ that given any two vertices s and t returns an estimated distance with multiplicative stretch $1+\varepsilon$ in $\tilde{O}(m^{1-\frac{\varepsilon}{4+2\varepsilon}})$ time. We can provide also an actual path in time proportional to its length.*

When $m = O(n)$ we get a data structure of size $O(n^{2-\frac{\varepsilon}{4+2\varepsilon}})$ with query time $\tilde{O}(n^{1-\frac{\varepsilon}{4+2\varepsilon}})$.

Studying sparse graphs makes a lot of sense not only from theoretical perspective but also from practical point of view. Sparse graphs are the most realistic network scenario in real world applications. Road networks and the Internet are the most popular examples to such scenarios.

Prior to the work of Pătraşcu and Roditty [11], the main focus of the theoretical research on distance oracles was on improving the construction and retrieval times of the distance oracle of Thorup and Zwick. Baswana and Kavitha [3] showed that distance oracles matching the performance of the oracles of Thorup and Zwick can be constructed in $\tilde{O}(n^2)$ time, which is an improvement over the original $O(mn^{1/k})$ construction time for dense enough graphs. Mendel and Naor [9] gave an oracle with $O(1)$ query time (as opposed to $O(k)$ query time of the Thorup-Zwick oracle), at the price of increasing the stretch to $O(k)$. The focus on faster implementations can be partially explained by the fact that it was widely believed that the space/approximation tradeoff of the distance oracle of Thorup and Zwick is optimal. Another line of research considered restricted metric spaces such as Euclidean metric spaces and metric spaces of low doubling dimension. In these restricted cases it is possible to get $1+\varepsilon$ multiplicative stretch with almost linear space. See Bartal et al. [2] and reference therein for the state of the art in distance oracles for restricted metric spaces. A summary of the main results appears in Table 1.

Experimental studies on the Thorup-Zwick distance oracle were conducted by Krioukov et al. [8]. Motivated by these experiments, Chen et al. [4] show that the Thorup-Zwick oracles requires less space on random power-law graphs. Enăchescu et al. [7] obtain similar results for Erdős-Rényi random graphs.

Closely related to the notion of distance oracles is the notion of *spanners*, introduced by Peleg and Schäffer [12]. The main difference between distance oracles and spanners is that spanners are only concerned with the question of how much space is needed to store approximate distances, disregarding the time needed to retrieve approximate distances and paths. Moreover, spanners are allowed to have only edges of

Table 1 A summary of distance data structures

Reference	Stretch	Query	Space	Input
Trivial	1	$\tilde{O}(m)$	$O(m)$	Weighted digraphs
Trivial	1	$O(1)$	$O(n^2)$	Weighted digraphs
[15]	$2k-1$	$O(k)$	$O(n^{1+1/k})$	Weighted graphs
[9]	$O(k)$	$O(1)$	$O(n^{1+1/k})$	Weighted graphs
[11]	$(2, 1)$	$O(1)$	$O(n^{5/3})$	Unweighted graphs
[11]	2	$O(1)$	$O(n^{5/3})$	Unweighted sparse graphs
[2]	$1+\varepsilon$	$O(1)$	$[\varepsilon^{-O(\lambda)} + 2^{O(\lambda \log \lambda)}]n$	Metric space of doubling dimension λ
This paper	$1+\varepsilon$	$\tilde{O}(n^{1-\frac{\varepsilon}{4+2\varepsilon}})$	$O(n^{2-\frac{\varepsilon}{4+2\varepsilon}})$	Unweighted sparse graphs
This paper	1.75	$\tilde{O}(n^{0.87})$	$O(n^{1.87})$	Unweighted sparse graphs

the original graph. Multiplicative-additive spanners were introduced by Elkin and Peleg [5, 6]. More constructions of such spanners were obtained by Thorup and Zwick [16] and Pettie [13]. Notice that for sparse graphs distance oracles and spanners are very different. If the graph has $O(n)$ edges, a spanner is trivial: just include all the edges. For a distance oracle, such graphs can be hard.

Our algorithm is composed of three stages. In the first stage we preprocess the graph and partition its edges into clusters. It is important to note that we use edge clusters as opposed to the more traditional vertex clusters that are usually used in spanners and distance oracles constructions. In the second stage we set some additional distance information to the clusters. Finally, in the third stage we are ready to answer distance queries. Our algorithm is remarkably simple and can be implemented easily.

Recent Developments Shortly after our paper was first published in its proceeding version Agarwal and Godfrey [1] presented a data structure for weighted graphs with improved tradeoffs.

Paper Organization The rest of this paper is organized as its title. In the next Section we describe how to preprocess the graph into clusters. In Sect. 3 we describe the additional information set to the cluster. In Sect. 4 we present the query algorithm and its analysis. We end in Sect. 5 with some concluding remarks and open problems.

2 Preprocess

In the preprocessing stage we partition the edges of the graph into clusters as follows. We start to grow in parallel shortest paths trees for each vertex of the graph. As the graph is unweighted each step of the shortest paths computation is a step of Breadth-First-Search (BFS) algorithm and thus equivalent to scanning a single edge. We stop the parallel search when we reach to a certain predefined limit L on the number of edges to be scanned in the construction of each of the trees. We refer to the limit

Algorithm 1: Clusters($G(V, E), L$)

```
$E' \leftarrow E$;
$V' \leftarrow V$;
$\mathcal{C} \leftarrow \emptyset$;
while $V' \neq \emptyset$ do
    foreach $u \in V'$ do
        $\langle Tree(u), H_u \rangle \leftarrow$ BFS($G(V, E'), u, L$);
    Let $s$ be a vertex with minimal fully explored depth;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \langle s, Tree(s), r_s, H_s \rangle$;
    $E' \leftarrow E' \setminus H_s$;
    $V' \leftarrow \{u \mid$
    $u$ is in a connected component of $G(V, E')$ with at least $L$ edges $\}$;
return $\mathcal{C}$;
```

parameter L as the budget of the BFS algorithm, that is, the number of edges it is allowed to scan. (It will be set, roughly, to m^{ϵ}.)

Given a vertex $v \in V$ let H_v be the edges that are scanned by a BFS from v with budget L. (Notice that H_v is the set of *all* L edges scanned by the BFS algorithm and thus may contain edges that are not part of the BFS tree.) Let $V(H_v)$ be the vertices that are endpoints of edges from H_v. Let $Tree(v)$ be the shortest paths tree of v in $G(V(H_v), H_v)$. For every $w \in V(H_v)$ we denote with $Tree(v, w)$ the length of the shortest path between v and w in $G(V(H_v), H_v)$. We say that the *fully explored depth* of $Tree(v)$ is its maximal depth minus one and denote it with r_v. We pick the tree with minimal fully explored depth and create a cluster. (If there is more than one we pick one arbitrarily.) Assume that s is the root of the BFS tree. A cluster is represented by the four tuple $\langle s, Tree(s), r_s, H_s \rangle$. We refer to s as the center of this cluster. Notice that a vertex can be the center of more than one cluster. We now proceed to compute clusters in the graph $G(V, E \setminus H_s)$. This process is repeated until every connected component of the graph has less than L edges. The set of clusters is then returned by the algorithm. It is important to note that in each iteration of the algorithm we remove the edges that belong to the new cluster and continue to compute clusters on the remaining graph. As a result of that an edge can be in at most one cluster[1] but a vertex can be in many different clusters. The computation of clusters is presented in Algorithm 1. In the next Lemma we bound the number of clusters.

Lemma 1 *If $\mathcal{C}$ is the set of clusters computed by Algorithm 1 with budget L then $|\mathcal{C}| \leq m/L$.*

Proof Each cluster contains exactly L edges. When a cluster is formed we removed all its edges from the graph. Thus, the total number of clusters is at most m/L. □

[1] An edge might not be in any cluster as we do not cluster edges in components with less than L edges.

Next, we divide clusters into sets according to their depth. We define $\mathcal{C}^{\Delta}$:

$$\mathcal{C}^{\Delta} = \{C \mid C = \langle u, Tree(u), r_u, H_u \rangle \in \mathcal{C} \wedge r_u \le \Delta\},$$

and $E(\mathcal{C}^{\Delta})$:

$$E(\mathcal{C}^{\Delta}) = \{e \mid \exists C = \langle u, Tree(u), r_u, H_u \rangle \in \mathcal{C}^{\Delta} \wedge e \in H_u\}.$$

Let $E' = E \setminus E(\mathcal{C}^{\Delta})$, that is, the set of edges that do not belong to clusters of fully explored depth at most Δ. Roughly speaking, in the next Lemma we show that since the graph $G(V, E')$ is relatively sparse it is possible to have a lower bound on the depth of any BFS tree computed from any of its vertices. This useful property will be used later on in the analysis of our query algorithm.

Lemma 2 *Let $\mathcal{C}$ be the set of clusters computed with budget L. Let Δ be integral and let $E' = E \setminus E(\mathcal{C}^{\Delta})$. For every $s \in V$ that belongs to a connected component with at least L edges in $G(V, E')$, the fully explored depth of the tree created by* BFS$(G(V, E'), s, L)$ *is at least $\Delta + 1$.*

Proof Clusters are added to $\mathcal{C}$ in a non-decreasing order of their fully explored depth. If we remove all the edges that belong to clusters with fully explored depth of at most Δ then any BFS from any vertex in a connected component of at least L edges with a budget of L must have fully explored depth of at least $\Delta + 1$ as otherwise a cluster of fully explored depth at most Δ is still present in $G(V, E')$, a contradiction. □

3 Set

We now turn to the second stage. In this stage we set some additional distance information to the clusters that were computed in the first stage. This information is kept in an additional data structure to allow efficient queries. For each cluster we compute the distance to every other vertex of the graph. Let $\delta_E(u, v)$ be the length of the shortest path between u and v in $G(V, E)$. The distance between a cluster $C = \langle s, Tree(s), r_s, H_s \rangle$ and a vertex $u \in V$ is defined as follows:

$$\delta_E(u, C) = \min_{x \in V(H_s)} \delta_E(u, x).$$

It is important to note that the distance between a vertex and a cluster is computed in the original graph $G(V, E)$. Let $p(u, C)$ be a vertex that realizes the minimal distance. If the cluster center s realizes the minimal distance we set $p(u, C)$ to s (the reason to that will be clear later), otherwise, we set $p(u, C)$ to an arbitrary vertex that realizes the minimal distance. This is all the information saved in our data structure. We summarize its main properties in the next lemma.

Lemma 3 *The data structure described above can be constructed in $O(mn)$ time and has a size of $O(mn/L)$.*

Proof From Lemma 1 it follows that there are $O(m/L)$ clusters. Thus, the clustering algorithm has $O(m/L)$ iterations and since each iteration takes $O(nL)$ time (as we grow n trees and scan L edges to grow each tree) its total running time is $O(mn)$. We compute also the distances of the graph in $O(mn)$ time. Using the distances and the clusters we compute the additional information. For each cluster we keep the distance to any vertex. This can be done in $O(nL)$ time per cluster. Since the clusters set is of size $O(m/L)$ the total size of the data structure is $O(mn/L)$. □

4 Query

Let $G(V, E)$ be an unweighted undirected graph and let $s, t \in V$ be two arbitrary vertices. Let $\Delta = \delta_E(s, t)$. In this section we show that for every $\varepsilon > 0$ the query algorithm returns an estimated distance with multiplicative stretch $1 + 4\varepsilon$ in $\tilde{O}(m^{1-\frac{\varepsilon}{4+2\varepsilon}})$ time. The size of the data structure is $O(nm^{1-\frac{\varepsilon}{4+2\varepsilon}})$.

Roughly speaking, the main idea of the query algorithm is to check whether there is a cluster with a relatively small fully explored depth ($\varepsilon\Delta$) that intersects the shortest path. If this is the case then we can obtain a good approximation using the information saved in our data structure. If there is no cluster of this scale that intersects the shortest path then the shortest path exists also in a much sparser graph on which a variant of bidirectional BFS can be executed in sub-linear time. Obviously, the full description is much more involved as we do not really know the shortest path or its length. Moreover, the case that $\varepsilon\Delta$ is less than 1 requires a special care.

We now describe more formally the query algorithm. We provide pseudo-code in Algorithm 2. The algorithm is composed of two stages.[2] In the first stage we scan the clusters that were created using the preprocessing algorithm from Sect. 2. We compute an estimate for each cluster $C = \langle u, \mathit{Tree}(u), r_u, H_u\rangle$ on the length of a path that is composed from the following three portions, a shortest path between s and $p(s, C)$, a shortest path between $p(s, C)$ and $p(t, C)$ that uses only edges of H_u, and a shortest path between $p(t, C)$ and t. The distance between s and $p(s, C)$ and the distance between t and $p(t, C)$ are computed in the preprocessing and the length of the shortest path between $p(s, C)$ and $p(t, C)$ is bounded $\hat{\delta}_{H_u}(p(s, C), p(t, C))$ which is 1 in case that $r_u = 0$ and either $s = u$ or $t = u$ and $2r_u + 2$ otherwise. (Notice that $\hat{\delta}_{H_u}(p(s, C), p(t, C))$ means that we have an approximation of a shortest path between $p(s, C)$ and $p(t, C)$ that uses only H_u edges.) We take the minimal path among all the possible clusters. This stage covers the case in which a shortest path intersects a cluster with a relatively small fully explored depth.

In the second stage we try to improve the estimate on the distance that we have obtained in the first stage by growing shortest paths trees from s and t. This stage is composed of $\log n$ iterations. In the i-th iteration, where $i \in [0, \log n - 1]$ we set $d = 2^i$. The graph that is used in this iteration is $G(V, E')$, where $E' = E \setminus \{e \mid \exists\langle u, \mathit{Tree}(u), r_u, H\rangle \in \mathcal{C} : r_u \le \varepsilon d \wedge e \in H\}$, that is, only edges that are part of clusters whose fully explored depth is at least $\lceil \varepsilon d\rceil$. We compute two sets of vertices S and T, where $S = \bigcup_{i=1}^{\lceil\frac{1}{\varepsilon}\rceil} S_i$ and $T = \bigcup_{i=1}^{\lceil\frac{1}{\varepsilon}\rceil} T_i$. The sets S_1 and T_1 are obtained by

[2]There is also a trivial step in which we perform BFS with budget L from the two vertices in the original graph to cover the case that their shortest path is in a component with less than L edges.

Algorithm 2: Query(s, t, ε)

```
$\hat{\delta}(s,t) \leftarrow \infty$;
// Stage 1
**foreach** $C = \langle u, Tree(u), r_u, H_u \rangle \in \mathcal{C}$ **do**
    **if** $r_u = 0$ *and* $(p(s,C) = u$ *or* $p(t,C) = u)$ **then**
        $\hat{\delta}_{H_u}(p(s,C), p(t,C)) = 1$
    **else**
        $\hat{\delta}_{H_u}(p(s,C), p(t,C)) = 2r_u + 2$
    **if** $\hat{\delta}(s,t) > \delta_E(s,C) + \hat{\delta}_{H_u}(p(s,C), p(t,C)) + \delta_E(t,C)$ **then**
        $\hat{\delta}(s,t) = \delta_E(s,C) + \hat{\delta}_{H_u}(p(s,C), p(t,C)) + \delta_E(t,C)$;
// Stage 2
**foreach** $i \in [0, \log n)$ **do**
    $d \leftarrow 2^i$;
    **if** $\varepsilon d \geq 1$ **then**
        $E' = E \setminus E(\mathcal{C}^{\varepsilon d})$;
    **else**
        $E' = E \setminus \{e \mid \exists \langle u, Tree(u), r_u, H_u \rangle \in \mathcal{C} : r_u = 0 \wedge e \in H_u \wedge V(H_u) =$
        $L + 1\}$;
    $\langle Tree(s), H_s \rangle \leftarrow \mathrm{BFS}(G(V, E'), s, L)$;
    $\langle Tree(t), H_t \rangle \leftarrow \mathrm{BFS}(G(V, E'), t, L)$;
    $S \leftarrow S_1 \leftarrow V(H_s)$;
    $T \leftarrow T_1 \leftarrow V(H_t)$;
    $h(s, \cdot) = Tree(s, \cdot)$;
    $h(t, \cdot) = Tree(t, \cdot)$;
    **for** $j \leftarrow 1$ **to** $\lceil \frac{1}{\varepsilon} \rceil - 1$ **do**
        $S_{j+1} \leftarrow \mathrm{Expand}(s, S_j, h)$;
        $T_{j+1} \leftarrow \mathrm{Expand}(t, T_j, h)$;
        $S \leftarrow S \cup S_{j+1}$;
        $T \leftarrow T \cup T_{j+1}$;
    **if** $\hat{\delta}(s,t) > \min_{x \in S \cap T} h(s,x) + h(t,x)$ **then**
        $\hat{\delta}(s,t) = \min_{x \in S \cap T} h(s,x) + h(t,x)$
**return** $\hat{\delta}(s,t)$;
```

executing a BFS with budget L from s and t, respectively. The sets S_{i+1} and T_{i+1} are obtained by executing BFS with budget L for every vertex of S_i and T_i, respectively (see Algorithm 3). We also maintain a distances array h. If $\Delta = \delta_{E'}(s,t)$ then the exact distance is found when $d < \Delta \leq 2d$. Finally, we output the minimal distance that we have obtained. It is important to note that the value of the output corresponds to an actual path in the input graph, thus, we only need to bound the output value from above as it cannot be smaller than Δ.

Algorithm 3: Expand(u, U, h)

```
$W \leftarrow \emptyset$;
foreach $x \in U$ do
    $\langle Tree(x), H_x \rangle \leftarrow$ BFS$(G(V, E'), x, L)$;
    $W \leftarrow W \cup V(H_x)$;
foreach $y \in W$ do
    if $h(u, y) > h(u, x) + Tree(x, y)$ then $h(u, y) = h(u, x) + Tree(x, y)$
return $W$;
```

Next, we show that the algorithm returns a $(1 + 4\varepsilon)$-approximation of Δ. For the simplicity of the presentation we divide the proof into two sections one for the case that case that $\varepsilon\Delta \geq 1$ and one for the case that $\varepsilon\Delta < 1$. We end with an analysis of the running time.

4.1 Case of $\varepsilon\Delta \geq 1$

Lemma 4 *Let $\varepsilon > 0$ and let $\varepsilon\Delta \geq 1$. The algorithm* Query(s, t, ε) *returns an estimated distance with $1 + 4\varepsilon$ stretch.*

Proof Let $P = \{(s, u_1), (u_1, u_2), \ldots, (u_\ell, t)\}$ be the edges of a shortest path between s and t. Let $\mathcal{C}' = \{\langle u, Tree(u), r_u, H_u\rangle \mid \langle u, Tree(u), r_u, H_u\rangle \in \mathcal{C} \wedge r_u \leq \varepsilon\Delta\}$. We first consider the case that one of the edges of P belongs to a cluster of $\mathcal{C}'$, that is, there is a cluster $C = \langle u, Tree(u), r_u, H_u\rangle$ in $\mathcal{C}'$, such that $H_u \cap P \neq \emptyset$.

Let $(x, y) \in H_u \cap P$. We show that in this case the first stage of the query algorithm finds a $(1 + \varepsilon)$-approximation of Δ. Recall that in the first stage we consider every cluster and in particular the cluster C. We use the value $\delta_E(s, C) + \hat{\delta}_{H_u}(p(s, C), p(t, C)) + \delta_E(t, C)$ as an approximation. Since $(x, y) \in H_u$ it follows that $\delta_E(s, C) \leq \delta_E(s, x)$ and $\delta_E(t, C) \leq \delta_E(y, t)$.

Moreover, as $C \in \mathcal{C}'$ it follows that $r_u \leq \varepsilon\Delta$. Combining this with the fact that $\varepsilon\Delta \geq 1$ it follows that $2r_u + 2 \leq 4\varepsilon\Delta$. Thus we get that $\delta_E(s, C) + \hat{\delta}_{H_u}(p(s, C), p(t, C)) + \delta_E(t, C) \leq (1 + 4\varepsilon)\Delta$.

We now consider the case in which there is no cluster $C \in \mathcal{C}'$ with edge set H, such that $H \cap P \neq \emptyset$. We show that in this case an exact shortest path is found by the second stage of the query algorithm. In this stage there is an iteration i such that $d = 2^i$ and $d < \Delta \leq 2d$. The graph that is used in this iteration is $G(V, E')$, where, $E' = E \setminus E(\mathcal{C}^{\varepsilon d})$. Recall that we compute two sets of vertices S and T, where $S = \bigcup_{i=1}^{\lceil\frac{1}{\varepsilon}\rceil} S_i$ and $T = \bigcup_{i=1}^{\lceil\frac{1}{\varepsilon}\rceil} T_i$ using Algorithm 3. We also maintain a distances array h. We now prove that: □

Claim 2 *For every $1 \leq i \leq \lceil\frac{1}{\varepsilon}\rceil$, if $\delta_{E'}(s, x) \leq i\lceil\varepsilon d\rceil$ then $x \in S$ and $h(s, x) = \delta_{E'}(s, x)$.*

Proof The proof is by induction on i. For the base of the induction we show that if $\delta_{E'}(s, x) \leq \lceil\varepsilon d\rceil$ then $x \in S_1$ and $h(s, x) = \delta_{E'}(s, x)$. We compute S_1 using a BFS from s with budget L on the graph $G(V,E')$. From Lemma 2 it follows that its fully explored depth is at least $\lceil \varepsilon d \rceil$. Thus, any vertex x for which $\delta_{E'}(s,x) \le \lceil \varepsilon d \rceil$ is added to S_1 by the algorithm and the value of $h(s,x)$ is set to $\delta_{E'}(s,x)$. For the induction hypothesis we assume that if $\delta_{E'}(s,x) \le i\lceil \varepsilon d \rceil$ then $x \in S_i$ and $h(s,x) = \delta_{E'}(s,x)$. We prove that if $i\lceil \varepsilon d \rceil < \delta_{E'}(s,x) \le (i+1)\lceil \varepsilon d \rceil$ then $x \in S_{i+1}$ and $h(s,x) = \delta_{E'}(s,x)$. Let x' be a vertex on a shortest path between s and x for which $\delta_{E'}(s,x') = i\lceil \varepsilon d \rceil$. From the induction hypothesis it follows that $x' \in S_i$ and $h(s,x') = \delta_{E'}(s,x')$. When Expand is called with S_i, a BFS with budget L is executed from x'. From Lemma 2 it follows that this BFS tree will contain x as its fully explored depth is at least $\lceil \varepsilon d \rceil$ and $\delta_{E'}(x',x) \le \lceil \varepsilon d \rceil$. As Expand updates $h(s,x)$ with the minimal available distance it also follows that $h(s,x) = \delta_{E'}(s,x)$.

The same claim holds for the vertex t and the set T with identical proof. We now turn to prove that there is at least one vertex from the shortest path between s and t in $S \cap T$.

Claim 3 *There is a vertex x on the shortest path between s and t such that $h(s,x) = \delta_{E'}(s,x)$, $h(t,x) = \delta_{E'}(t,x)$ and $x \in S \cap T$.*

Proof Recall that $\delta_{E'}(s,t) = \Delta$ and $d < \Delta \le 2d$. Thus, there is a vertex x from the shortest path between s and t for which $\delta_{E'}(s,x) \le d$ and $\delta_{E'}(t,x) \le d$. This implies that $\delta_{E'}(s,x) \le i\lceil \varepsilon d \rceil$ for some $1 \le i \le \lceil \frac{1}{\varepsilon} \rceil$ and $\delta_{E'}(t,x) \le j\lceil \varepsilon d \rceil$ for some $1 \le j \le \lceil \frac{1}{\varepsilon} \rceil$. By applying Claim 2 for s we get that $x \in S_i$ and $h(s,x) = \delta_{E'}(s,x)$. Similarly, by applying Claim 2 for t we get that $x \in T_j$ and $h(t,x) = \delta_{E'}(t,x)$. Hence $x \in S \cap T$. □

From Claim 3 it follows that there is a vertex $x \in S \cap T$ such that $h(s,x) = \delta_{E'}(s,x)$ and $h(t,x) = \delta_{E'}(t,x)$. Thus, the value of $\hat{\delta}(s,t)$ is set by the algorithm to $\delta_{E'}(s,x) + \delta_{E'}(t,x) = \Delta$. □

4.2 Case of $\varepsilon\Delta < 1$

We now turn to the case that $\varepsilon\Delta < 1$. One possible solution to this case is to divide every edge into a constant number of edges as suggested in the Introduction. This however increases the size of our data structure to $O(m^2/L)$. We show here that by a slight change to the second stage of the query algorithm and a careful analysis it is possible to keep the size of the data structure $O(nm/L)$.

Lemma 5 *Let $\varepsilon > 0$ and let $\varepsilon\Delta < 1$. The algorithm* Query(s,t,ε) *returns the exact distance Δ.*

Proof Let $P = \{(s,u_1),(u_1,u_2),\ldots,(u_\ell,t)\}$ be the set of edges of a shortest path between s and t. Let $\mathcal{C}' = \{\langle u, \mathit{Tree}(u), r_u, H_u\rangle \mid \langle u, \mathit{Tree}(u), r_u, H_u\rangle \in \mathcal{C} \wedge r_u = 0 \wedge V(H_u) = L+1\}$. Notice that every cluster $\langle u, \mathit{Tree}(u), r_u, H_u\rangle \in \mathcal{C}'$ is a star with $L+1$ vertices whose center is u.

We first consider the case that there is a cluster $C = \langle u, \mathit{Tree}(u), r_u, H_u\rangle \in \mathcal{C}'$ such that $H_u \cap P \neq \emptyset$. In the first stage of the query algorithm we consider every cluster and in particular the cluster C. We use the value $\delta_E(s,C)+\hat{\delta}_{H_u}(p(s,C),p(t,C))+\delta_E(t,C)$ as an approximation.

A shortest path P can intersect a star cluster in two possible ways. One is that $H_u \cap P=\{(x,y)\}$, where $\delta_E(s,x)<\delta_E(s,y)$. The other one is that $H_u\cap P=\{(x,y),(y,z)\}$, where $\delta_E(s,x)<\delta_E(s,y)<\delta_E(s,z)$.

Assume that $H_u\cap P=\{(x,y)\}$, and there is no other shortest path P' such that $H_u\cap P'=\{(x,y),(y,z)\}$. The cluster center u must be either x or y. Assume, wlog, that $u=x$. As we assume that there is no other shortest path P' that intersects the cluster in the second way it cannot be that $\delta_E(s,C)<\delta_E(s,u)$. Hence, $\delta_E(s,C)=\delta_E(s,u)$ and $p(s,C)=u$. (Recall that we have set $p(s,C)$ to the cluster center if it realizes the minimal distance.) Moreover, since $p(s,C)=u$ it follows that $\delta_{H_u}(p(s,C),p(t,C))=1$. Now, our algorithm sets $\hat{\delta}_{H_u}(p(s,C),p(t,C))$ to 1 since $p(s,C)=u$ and $r_u=0$.

Finally, as $H_u\cap P=\{(x,y)\}$ it follows that $\delta_E(t,C)\le\delta_E(y,t)$. Adding it all together we get:

$$\delta_E(s,C)+\hat{\delta}_{H_u}\bigl(p(s,C),p(t,C)\bigr)+\delta_E(t,C)\le\delta_E(s,x)+1+\delta_E(y,t)=\Delta.$$

We now assume that there is at least one shortest path P for which $H_u\cap P=\{(x,y),(y,z)\}$, where $\delta_E(s,x)<\delta_E(s,y)<\delta_E(s,z)$. Since the cluster is a star y must be its center. This implies that the algorithm sets $\hat{\delta}_{H_u}(p(s,C),p(t,C))$ to $2r_u+2=2$. As $H_u\cap P=\{(x,y),(y,z)\}$ it follows that $\delta_E(s,C)\le\delta_E(s,x)$ and $\delta_E(t,C)\le\delta_E(t,z)$.

Adding it all together we get:

$$\delta_E(s,C)+\hat{\delta}_{H_u}\bigl(p(s,C),p(t,C)\bigr)+\delta_E(t,C)\le\delta_E(s,x)+2+\delta_E(z,t)=\Delta.$$

We now turn to the case in which there is no cluster $C\in\mathcal{C}'$ with edge set H, such that $H\cap P\neq\emptyset$. In the second stage of the query algorithm there exists an iteration i such that $d=2^i$ and $d<\Delta\le 2d$. In this iteration $\varepsilon d<\varepsilon\Delta<1$ and we consider the graph $G(V,E')$, where $E'=E\setminus\{e\mid\exists\langle u,\mathit{Tree}(u),r_u,H_u\rangle\in\mathcal{C}:r_u=0\ \wedge\ e\in H_u\ \wedge\ V(H_u)=L+1\}$, that is, only edges that are not part of a star cluster are used. The next Claim is similar to Claim 2.

Claim 4 *For every $1\le i\le\lceil\frac{1}{\varepsilon}\rceil$, if $\delta_{E'}(s,x)\le i$ then $x\in S$ and $h(s,x)=\delta_{E'}(s,x)$.*

Proof The proof is by induction on i. For the base of the induction we show that if $\delta_{E'}(s,x)\le 1$ then $x\in S_1$ and $h(s,x)=\delta_{E'}(s,x)$. We compute S_1 using a BFS from s with budget L on the graph $G(V,E')$. The degree of s is strictly less than L as otherwise $G(V,E')$ contains a star cluster. Hence, x must be reached by a BFS from s with budget L and it is added to S_1. The value of $h(s,x)$ is set to $\delta_{E'}(s,x)$. We now turn to prove the general case. We assume that if $\delta_{E'}(s,x)\le i$ then $x\in S_i$ and $h(s,x)=\delta_{E'}(s,x)$. We prove that if $\delta_{E'}(s,x)=i+1$ then $x\in S_{i+1}$ and $h(s,x)=\delta_{E'}(s,x)$. Let x' be a vertex on a shortest path between s and x for which $\delta_{E'}(s,x')=i$. From the induction hypothesis it follows that $x'\in S_i$ and $h(s,x')=\delta_{E'}(s,x')$. When Expand is called with S_i, a BFS with budget L is executed from x'. The degree of x' is strictly less than L as otherwise $G(V,E')$ contains a star cluster. Hence this BFS tree will contain x. As Expand updates $h(s,x)$ with the minimal available distance it also follows that $h(s,x)=\delta_{E'}(s,x)$. □

The same claim holds for the vertex t and the set T with identical proof. We now show that $S\cap T\neq\emptyset$. Recall that $\delta_{E'}(s,t)=\Delta$ and $d<\Delta\le 2d$. Thus, there is a vertex x such that $\delta_{E'}(s,x)\le d$ and $\delta_{E'}(t,x)\le d$. This implies that $\delta_{E'}(s,x)\le i$ for some $1\le i\le\lceil\frac{1}{\varepsilon}\rceil$ and $\delta_{E'}(t,x)\le j$ for some $1\le j\le\lceil\frac{1}{\varepsilon}\rceil$. By applying Claim 4 for s we get that $x\in S_i$ and $h(s,x)=\delta_{E'}(s,x)$. Similarly, by applying Claim 4 for t we get that $x\in T_j$ and $h(t,x)=\delta_{E'}(t,x)$. Hence $x\in S\cap T$ and the value of $\hat{\delta}(s,t)$ is set by the algorithm to $\delta_{E'}(s,x)+\delta_{E'}(t,x)=\Delta$ □

4.3 Running Time

Next, we analyze the running time of the query algorithm and its space usage. The algorithm produces a $(1+4\varepsilon)$ stretch. In the first stage we scan m/L clusters. In the second stage we have $\log n$ iterations. The cost of each iteration is $O(L^{\lceil\frac{1}{\varepsilon}\rceil})$. If we omit logarithmic factor and balance $O(L^{\lceil\frac{1}{\varepsilon}\rceil})$ with m/L we get that $L=m^{\frac{\varepsilon}{1+2\varepsilon}}$ and the cost of a query is $\tilde{O}(m^{1-\frac{\varepsilon}{1+2\varepsilon}})$. In terms of space the size of the data structure is $O(mn/L)$ and for $L=m^{\frac{\varepsilon}{1+2\varepsilon}}$ it is $O(nm^{1-\frac{\varepsilon}{1+2\varepsilon}})$. If we set $\varepsilon'=4\varepsilon$ we get $1+\varepsilon'$ stretch with query time of $\tilde{O}(m^{1-\frac{\varepsilon'}{4+2\varepsilon'}})$ and space of $O(nm^{1-\frac{\varepsilon'}{4+2\varepsilon'}})$.

5 Concluding Remarks and Open Problems

In this paper we show that it is possible to obtain a multiplicative stretch that is arbitrarily close to 1 with sub-linear space and sub-linear time in sparse graphs. An interesting direction is what can we gain from a non-constant query time for stretch 2 and more. Finally, our clustering technique is natural and simple, thus, it will be interesting to see whether it can be used in other closely related problems such as efficient computation of shortest paths approximation.

References

1. Agarwal, R., Godfrey, P.B.: Distance oracles for stretch less than 2. In: SODA, pp. 526–538 (2013)
2. Bartal, Y., Gottlieb, L., Kopelowitz, T., Lewenstein, M., Roditty, L.: Fast, precise and dynamic distance queries. In: Proc. of 22th SODA, pp. 840–853 (2011)
3. Baswana, S., Kavitha, T.: Faster algorithms for all-pairs approximate shortest paths in undirected graphs. SIAM J. Comput. **39**(7), 2865–2896 (2010)
4. Chen, W., Sommer, C., Teng, S.H., Wang, Y.: Compact routing in power-law graphs. In: Proceedings of the 23rd International Conference on Distributed Computing, DISC'09, pp. 379–391 (2009)
5. Elkin, M.: Computing almost shortest paths. ACM Trans. Algorithms **1**(2), 283–323 (2005)
6. Elkin, M., Peleg, D.: (1 + epsilon, beta)-spanner constructions for general graphs. SIAM J. Comput. **33**(3), 608–631 (2004)
7. Enachescu, M., Wang, M., Goel, A.: Reducing maximum stretch in compact routing. In: INFOCOM, pp. 336–340 (2008)
8. Krioukov, D., Fall, K.R., Yang, X.: Compact routing on Internet-like graphs. In: INFOCOM (2004)
9. Mendel, M., Naor, A.: Ramsey partitions and proximity data structures. J. Eur. Math. Soc. **9**, 253–275 (2007)

10. Pătraşcu, M.: (Data) structures. In: Proc. of 49th FOCS, pp. 434–443 (2008)
11. Pătraşcu, M., Roditty, L.: Distance oracles beyond the Thorup–Zwick bound. In: Proc. of 51st FOCS (2010)
12. Peleg, D., Schäffer, A.A.: Graph spanners. J. Graph Theory 99–116 (1989)
13. Pettie, S.: Low distortion spanners. ACM Trans. Algorithms **6**(1) (2009)
14. Sommer, C., Verbin, E., Yu, W.: Distance oracles for sparse graphs. In: Proc. of 50th FOCS, pp. 703–712 (2009)
15. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM **52**(1), 1–24 (2005)
16. Thorup, M., Zwick, U.: Spanners and emulators with sublinear distance errors. In: Proc. of 17th SODA, pp. 802–809 (2006)