# Approximate Distance Oracles Revisited*

Joachim Gudmundsson[1], Christos Levcopoulos[2], Giri Narasimhan[3], and Michiel Smid[4]

[1] Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. joachim@cs.uu.nl
[2] Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. christos@cs.lth.se
[3] School of Computer Science, Florida International University, Miami, FL 33199, USA. giri@fiu.edu
[4] School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6. michiel@scs.carleton.ca

**Abstract.** Let $G$ be a geometric $t$-spanner in $\mathbb{E}^d$ with $n$ vertices and $m$ edges, where $t$ is a constant. We show that $G$ can be preprocessed in $O(m \log n)$ time, such that $(1+\varepsilon)$-approximate shortest-path queries in $G$ can be answered in $O(1)$ time. The data structure uses $O(n \log n)$ space.

## 1 Introduction

The *shortest-path* (SP) problem for weighted graphs with $n$ vertices and $m$ edges is a fundamental problem for which efficient solutions can now be found in any standard algorithms text. The approximation version of this problem has been studied extensively, see [1, 7–9]. In numerous algorithms, query versions frequently appear as subroutines. In such a query, we are given two vertices and have to compute or approximate the shortest path between them. The latest in a series of results for undirected weighted graphs is by Thorup and Zwick [14]; their algorithm computes $(2k-1)$-approximate solutions to the query version of the SP problem in $O(k)$ time, using a data structure that takes (expected) time $O(kmn^{1/k})$ to construct and utilises $O(kn^{1+1/k})$ space. It is not an approximation scheme in the true sense because the value $k$ needs to be a positive integer. Since the query time is essentially bounded by a constant, Thorup and Zwick refer to their queries as approximate *distance oracles*.

We focus on the geometric version of this problem. A geometric graph has vertices corresponding to points in $\mathbb{R}^d$ and edge weights from a Euclidean metric. Again, considerable previous work exists on the shortest path and related problems. A good survey on the topic was written by Mitchell and can be found in [13], see also [3–6]. Recently Gudmundsson *et al.* [9] presented the first data structure that answers approximate shortest-path distance queries in constant

* J.G. is supported by The Swedish Foundation for International Cooperation in Research and Higher Education and M.S. is supported by NSERC.

time, with $O(n \log n)$ space and preprocessing time for geometric graphs with $O(n)$ edges that are $t$-spanners for some (possibly large) constant $t$. This result is *restricted* in the sense that it only supports queries between vertices in $G$ whose Euclidean distance is at least $D/n^k$, where $D$ is the length of the longest edge in $G$, and $k > 0$ is an arbitrary integer constant. In this paper we extend the results in [9] so that it holds for *any* two query points, i.e., we remove the restriction on the distance between the query points.

A graph $G = (V, E)$ is said to be a *$t$-spanner* for $V$, if for any two points $p$ and $q$ in $V$, there exists a path in $G$ between $p$ and $q$ of length at most $t$ times the Euclidean distance between $p$ and $q$. Given a geometric $t$-spanner $G = (V, E)$ on $n$ vertices and $m$ edges, for some constant $t$, we show how to build a data structure in time $O(m \log n)$ that answers $(1 + \varepsilon)$-approximate shortest path length queries in $G$ in constant time, for any given real constant $\varepsilon > 0$. The structure uses $O(n \log n)$ space.

We remark that many "naturally occurring" geometric graphs are $t$-spanners, for some constant $t > 1$, thus justifying the interest in the restricted inputs considered in this paper. In [9] it was shown that an approximate shortest-path distance (ASD) oracle can be applied to a large number of problems, for example, finding a shortest obstacle-avoiding path between two vertices in a planar polygonal domain with obstacles, interesting query versions of *closest pair* problems and, computing the approximate stretch factor of geometric graphs.

In the full paper we also show that the structure presented in this paper easily can be extended such that $(1 + \varepsilon)$-approximate shortest path queries for geometric $t$-spanners can be answered in time proportional to the number of edges along the path.

## 2 Preliminaries

Our model of computation is the traditional algebraic computation model with the added power of indirect addressing. We will use the following notation. For points $p$ and $q$ in $\mathbb{R}^d$, $|pq|$ denotes the Euclidean distance between $p$ and $q$. If $G$ is a geometric graph, then $\delta_G(p, q)$ denotes the Euclidean length of a shortest path in $G$ between $p$ and $q$. Hence, $G$ is a $t$-spanner for $V$ if $\delta_G(p, q) \leqslant t \cdot |pq|$ for any two points $p$ and $q$ of $V$. A geometric graph $G$ is said to be a $(t, L)$-spanner for $V$ if for every pair of points $p, q \in V$ with $|pq| < L$ it holds that $\delta_G(p, q) \leqslant t \cdot |pq|$. If $P$ is a path in $G$ between $p$ and $q$ having length $\Delta$ with $\delta_G(p, q) \leqslant \Delta \leqslant (1+\varepsilon) \cdot \delta_G(p, q)$, then $P$ is a $(1+\varepsilon)$-approximate shortest path for $p$ and $q$. Finally, a subgraph $G'$ of $G$ is a $t'$-spanner of $G$, if $\delta_{G'}(p, q) \leqslant t' \cdot \delta_G(p, q)$ for any two points $p$ and $q$ of $V$.

The main result of this paper is stated in the following theorem:

**Theorem 1.** *Let $V$ be a set of $n$ points in $\mathbb{R}^d$, let $\varepsilon$ be a positive real constant and let $G = (V, E)$ be a $t$-spanner for $V$, for some real constant $t > 1$, having $O(n)$ edges. We can preprocess $G$ in $O(n \log n)$ time using $O(n \log n)$ space, such that for any two points $p$ and $q$ in $V$, we can compute, in $O(1)$ time, a $(1 + \varepsilon)$-approximation to the shortest-path distance in $G$ between $p$ and $q$.*

If $G$ has $m$ edges, where $m$ grows superlinearly with $n$, then we first apply Theorem 3.1 in [9] to obtain a $(1+\varepsilon)$-spanner of $G$ with $O(n)$ edges. Then we apply Theorem 2 to $G'$. The preprocessing increases by an additive term of $O(m \log n)$ and the space requirement increases by an additive term of $O(m+n)$, but note that the data structure only uses $O(n \log n)$ space. As described in the theorem we will present a data structure that supports $(1+\varepsilon)$-ASD queries in a $t$-spanner graph. As a substructure we will use the data structure in [9] that supports *restricted* $(1+\varepsilon)$-shortest path queries in $t$-spanner graphs. In the rest of this paper we will refer to the data structure in [9] as the PSP-structure and the construction algorithm of the PSP-structure as the PSP-algorithm.

**Fact 2** *(Theorem 4.1 in [9])*
*Let $V$ be a set of $n$ points in $\mathbb{R}^d$, and let $G = (V, E)$ be a $t$-spanner for $V$, for some real constant $t > 1$, having $O(n)$ edges. Let $D$ be the length of the longest edge in $G$, let $\varepsilon$ be a positive real constant, and let $k$ be a positive real constant. We can preprocess $G$ in $O(n \log n)$ time using $O(n \log n)$ space, such that for any two points $p$ and $q$ in $V$ with $|pq| > D/n^k$, we can compute, in $O(1)$ time, a $(1+\varepsilon)$-approximation to the shortest-path distance in $G$ between $p$ and $q$.*

We need a slightly modified version of the above fact, stated in the following corollary. The proof is implicit in [9].

**Corollary 1.** *Let $V$ be a set of $n$ points in $\mathbb{R}^d$, and let $G = (V, E)$ be a $(t, L)$-spanner for $V$, having $O(n)$ edges, for some real constant $t > 1$ and some real number $L$. Let $D$ be the length of the longest edge in $G$, let $\varepsilon$ be a positive real constant, and let $k$ be a positive integer constant. We can preprocess $G$ in $O(n \log n)$ time using $O(n \log n)$ space, such that for any two points $p$ and $q$ in $V$ with $D/n^k < |pq| < L/t$, we can compute, in $O(1)$ time, a $(1+\varepsilon)$-approximation to the shortest-path distance in $G$ between $p$ and $q$.*

In the rest of this paper, we will apply Corollary 1 with a sufficiently large value of $k$.

## 3 A first result

A straightforward idea is to divide the set $E$ of edges of the $t$-spanner $G = (V, E)$ into subsets $E_1, \ldots, E_\ell$, and then build a PSP-structure $P_i$ for each subset $E_i$. In this way, a query can be reduced to the problem of finding an appropriate, to be defined below, PSP-structure. A first step towards our goal is to construct the PSP-structures for each subset of edges and then show that given a query, the correct PSP-structure can be found in constant time.

### 3.1 Building the PSP-structures

For each $i$, $1 \leqslant i \leqslant \ell$, construct the edge set $E_i$ as follows. Initially $E'_1 = E$. Let $e_{\min}$ be the edge in $E'_i$ with smallest weight and let $E_i = \{e \in E'_i | wt(e) \leqslant$

$n^k \cdot wt(e_{\min})\}$. Finally, set $E'_{i+1} = E'_i \backslash E_i$. The resulting edge sets are denoted $E_1, \ldots, E_\ell$. We need some straightforward properties of the edge sets that will be needed for the improved algorithm in Section 4.

**Observation 3** *Given the subsets of edges $E_1, \ldots, E_\ell$, as described above, the following properties hold.*

1. *If $e_1, e_2 \in E_i$ then $wt(e_1)/wt(e_2) \leqslant n^k$.*
2. *If $e_1 \in E_i$ and $e_2 \in E_{i+1}$ then $wt(e_1) < wt(e_2)$.*
3. *If $e_1 \in E_i$ and $e_2 \in E_{i+2}$ then $wt(e_2) \geqslant n^{2k} \cdot wt(e_1)$.*

When the edge sets are constructed, the $\ell$ PSP-structures, $P_1, \ldots, P_\ell$, can be computed as follows. For each $i$, $1 \leqslant i \leqslant \ell$, construct a PSP-structure $P_i$ with the graph $G_i = (\mathcal{V}, \mathcal{E}_i)$ as input, where $\mathcal{V} = V$ and $\mathcal{E}_i = E_1 \cup \ldots \cup E_i$. Since $G_i$ is a $(t, L_i)$-spanner, where $L_i$ is equal to $1/t$ times the maximum edge weight in $\mathcal{E}_i$, we can apply Corollary 1 to $G_i$. Hence, the complexity of building the data structure is $\ell \times O(n \log n)$, according to Corollary 2, which can be bounded by $O(n^2 \log n)$. It remains to show that given the $\ell$ PSP-structures, a $(1+\varepsilon)$-approximate shortest path query can be answered in constant time, i.e., an appropriate PSP-structure can be found in constant time. Given a query pair $(p, q)$, if $i$ is an integer such that $\min_{e \in E_i} |e| \leqslant t \cdot |pq|$ and $|pq| \leqslant n^{k+1} \min_{e \in E_i} |e|$, then the PSP-structure $P_i$ is said to be *appropriate* for the query pair $(p, q)$. Later in the paper, we will see how this notion can be used to answer ASD-queries in the $t$-spanner $G$.

**Observation 4** *For every query $(p, q)$ there is an appropriate PSP-structure $P_i$.*

*Proof.* Since $G$ is a $t$-spanner, there is a path in $G$ of length at most $t \cdot |pq|$. Clearly, this path has length at least $|pq|$, and it contains at most $n$ edges. Let $e'$ be the longest edge on this path. Then $|pq|/n \leqslant wt(e') \leqslant t \cdot |pq|$. Let $i$ be the index such that $e' \in E_i$. Then $\min_{e \in E_i} wt(e) \leqslant wt(e') \leqslant t \cdot |pq|$ and $|pq| \leqslant n \cdot wt(e') \leqslant n^{k+1} \min_{e \in E_i} |e|$.

In Section 4, we will show how to improve the time and space bounds to $O(n \log n)$, by using so-called cluster graphs as input to the PSP-algorithm instead of the entire graphs $G_i$.

### 3.2 Answering ASD-queries

Given a query pair $(p, q)$ it suffices to prove, according to Fact 2 and Observation 4, that the appropriate PSP-structure $P_i$ can be found in constant time. For this we build an approximate single-link hierarchical cluster tree, denoted by $\mathcal{T}$. Given a set $V$ of points, a single-link hierarchical cluster tree can be built as follows. Initially, each point is considered to be a cluster. As long as there are two or more clusters, the pair $C_1$ and $C_2$ of clusters for which $\min\{|pq| : p \in C_1, q \in C_2\}$ is minimum is joined into one cluster. Constructing the single-linkage hierarchy $\mathcal{T}$ for $V$ just means simulating the greedy algorithm of Kruskal [11] for computing the MST of $V$. This works fine in the plane but not in higher dimensions.

Instead we build an approximate single-link hierarchical cluster tree using an $O(1)$-approximation of the MST of $V$, that is, a variant of the MST which can be obtained, as in Kruskal's algorithm, by inserting in each step an almost shortest edge which does not induce a cycle with the edges already inserted. Methods for constructing such MST's are given, for example, in [12].

When the approximate single-link hierarchical cluster tree $\mathcal{T}$ is built we preprocess it in $O(n)$ time, such that lowest common ancestors (LCA) queries can be answered in constant time [10].

**Observation 5** *Let $(u_1, v_1)$ and $(u_2, v_2)$ be two pairs of vertices that have the same least common ancestor $\nu$. Then $wt(u_1, v_1)/wt(u_2, v_2) = O(n)$.*

*Proof.* The observation is trivial since an $O(1)$-approximation of an MST is an $O(n)$-spanner of the complete graph.

The final step of the preprocessing is to associate a PSP-structure to each node in $\mathcal{T}$. Let $\nu$ be an arbitrary node in $\mathcal{T}$ and let $lc(\nu)$ and $rc(\nu)$ be the left- and right child respectively of $\nu$. Every node $\nu$ in $\mathcal{T}$ represents a cluster $C$. Associated to $\nu$ is also a value $d(\nu)$, i.e., the length of the edge connecting $C_1$ and $C_2$ where $C_1$ is the cluster represented by $lc(\nu)$ and $C_2$ is the cluster represented by $rc(\nu)$. According to Observation 5, this value is an $O(n)$-approximation of the distance between any pair of points $(p, q)$ where $p \in C_1$ and $q \in C_2$. For each node $\nu$ in $\mathcal{T}$ we add a pointer to the appropriate PSP-structure. By using binary search on the $\ell$ PSP-structures one can easily find the appropriate PSP-structure in $O(\log n)$ time, hence in $O(n \log n)$ time in total.

Now we are finally ready to handle a query. Given a query $(p, q)$, compute the LCA $\nu$ of $p$ and $q$ in $\mathcal{T}$. The LCA-query will approximate the length of $(p, q)$ within a factor of $O(n)$, according to the above observation, which is sufficient for our needs. Hence, $\nu$ will have a pointer to a PSP-structure $P_j$. One of the PSP-structures $P_{j-1}$ or $P_j$ will be an appropriate PSP-structure, this can be decided in constant time. Finally the query is passed on to the appropriate PSP-structure that answers the query in constant time.

We can now summarize the results presented in this section.

**Lemma 1.** *Let $V$ be a set of $n$ points in $\mathbb{R}^d$, and let $G = (V, E)$ be a $t$-spanner for $V$, for some real constant $t > 1$, having $O(n)$ edges. We can preprocess $G$ in $O(n^2 \log n)$ time and space, such that for any two points $p$ and $q$ in $V$, we can compute, in $O(1)$ time, a $(1 + \varepsilon)$-approximation to the shortest-path distance in $G$ between $p$ and $q$, where $\varepsilon$ is a positive real constant.*

This is hardly impressive since using Dijkstra's algorithm to compute all-pairs-shortest-paths and then saving the results in a matrix would give a better result. In the next section we will show how to improve both the time and space used to $O(n \log n)$.

## 4 Improving the complexity

By considering the above construction, one observes that the main reason why the complexity is so high, is the size of the $\ell$ graphs constructed as input to the

PSP-algorithm. The trivial bound we obtain is that each of the $\ell$ graphs has size $O(n)$ and $\ell \leqslant n$, thus resulting in an overall $O(n^2 \log n)$ time and space bound. A way to improve the complexity would be to construct "sparser" graphs, for example cluster-graphs. This will improve both the preprocessing time and the space complexity to $O(n \log n)$, as will be shown below. Since each input graph will be a subset of the original input graph, this will complicate the construction and the query structure considerably. This section is organized as follows. First we show how to construct the $\ell$ cluster graphs, then it is shown that the total space complexity of the input graphs can be bounded by $O(n \log n)$. Finally we consider the problem of answering queries. The correctness of the data structure is proven and finally it is shown how to answer a query in constant time.

### 4.1 Constructing the cluster-graphs

The aim of this section is to produce the $\ell$ graphs $G_1, \ldots, G_\ell$ that will be the input to the PSP-algorithms. We define the notions of *cluster* and *cluster graph* as follows. Given a graph $G = (V, E)$, a cluster $C$ with cluster center at $v$ is a maximal set of vertices of $V$ such that $\delta_G(v, u) < \infty$ for every vertex $u$ in $C$. Let $cc(u)$ denote the cluster center of the cluster that $u$ belongs to.

A cluster graph $H_{G,E'}$, of a graph $G = (V, E)$ and a set of edges $E'$, has vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ which are defined as follows. Let $C = \{C_1, \ldots, C_b\}$ be a minimal set of clusters of $G$. For every edge $(u, v) \in E'$ there is an edge $(cc(u), cc(v)) \in \mathcal{E}$, with weight $|cc(u), cc(v)|$, if and only if $u$ and $v$ are in different clusters. The vertex set $\mathcal{V}$ is the set of cluster centers of $C$ adjacent to at least one edge in $\mathcal{E}$. The cluster graph can be constructed in linear time with respect to the complexity of $G$ and the number of edges in $E'$. Below we describe, using pseudo code, how this can be done.

**ComputeClusterGraph**$(G = (V, E), E')$
1. **for** each edge $(u, v) \in E'$ **do**
2. add edge $(cc(u), cc(v))$ with weight $|cc(u), cc(v)|$ to $\mathcal{E}$
3. **while** $V$ not empty **do**
4. pick an arbitrary vertex $v \in V$
5. remove every vertex $u$ from $V$ for which $\delta_G(v, u) < \infty$
6. **if** $\exists u \in V$ such that $(v, u) \in \mathcal{E}$ **then**
7. add $v$ to $\mathcal{V}$
8. **output** $H = (\mathcal{V}, \mathcal{E})$

Following the above definition of a cluster graph it is now easy to construct the $\ell$ cluster graphs $G_1, \ldots, G_\ell$ which will be the input graphs for the construction of the $\ell$ PSP-structures, $P_1, \ldots, P_\ell$. Let $G_1 = (V, E_1)$ and $G_2 = (V, E_1 \cup E_2)$. For each value of $i$, $3 \leqslant i \leqslant \ell$, the input graph $G_i = (\mathcal{V}_i, \mathcal{E}_i)$ is the cluster graph of $G_{i-2}$ and the edge set $(E_{i-1} \cup E_i)$.

**Lemma 2.** *The $\ell$ PSP-structures uses $O(n \log n)$ space and can be constructed in time $O(n \log n)$.*

*Proof.* One PSP-structure $P_i$ uses $O((|\mathcal{V}_i| + |\mathcal{E}_i|)\log(|\mathcal{V}_i| + |\mathcal{E}_i|))$ preprocessing time and space, according to Corollary 2. So the total time and space needed is:

$$\sum_{i=1}^{\ell}(|\mathcal{V}_i| + |\mathcal{E}_i|)\log(|\mathcal{V}_i| + |\mathcal{E}_i|). \tag{1}$$

We know from the above definition of $\mathcal{V}_i$ and $\mathcal{E}_i$ that $|\mathcal{V}_i| \leqslant 2|E_i \cup E_{i-1}|$ and that $|\mathcal{E}_i| = |E_i \cup E_{i-1}|$. Hence $|\mathcal{V}_i| + |\mathcal{E}_i| \leqslant 3|E_i \cup E_{i-1}|$. The input graph $G$ has $|E| = O(n)$ edges, which implies that $\sum_{i=1}^{\ell}(|\mathcal{V}_i| + |\mathcal{E}_i|) \leqslant 3\sum_{i=1}^{\ell}|E_i \cup E_{i-1}| = O(n)$ and, hence, (1) will sum up to $O(n \log n)$. The time to compute the $\ell$ cluster graphs is linear with respect to the total input size, hence a total time of $O(n \log n)$.

**PreProcessing**$(G = (V, E), \varepsilon)$
1. construct $E_1, \ldots, E_\ell$
2. $G_1 := (V, E_1)$, $G_2 := (V, E_1 \cup E_2)$
3. **for** $i := 3$ to $\ell$ **do**
4. $G_i :=$ComputeClusterGraph$(G, E_{i-1} \cup E_i)$
5. $P_i :=$ConstructPSP$(G_i, \varepsilon')$
5. construct an approximate single-linkage hierarchy $\mathcal{T}$ of $V$
6. process $\mathcal{T}$ to allow efficient LCA-queries
7. construct cluster tree $L$ of $G$
8. process $L$ to allow efficient level-ancestor queries

Next it will be shown that given a query $(p, q)$ there always exists an appropriate PSP-structure that answers $(1 + \varepsilon)$-ASD queries. Finally, in Section 4.3, we show how this appropriate PSP-structure can be found in constant time.

### 4.2 There exists an appropriate PSP-structure

Assume that we are given a spanner $G = (V, E)$, a $(1 + \varepsilon)$-ADS query $(p, q)$, and the appropriate PSP-structure $P_i$ for $(p, q)$. As the observant reader already noticed, $p$ and/or $q$ may not be vertices in $\mathcal{V}_i$, which implies that we cannot query $P_i$ with $(p, q)$. Instead it will be shown that querying $P_i$ with $(cc(p), cc(q))$ will help to answer the $(1 + \varepsilon)$-ASD query on $G$. Let $G_i$ be the cluster graph given as input to the PSP-algorithm that constructed $P_i$.

**Observation 6** *The diameter of a cluster in $G_i$ is at most $|pq|/n^{2k-1}$.*

*Proof.* According to the requirements the ratio between edge weights in $E_i$ and edge weights in $E_{i-2}$ is at least $n^{2k}$, and since the longest path in a graph contains at most $n$ edges, we know that the diameter of a cluster in $G_i$ is at most a factor $1/n^{2k-1}$ times longer than the shortest edge in $E_i$.

The following observation can now be derived.

**Observation 7** *Given a $(1+\varepsilon)$-ASD query $(p,q)$ on a graph $G$ and a cluster graph $G_i$, where $P_i$ is the appropriate PSP-structure for $(p,q)$ and $G$ constructed with $G_i$ as input, we have*

$$\delta_{G_i}(cc(p),cc(q)) - \frac{2|pq|}{n^{2k-2}} \leqslant \delta_G(p,q) \leqslant \delta_{G_i}(cc(p),cc(q)) + \frac{2|pq|}{n^{2k-2}}.$$

*Proof.* Let $L_1$ the path in $G$ between $p$ and $q$ having weight $\delta_G(p,q)$. We shall use the notation $L_1(y,x)$ to denote the vertices of $L_1$ between vertices $y$ and $x$, not including $y$. We construct a cluster path $L_2$ from $p$ to $q$ in $G_i$ as follows. Let $C_0$ be the cluster with cluster center $v_0 := cc(p)$. Among all vertices adjacent to $v_0$ in $G_i$, let $v_1$ be the cluster center whose cluster $C_1$ intersects $C_0$ in the furthest vertex, say $w_1$, along $L_1(p,q)$. Add the edge $(v_0,v_1)$ to $L_2$. Next, among all vertices adjacent to $v_1$ in $G_i$, let $v_2$ be the cluster center whose cluster $C_2$ intersects $C_1$ in the furthest vertex, say $w_2$, along $L_1(w_1,v)$. Add the edge $(v_1,v_2)$ to $L_2$. This process continues until we reach $cc(q) = v_m$. The two paths are illustrated in Figure 1.

From Observation 6, the difference in length between $L_1$ and $L_2$ within one cluster is bounded by $2|pq|/n^{2k-1}$. Since a shortest path can visit at most $n$ clusters and at most $n$ vertices, it follows that the error is bounded by $n \cdot 2|pq|/n^{2k-1} = 2|pq|/n^{2k-2}$, hence the observation follows.

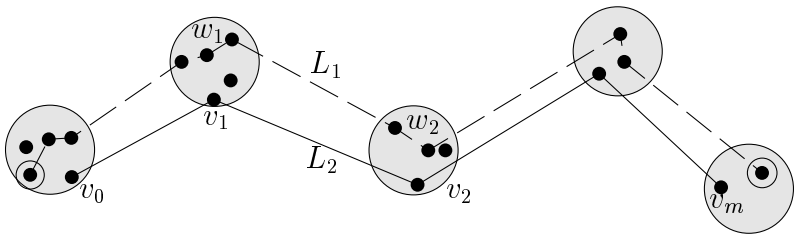


**Fig. 1.** Illustrating the two paths $L_1$ and $L_2$.

**Observation 8** *For every graph $G$ and every query $(p,q)$ there is an appropriate PSP-structure $P_i$.*

*Proof.* There is a path in $G$ between $p$ and $q$ of length at most $t \cdot |pq|$. This path has length at least $|pq|$, and contains at most $n$ edges. Hence, the longest edge, say $e'$ on this path satisfies $|pq|/n \leqslant wt(e') < t \cdot |pq|$. This means that there is a set $E_i$ such that $\min_{e \in E_i} |e| \leqslant t \cdot |pq|$ and $|pq| \leqslant n^{k+1} \min_{e \in E_i} |e|$.

We summarize this section with the following lemma:

**Lemma 3.** *A $(1+\varepsilon)$-ADS query $(p,q)$ on $G$ can be answered by returning the value $\delta + \frac{2|pq|}{n^{2k-2}}$, where $\delta$ is the answer obtained by performing a $(1+\varepsilon')$-ADS*

*query* $(cc(p), cc(q))$ *on the appropriate PSP-structure for* $(p, q)$ *on* $G$*, where* $\varepsilon' \leqslant \varepsilon - \frac{\varepsilon+4}{n^{2k-2}}$.

*Proof.* Assume that we are given a $(1+\varepsilon)$-ASD query $(p, q)$ on $G$. According to Observation 8 there is an appropriate PSP-structure $P_i$ for $(p, q)$ and $G$. Now assume that we perform a $(1+\varepsilon')$-ASD query $(cc(p), cc(q))$ on $P_i$ where we let $\delta$ denote the answer of the query. According to Observation 7, we have $\delta_G(p, q) - \frac{2|pq|}{n^{2k-2}} \leqslant \delta_{G_i}(cc(p), cc(q)) \leqslant \delta \leqslant (1+\varepsilon') \cdot \delta_{G_i}(cc(p), cc(q)) \leqslant (1+\varepsilon') \cdot (\delta_G(p, q) - \frac{2|pq|}{n^{2k-2}})$, which gives

$$\delta_G(p, q) \leqslant \delta + \frac{2|pq|}{n^{2k-2}} \leqslant (1+\varepsilon') \cdot \delta_{G_i}(cc(p), cc(q)) + 4(1+\varepsilon')\frac{2|pq|}{n^{2k-2}}.$$

Now choose $\varepsilon' \leqslant \varepsilon - \frac{\varepsilon+4}{n^{2k-2}}$. If a $(1+\varepsilon)$-ASD query $(p, q)$ is performed on $G$ we can answer the query with the value $\delta + \frac{2|pq|}{n^{2k-2}}$ where $\delta$ is obtained by performing a $(1+\varepsilon')$-ADS query $(cc(p), cc(q))$ on the appropriate PSP-structure $P_i$ for $(p, q)$.

The result of this section is that if the appropriate PSP-structure for $G$ and $(p, q)$ is $P_i$ then the $(1+\varepsilon)$-ASD query $(p, q)$ on $G$ can be replaced by a $(1+\varepsilon')$-ADS query $(cc(p), cc(q))$ on $G_i$ with $\varepsilon' \leqslant \varepsilon - \frac{\varepsilon+4}{n^{2k-2}}$. By choosing $k$ equal to 2, and assuming that $n$ is sufficiently large, we can take $\varepsilon' = \varepsilon/2$.

### 4.3 Answering ASD-queries

Above it was shown that a $(1+\varepsilon)$-ASD query $(p, q)$ can be answered provided that the appropriate PSP-structure and the cluster centers of $p$ and $q$ are found. In this section we will first show how the appropriate PSP-structure $P_i$ can be found in constant time and finally we show how the cluster centers of $p$ and $q$ can be found in constant time.

Build an approximate single-link hierarchical cluster tree, denoted $\mathcal{T}$, as described in Section 3.2. The tree is then preprocessed such that LCA-queries can be answered in constant time [10]. The final step of the preprocessing is to associate a PSP-structure to each node in $\mathcal{T}$, see Section 3.2.

Given a query $(p, q)$ compute the LCA $\nu$ of $p$ and $q$ in $\mathcal{T}$. The LCA-query will approximate the length of $(p, q)$ within a factor of $O(n)$, according to Observation 5, which is sufficient for our needs. Hence, $\nu$ will have a pointer to a PSP-structure $P_i$ and one of the PSP-structures $P_{i-1}$ or $P_i$ will be the appropriate PSP-structure, which one it is can be decided in constant time.

*Finding the cluster centers.* Next the query should be passed on to the appropriate PSP-structure. It might be that $p$ and/or $q$ are not points in $\mathcal{V}_i$. Recall that $\mathcal{V}_i$ is the set of input points for the appropriate PSP-structure $P_i$. Hence, it remains to show that one can find the vertex $cc(p)$ in $\mathcal{V}_i$ that is the cluster center of a query point $p$ in constant time. For this purpose we build the cluster tree $L$. Let $r$ be the root of $L$. The parent of a node $v$ in $L$ is denoted $parent(v)$. The depth of $v$ is $d(v) = d(parent(v)) + 1$, and $d(r) = 0$. The tree has $\ell + 1$ levels

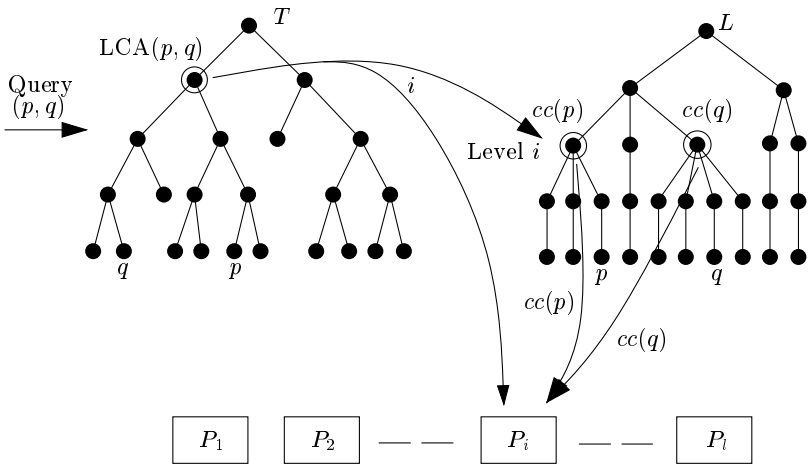


**Fig. 2.** Processing a query $(p, q)$.

where the number of nodes at level $1 \leqslant j \leqslant \ell$ is $|\mathcal{V}_{\ell-j+1}|$. At level $j$ in $L$, every node $\mu$ corresponds to a vertex $v$ in $\mathcal{V}_{\ell-j+1}$, hence at leaf level, level $\ell$, there are $n$ nodes (the same holds for level $\ell-1$). It holds that $parent(\mu)$ is the node $\mu'$ at level $j-1$ that corresponds to the cluster center $cc(v)$ in $G_{\ell-j}$. To obtain a tree we finally add a root at level 0 that connects the nodes at level 1. The cluster tree $L$ can be computed in linear time w.r.t. the number of nodes, and according to the proof of Lemma 2 this is $O(n)$, hence, in total time $O(n \log n)$.

Note that finding the cluster center $cc(p)$ in $G_i$ of a point $p$ in $G$ is equivalent of finding the ancestor of the leaf $p$ at level $j = \ell - i + 1$ in $L$. We will use a modified variant of the data structure presented by Alstrup and Holm [2]. The structure answers level ancestor queries in constant time. Given a leaf $v$ of a tree $T$ and a level $l$ of $T$, the level ancestor of $v$ and $l$ is the ancestor of $v$ at level $l$. Note that if we query the cluster tree $\mathcal{T}$ with a vertex $v$ and a level $j$ the data structure would return the ancestor of $v$ at level $j$ which is the cluster center of $v$ in the cluster graph $G_{\ell-i+1}$.

We briefly describe the modified version of the data structure by Alstrup and Holm [2].

*Finding level ancestor* Assume that we are given a tree $L$ with a total of $N$ nodes. A query $LevelAncestor(x, l)$ returns the ancestor $y$ to $x$ at level $l$ in $L$. The number of nodes in the subtree rooted at $v$ is denoted $s(v)$. The notation $d|a$ means that $a = k \cdot d$ for some positive integer $k$. Define the *rank* of $v$, denoted $r(v)$ to be the maximum integer $i$ such that $2^i | d(v)$ and $s(v) \geqslant 2^i$. Note that the rank of the root will be $\lfloor \log_2 n \rfloor$.

**Fact 9** *(Observation 1 in [2]) The number of nodes of rank$\geqslant i$ is at most $\lfloor N/2^i \rfloor$.*

The preprocessing algorithm consists of precomputing the depth, size and rank of each node in the tree and then constructing the following three tables:

`logtable`$[x]$: contains the value $\lfloor \log_2 x \rfloor$, for $0 \leqslant x \leqslant N$.
`levelanc`$[v][x]$: contains the $x$'th ancestor to $v$, for $0 \leqslant x \leqslant 2^{r(v)}$.
`jump`$[v][i]$: contains the first ancestor to $v$ whose depth is divisible by $2^i$, for $0 \leqslant i \leqslant \log_2(d(v)+1)$.

The idea is then to use the `jump[][]` table a constant number of times in order to reach a node $w$ with a sufficiently large rank to hold the answer to the level ancestor query in its table. The pseudocode looks as follows:

**LevelAncestor**$(v, x)$
1. $i :=$`logtable`$[x+1]$
2. $d := d(v) - x$
3. **while** $2(d(v)-d) \geqslant 2^i$ **do**
4. $v :=$`jump[v][i-1]`
5. **return** `levelanc[v][d(v)-d]`

**Corollary 2.** *(Modified version of Lemma 2 in [2])*
*A tree $T$ with $N$ nodes can be preprocessed in $O(N \log N)$ time and space allowing level ancestor queries to be answered in constant time.*

Since the size of the $L$ is $O(n)$ we get that $L$ can be constructed in total time $O(n \log n)$ and then processed in time $O(n \log n)$ using $O(n \log n)$ space such that level ancestor-queries can be answered in constant time.

**ASD-query**$(p, q)$
1. $\nu :=$LCA$(\mathcal{T}, p, q)$
2. $P_i := \nu$.PSP-structure
3. $cc(p) :=$LevelAncestor$(L, p, i)$
4. $cc(q) :=$LevelAncestor$(L, q, i)$
5. $\delta :=$QueryPSP$(P_i, \varepsilon')$
6. **return** $(\delta + \frac{2|pq|}{n^{k-2}})$

The following lemma concludes this section.

**Lemma 4.** *We can preprocess a graph $G = (V, E)$ and a set of PSP-structures of $G$ in time $O(n \log n)$ using $O(n \log n)$ space such that for any two points $p, q \in V$, we can compute in constant time the appropriate PSP-structure $P$ for $(p, q)$ and $G$, and the cluster centers of $p$ and $q$ in the cluster graph corresponding to $P$.*

Putting together the results in Lemmas 2, 3 and 4 gives us Theorem 1.

## 5 Concluding remarks

We have presented a data structure which supports $(1+\varepsilon)$-approximate shortest distance queries in constant time for geometric $t$-spanners, hence functions as an approximate distance oracle. This generalises the ASD-oracle for restricted queries presented by the authors in [9] to arbitrary queries. It has been shown that the data structure can be applied to a large number of problems, for example, closest pair queries, shortest paths among obstacles, computing stretch factors of a geometric graph, and so on. We believe that the techniques used are of independent interest.

In the full paper we also show that the structure easily can be extended such that $(1+\varepsilon)$-approximate shortest path queries for geometric $t$-spanners can be answered in time proportional to the number of edges along the path.

## References


1. D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.
2. S. Alstrup and J. Holm, Improved algorithms for finding level ancestors in dynamic trees. In Proc. *27th International Colloquium on Automata, Languages, and Programming*, 2000.
3. S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In Proc. *4th European Symposium on Algorithms*, LNCS 1136, pp. 514–528, 1996.
4. D. Z. Chen. On the all-pairs Euclidean short path problem. In Proc. *6th ACM-SIAM Symposium on Discrete Algorithms*, pp. 292–301, 1995.
5. D. Z. Chen, K. S. Klenk, and H.-Y. T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM Journal on Computing*, 29:1223–1246, 2000.
6. Y.-J. Chiang and J. S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. In Proc. *10th ACM-SIAM Symposium on Discrete Algorithms*, 1999.
7. E. Cohen. Fast algorithms for constructing $t$-spanners and paths with stretch $t$. *SIAM Journal on Computing*, 28:210–236, 1998.
8. D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29:1740–1759, 2000.
9. J. Gudmundsson, C. Levcopoulos, G. Narasimhan and M. Smid. Approximate Distance Oracles for Geometric graphs. In Proc. *13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.
10. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355, 1984.
11. J. B. Kruskal, Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Amer. Math. Soc. 7(1956):48–50, 1956.
12. D. Krznaric, C. Levcopoulos and B. J. Nilsson. Minimum Spanning Trees in $d$ Dimensions. *Nordic Journal of Computing*, 6(4):446–461, 1999.
13. J. S. B. Mitchell. Shortest paths and networks. In *Handbook of Discrete and Computational Geometry*, pp. 445–466. CRC Press LLC, 1997.
14. M. Thorup and U. Zwick. Approximate distance oracles. In Proc. *33rd ACM Symposium on Theory of Computing*, 2001.