

Approximate Distance Oracles for Geometric Spanners

Joachim Gudmundsson^{1*}, Christos Levkopoulos², Giri Narasimhan³, and Michiel Smid⁴

¹ Department of Mathematics and Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands h.j.gudmundsson@TUE.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, Ottawa, Canada K1S 5B6. michiel@scs.carleton.ca

Abstract. Given an arbitrary real constant $\varepsilon > 0$, and a geometric graph G in d -dimensional Euclidean space with n points, m edges, and constant dilation, we present a data structure that answers $(1 + \varepsilon)$ -approximate shortest path length queries in constant time. The data structure can be constructed in $O(m + n \log n)$ time using $O(n \log n)$ space. This represents the first data structure that answers $(1 + \varepsilon)$ -approximate shortest path queries in constant time, and hence functions as an approximate distance oracle. The data structure is also applied to several other problems. In particular, we also show that approximate shortest path queries between vertices in a planar polygonal domain with “rounded” obstacles can be answered in constant time. Other applications include query versions of *closest pair* problems, and the efficient computation of the approximate dilations of geometric graphs.

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. In the *query* version of the problem, one is allowed to preprocess the graph, so that the shortest path (either the length, or the actual path itself) between two given vertices can be efficiently reported. In numerous algorithms, the query versions of the problem appear as subroutines. Furthermore, it is often sufficient to solve the problem approximately. The approximation version of the shortest path problem has been studied extensively; see [1, 15, 18]. The latest in a series of results for undirected weighted graphs is by Thorup and Zwick [32]; their algorithm computes $(2k - 1)$ -approximate solutions to the query version of the SP problem in $O(k)$ time, using a data structure that can be constructed in $O(kmn^{1/k})$ expected time with $O(kn^{1+1/k})$ 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*. For the unweighted case, the preprocessing time was recently improved by Baswana and Sen [5] to $\tilde{O}(n^2)$.

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. Throughout this paper, we will assume that d is constant. Again, considerable previous work exists on the geometric shortest path and related problems. A survey can be found in [27], see also [2, 11, 12, 14].

We consider geometric graphs that are t -spanners for some constant $t > 1$. 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

* J.G. was supported by the Netherlands Organisation for Scientific Research (NWO) and M.S. was supported by NSERC.

and q . The minimum value t such that G is a t -spanner for V is called the *dilation* of G . Given a geometric t -spanner $G = (V, E)$ on n vertices and m edges, we consider the problem of constructing a data structure that supports $(1 + \varepsilon)$ -approximate shortest path queries, for any given real constant $\varepsilon > 0$. We present an algorithm that preprocesses G in $O(m + n \log n)$ time, after which shortest path length queries can be answered in constant time.

We remark that many “naturally occurring” geometric graphs are t -spanners, for some constant $t > 1$, thus justifying the interest in the problem considered in this paper. Many theoretical geometric graphs as well as practical networks are known to be t -spanners. For example, for any point set in the plane, the Delaunay triangulation, the greedy triangulation, and the minimum weight triangulation are t -spanners for some constant $t > 1$, see [25, 16]. As a more practical example, the railroad network in southern Scandinavia is a 1.85-spanner. As a further motivation, many algorithms for constructing t -spanners of point sets have been published. Even though any pair of points is connected by a t -spanner path in such a spanner, it is often far from obvious how to actually compute such a path.

Previous work on the geometric case has dealt with visibility graphs arising in polygonal domains with or without polygonal obstacles, and for various metrics. Note that none of the existing algorithms answer shortest path length queries in constant time with subquadratic space and preprocessing time.

The main data structure used is a series of “cluster graphs” of increasing coarseness each of which helps answer queries for different size edges. The idea of using cluster graphs to speed up geometric algorithms was first introduced by Das and Narasimhan [17] and later used by Gudmundsson *et al.* [20] to design an efficient algorithm to compute $(1 + \varepsilon)$ -spanners. More recently, similar ideas have been used by Gao *et al.* [19].

Another contribution of this paper is a technical tool to “bucket” edge lengths in constant time, without using the floor function. This tool was used in our algorithm in order to decide which cluster graph to perform our search in. We believe that this tool is of independent interest as a general device to speed up geometric algorithms.

We also show that the main data structure presented has many applications. We consider the problem of approximating a shortest obstacle-avoiding path between two vertices in a planar polygonal domain with obstacles, having a total of n vertices. In accordance with the spanner constraints imposed above, we consider restricted planar polygonal domains, which we will refer to as t -rounded domains. A polygonal domain P is said to be t -rounded if for any two vertices p and q , the length of the shortest obstacle-avoiding path between p and q is at most t times the Euclidean distance between them. The concept of t -roundedness is similar in spirit to that of domains with *fat* obstacles or with bounded *aspect ratio* obstacles, see [7, 29]. We show how to preprocess a planar t -rounded domain in $O(n \log n)$ time, such that approximate shortest path length queries can be solved in constant time for any two vertices of the domain, and in $O(\log n)$ time for any two points in the plane. Other applications include several query versions of *closest pair* problems, and a significant improvement in the time complexity of an algorithm to compute the approximate dilation of geometric graphs.

Our model of computation is the traditional algebraic computation tree model with the added power of indirect addressing. We will use the following notation. For points p and q in \mathbb{R}^d , let $|pq|$ denote the Euclidean distance between p and q . For a geometric graph G , let $\delta_G(p, q)$ denote the Euclidean length of a shortest path in G between p and q . Hence, G is a t -spanner for V if $\delta_G(p, q) \leq t|pq|$ for any two points p and q of V . If P is a path in G between p and q having length Δ with $\delta_G(p, q) \leq \Delta \leq (1 + \varepsilon)\delta_G(p, q)$, then P is said to be a

$(1 + \varepsilon)$ -approximate shortest path for p and q . Finally, we say that a subgraph G' of G is a t' -spanner of G , if $\delta_{G'}(p, q) \leq t' \cdot \delta_G(p, q)$ for any two points p and q of V .

In Section 2, we present a data structure that answers approximate distance queries in a spanner G for points p and q for which the ratio between $|pq|$ and the length of the longest edge in G is polynomially bounded. We present the data structure and describe its properties. In Section 3, the result is extended to arbitrary edge lengths. In Section 4, we discuss various applications of our data structure, including answering approximate shortest path-length queries in polygonal domains with obstacles, computing dilations of geometric graphs, and answering closest pair queries. Finally, in Section 5, we present a tool for bucketing edge lengths without using the floor function. This tool is used repeatedly in Sections 2-3.

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 , and let $G = (V, E)$ be a t -spanner for V , for some real constant $t > 1$, having m edges. We can preprocess G into a data structure of size $O(n \log n)$ in time $O(m + n \log n)$, such that for query points $p, q \in V$, we can compute a $(1 + \varepsilon)$ -approximation of the shortest-path distance in G between p and q in constant time.*

2 Answering approximate distance queries for points that are separated

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 known real constant $t > 1$. We assume that the number of edges of G is $O(n)$. If this is not the case, then we can use the algorithm in [22] to compute, in $O(m + n \log n)$ time, a linear-sized $(1 + \varepsilon)$ -spanner of G , which can then be used to answer the queries, instead of G .

Let D be the length of the longest edge in G , let ε be a positive real constant, and let c be a positive integer constant. In this section, we will show how to preprocess G into a data structure such that for any two points p and q in V with $|pq| > D/n^c$, we can compute in constant time a $(1 + \varepsilon)$ -approximation of the shortest-path distance in G between p and q , i.e., a real number Δ such that

$$\delta_G(p, q) \leq \Delta \leq (1 + \varepsilon) \cdot \delta_G(p, q).$$

This data structure will be based on the cluster-based implementation of the greedy spanner algorithm by Das and Narasimhan [17]; see also Gudmundsson *et al.* [20]. Therefore, we start by describing the main ideas of this algorithm.

2.1 Constructing a spanner using a greedy approach

We may assume without loss of generality that $D = n^c$. If this is not the case, then we scale the coordinates of the points of V so that this is true. The greedy spanner algorithm computes a $(1 + \varepsilon)$ -spanner $G' = (V, E')$ of the t -spanner $G = (V, E)$. The basic version of this algorithm works as follows. The graph G' is initialized to (V, \emptyset) , and the edges of E are sorted according to their lengths. Then, the algorithm examines the edges of E one by one, in increasing order. When the algorithm examines an edge (p, q) , it checks if there is a path between p and q in the current graph G' whose length is at most $(1 + \varepsilon)|pq|$; the edge (p, q) is added to G' if and only if there is no such path. The resulting graph G' is a $(1 + \varepsilon)$ -spanner of G .

Since a naïve implementation requires cubic time we need a faster algorithm. Das and Narasimhan [17] showed how one can use so-called cluster graphs to speed up the algorithm.

For this we need to choose arbitrary real constants μ and λ with $1 \leq \lambda < \mu$, and partition the edge set E into subsets

$$E_0 := \{(p, q) \in E : |pq| \leq \lambda\}$$

and

$$E_i := \{(p, q) \in E : \mu^{i-1}\lambda < |pq| \leq \mu^i\lambda\},$$

$1 \leq i \leq 1 + \lfloor \log_\mu(n^c/\lambda) \rfloor$. After the graph G' has been initialized to (V, E_0) , the remaining subsets E_i , $i \geq 1$, are processed one after another. At the beginning of the phase in which E_i is processed, the algorithm constructs a *cluster graph* H of the current graph G' , which is dynamically maintained during this phase. Informally, the cluster graph H approximates the current graph G' , in the sense that for any two points p and q for which $|pq|$ is approximately equal to $\mu^i\lambda$, the value of $\delta_H(p, q)$ approximates $\delta_{G'}(p, q)$. Moreover, for two such points p and q , the value of $\delta_H(p, q)$ can be computed in $O(1)$ time, since the degree in H is bounded by a constant and the number of edges in H on the shortest path between p and q is also bounded by a constant. When the algorithm processes edge (p, q) of E_i , it uses the current cluster graph H to decide whether or not to add this edge to the current graph G' . That is, (p, q) is added to G' if and only if $\delta_H(p, q) > (1 + \varepsilon)|pq|$. If this edge is added, then H is updated so that it is a valid cluster graph for the new graph G' . Das and Narasimhan proved that the final graph G' (i.e., after all subsets E_i , $i \geq 1$, have been processed) is a $(1 + \varepsilon)$ -spanner of G . They also show that for each i , $1 \leq i \leq 1 + \lfloor \log_\mu(n^c/\lambda) \rfloor$, the algorithm spends $O(n \log n + |E_i|)$ time to process the edge set E_i . Hence the overall algorithm runs in $O(n \log^2 n + \sum_{i \geq 1} |E_i|) = O(n \log^2 n)$ time. We now present some properties of the cluster graph in more detail.

The cluster graph Let $L > 1$ be a real number, and consider the graph G' at the moment when the greedy algorithm has just processed all edges of E whose length is at most L . In [17] Das and Narasimhan prove that $\delta_{G'}(x, y) \leq (1 + \varepsilon)|xy|$, for all edges $(x, y) \in E$ with $|xy| \leq L$. Let i be the integer such that $\mu^{i-1}\lambda < L \leq \mu^i\lambda$. They choose an arbitrary real constant⁵ α with $0 < \alpha < 1/4$, and use it to define the cluster graph H for G' . This cluster graph has the points of V as its vertices, and it has $O(n)$ edges. Moreover, for all $p, q \in V$

1. $\delta_{G'}(p, q) \leq \delta_H(p, q)$.
2. If $\delta_{G'}(p, q) > (1 - 2\alpha)\mu^{i-1}\lambda$ then,

$$\delta_H(p, q) \leq \left(\frac{1 + 6\alpha}{1 - 2\alpha} \right) \delta_{G'}(p, q).$$

3. For any real number ℓ with $0 < \ell \leq (1 + \varepsilon)\mu^i\lambda$, it takes $O(1)$ time to decide if $\delta_H(p, q) \leq \ell$.

2.2 Using the cluster graph for approximate shortest path queries

Let p and q be two points of V such that $|pq| > \lambda$, and let i be the positive integer such that $\mu^{i-1}\lambda < |pq| \leq \mu^i\lambda$. Consider the graph G' and the corresponding cluster graph H at the moment when the greedy algorithm has just processed all edges of E_i .

Assume that we have chosen the constant μ such that $\mu > t$. Furthermore, assume that $|pq| \leq \mu^i\lambda/t$. Then the following holds.

⁵ Das and Narasimhan use the notation δ instead of α .

- $\delta_G(p, q) \leq \delta_{G'}(p, q)$. This holds because G' is a subgraph of G .
- $\delta_{G'}(p, q) \leq \delta_H(p, q)$. This is one of the general properties of the cluster graph, see Section 2.1.
- $\delta_H(p, q) \leq \left(\frac{1+6\alpha}{1-2\alpha}\right) \cdot \delta_{G'}(p, q)$. Since $\delta_{G'}(p, q) \geq |pq| > \mu^{i-1}\lambda > (1-2\alpha)\mu^{i-1}\lambda$, this follows from the general properties of the cluster graph, see Section 2.1.
- $\delta_{G'}(p, q) \leq (1+\varepsilon) \cdot \delta_G(p, q)$. To prove this, first observe that $\delta_G(p, q) \leq t|pq| \leq \mu^i\lambda$. Hence, each edge on the shortest path in G between p and q has length at most $\mu^i\lambda$. The claim then follows from the general properties of the cluster graph, see Section 2.1.

Combining these inequalities, it follows that

$$\delta_G(p, q) \leq \delta_H(p, q) \leq (1+\varepsilon) \left(\frac{1+6\alpha}{1-2\alpha}\right) \cdot \delta_G(p, q),$$

provided that $\mu^{i-1}\lambda < |pq| \leq \mu^i\lambda/t$. Hence, by choosing the constant α such that $0 < \alpha < 1/4$ and $((1+6\alpha)/(1-2\alpha)) \leq 1+\varepsilon$, the value $\delta_H(p, q)$ is a $(1+\varepsilon)^2$ -approximation to the length of the shortest path in G between p and q . By replacing ε by $\varepsilon/3$ in the entire construction, we get a $(1+\varepsilon)$ -approximation to $\delta_G(p, q)$.

Hence, we have shown the following. For any constant λ with $1 \leq \lambda < \mu$, if p and q are two points with $|pq| > \lambda$, and if i is the positive integer such that $\mu^{i-1}\lambda < |pq| \leq \mu^i\lambda$, then we can compute a $(1+\varepsilon)$ -approximation to $\delta_G(p, q)$ in constant time, provided that (i) we know the index i , and (ii) $|pq| \leq \mu^i\lambda/t$. We will see in Section 5 how the index i can be computed. Condition (ii) says that the distance $|pq|$ must be in the “initial” part of the interval $(\mu^{i-1}\lambda, \mu^i\lambda]$. This condition can be met if we apply this construction using a constant number of values for λ . The details will be given in the following subsection.

2.3 The data structure

We choose constants μ and α such that $\mu > t$, $0 < \alpha < 1/4$ and $((1+6\alpha)/(1-2\alpha)) \leq 1+\varepsilon$. For any j , $0 \leq j \leq \lfloor \log_{\mu/t} \mu \rfloor$, let $\lambda_j := (\mu/t)^j$. For each such j , we run the greedy spanner algorithm on the t -spanner G , using the subsets

$$E_0^j := \{(p, q) \in E : |pq| \leq \lambda_j\}$$

and

$$E_i^j := \{(p, q) \in E : \mu^{i-1}\lambda_j < |pq| \leq \mu^i\lambda_j\},$$

$$1 \leq i \leq 1 + \lfloor \log_{\mu}(n^c/\lambda_j) \rfloor.$$

Our data structure consists of the collection of cluster graphs H_i^j , for $0 \leq j \leq \lfloor \log_{\mu/t} \mu \rfloor$, and $1 \leq i \leq 1 + \lfloor \log_{\mu}(n^c/\lambda_j) \rfloor$, where H_i^j is the cluster graph at the moment when the greedy algorithm has just processed all edges of the subset E_i^j . Let us now see how this data structure can be used to answer approximate shortest path queries. Let p and q be two points of V with $|pq| > 1$. Let i be the positive integer such that $\mu^{i-1} < |pq| \leq \mu^i$, and let j be the positive integer such that $\lambda_j < |pq|/\mu^{i-1} \leq (\mu/t)\lambda_j$. Then $\mu^{i-1}\lambda_j < |pq| \leq \mu^i\lambda_j/t$ and the results of Section 2.2 imply that $\Delta := \delta_{H_i^j}(p, q)$ can be computed in $O(1)$ time and satisfies $\delta_G(p, q) \leq \Delta \leq (1+\varepsilon) \cdot \delta_G(p, q)$.

As we will show in Corollary 1, the integer i can be computed in $O(1)$ time. Similarly, given i , the integer j can be computed in $O(1)$ time. Hence, our data structure answers

approximate shortest path queries, for pairs of points with inter point distance at least 1, in constant time. As mentioned in Section 2.1, we spend $O(n \log^2 n)$ time to compute the graphs $H_1^j, \dots, H_{\lceil c \log_\mu n \rceil}^j$, for each value of j , and since j is bounded by $\lceil \log_{\mu/t} \mu \rceil$ we get that the overall preprocessing time is $O(n \log^2 n)$. Finally, since each cluster graph H_i^j has n vertices and $O(n)$ edges, the complete data structure uses $O(n \log n)$ space.

We can improve this result in the following way. Gudmundsson *et al.* [20] approximate the cluster graphs using appropriate integer weight functions. They show that this leads to an implementation of the greedy algorithm that runs in $O(n \log n)$ time. If i and j are integers as above, and if we denote the cluster graph of [20] by F_i^j , then $\Delta := \delta_{F_i^j}(p, q)$ can be computed in constant time and satisfies $\delta_G(p, q) \leq \Delta \leq (1 + \epsilon)^2 \cdot \delta_G(p, q)$. Hence, using Corollary 1, the value of Δ can be computed in constant time.

We summarize our results obtained so far.

Theorem 2. *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 ϵ be a positive real constant, and let c be a positive integer constant. In $O(n \log n)$ time, we can preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q in V with $|pq| > D/n^c$, we can compute, in $O(1)$ time, a $(1 + \epsilon)$ -approximation to the shortest-path distance in G between p and q .*

In the next section, we will need a generalization of Theorem 2. If $t > 1$ and $L > 0$ are real numbers, then we say that the graph $G = (V, E)$ is an L -partial t -spanner for the point set V , if for any two points p and q of V with $|pq| < L$, we have $\delta_G(p, q) \leq t \cdot |pq|$. The following theorem states that Theorem 2 remains true for partial spanners. The proof is basically the same as that of Theorem 2.

Theorem 3. *Let V be a set of n points in \mathbb{R}^d , let $L > 0$ be a real number, and let $G = (V, E)$ be an L -partial t -spanner for V , for some real constant $t > 1$, having $O(n)$ edges. Let ϵ be a positive real constant and let c be a positive integer constant. In $O(n \log n)$ time, we can preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q of V with $L/n^c \leq |pq| < L$, we can compute, in $O(1)$ time, a $(1 + \epsilon)$ -approximation to the shortest-path distance in G between p and q .*

3 Extending the structure to support arbitrary queries

The main drawback with the query structure presented in the previous section is that one is limited to perform only “long queries”. In this section we show how to extend Theorem 2 to hold for any query, independent of its length.

The obvious idea is to construct a sequence of graphs that approximates G and fulfills the requirements of Theorem 3. We will in this section show how this can be done.

The construction consists of two main steps. First we construct a hierarchy of so-called component graphs, denoted G_1, \dots, G_ℓ . The general idea is to partition E into subsets E_1, \dots, E_ℓ such that the edge lengths in each subset differ by a factor of $n^{O(1)}$ of each other. For each index i , we consider each connected component in $(V, E_1 \cup E_2 \cup \dots \cup E_i)$ as a super vertex, the set of super vertices is denoted V_i . Then G_i is the graph with vertex set V_i and edge set E_i' where (x, y) is an edge of E_i' between two super vertices x and y if and only if there exists an edge $(p, q) \in E_{i-1} \cup E_i$ such that x is the super vertex representing

the connected component containing p and y is the super vertex representing the connected component containing q . This step will be described in more detail in Section 3.1, and then analyzed in Section 3.2.

The second step of the construction is needed to fulfill the condition in Theorem 3 that requires the graph to be a partial spanner. This condition is satisfied by adding a set of short edges to G_i , and we obtain a graph G_i'' .

As we will see, the sequence of graphs G_1, \dots, G_ℓ defines a natural hierarchical representation of G . This representation will allow us to find, for any two query points p and q of V , an index i and two vertices x and y in G_i'' such that:

- i) p and x , and also q and y , are close to each other compared to the distance between p and q , and
- ii) $|xy|$ is within a factor of $n^{O(1)}$ more than the edge lengths in E_i .

These two properties imply that $\delta_{G_i''}(x, y)$ closely approximates $\delta_G(p, q)$, as will be shown in Section 3.4.

3.1 A hierarchy of component graphs

Let V be a set of n points in \mathbb{R}^d , where n is a sufficiently large integer, let $t > 1$ be a real constant, and let $G = (V, E)$ be a t -spanner for V . We fix an integer constant c , which is assumed to be sufficiently large. (In fact, $c = 7$ will be sufficient in our application.) We also fix a real constant $\epsilon > 0$.

The following algorithm partitions the edge set E into subsets (some of which are empty).

- Step 1:** Initialize $E' := E$.
- Step 2:** Compute the length L of a shortest edge in E' , and set $L_1 := n^c L$, $L_2 := n^c L_1$, and $L_3 := n^c L_2$.
- Step 2.1:** Compute the set E_1 consisting of all edges in E' whose lengths are less than L_1 , and compute the set $E' := E' \setminus E_1$.
- Step 2.2:** Compute the set E_2 consisting of all edges in E' whose lengths are less than L_2 , and compute the set $E' := E' \setminus E_2$.
- Step 2.3:** Compute the set E_3 consisting of all edges in E' whose lengths are less than L_3 , and compute the set $E' := E' \setminus E_3$.
- Step 2.4:** Initialize $i := 3$.
- Step 3:** If $E_i = \emptyset$ and $E' = \emptyset$, then go to Step 4. Otherwise, do the following.
 - Step 3.1:** If $E_i \neq \emptyset$, then set $L_{i+1} := n^c L_i$, else let L be the length of a shortest edge in E' and set $L_{i+1} := n^c L$.
 - Step 3.2:** Compute the set E_{i+1} consisting of all edges in E' whose lengths are less than L_{i+1} , compute the set $E' := E' \setminus E_{i+1}$, set $i := i + 1$, and go to Step 3.
- Step 4:** Set $\ell := i$.

Consider the sequence L_1, L_2, \dots, L_ℓ of real numbers and the sequence E_1, E_2, \dots, E_ℓ of edge sets that are computed by this algorithm. For each i with $1 \leq i \leq \ell$, we define the

interval I_i by

$$I_i := [L_i/n^c, L_i).$$

The following lemma follows immediately from the algorithm.

Lemma 1. *The following properties hold.*

1. For each i with $1 \leq i \leq \ell - 1$, for each $L \in I_i$, and for each $L' \in I_{i+1}$, we have $L < L'$.
2. For each i with $1 \leq i \leq \ell$ and for each edge (p, q) in E_i , we have $|pq| \in I_i$, i.e., $L_i/n^c \leq |pq| < L_i$.
3. For each edge $(p, q) \in E$, there is a unique index i with $1 \leq i \leq \ell$ such that $|pq| \in I_i$.
4. For each i with $1 \leq i \leq \ell - 1$, we have $L_{i+1} \geq n^c L_i$.
5. $L_2 = n^c L_1$ and $L_3 = n^c L_2$.
6. For each i with $1 \leq i \leq \ell - 1$ and for which $E_i \neq \emptyset$, we have $L_{i+1} = n^c L_i$.
7. $E_\ell = \emptyset$.
8. $3 \leq \ell \leq 2|E| + 1 = O(n)$.

We now define a sequence $G_i = (V_i, F_i)$, $1 \leq i \leq \ell$, of Euclidean graphs, and a sequence U_i , $1 \leq i \leq \ell$, of forests. Let $V_1 := V$, $F_1 := E_1$, $V_2 := V$, and $F_2 := E_1 \cup E_2$, so that $G_1 = (V_1, F_1)$ and $G_2 = (V_2, F_2)$.

For each connected component C of G_1 , let T_C be the tree whose root stores the index 1 and an arbitrary point of C , and that has $|C|$ children (which are leaves), each leaf storing a unique point of C . The forest U_1 is the collection of trees T_C , where C ranges over all connected components of G_1 . We define the forest U_2 in the same way with respect to the graph G_2 . The root of each tree in this forest stores the index 2.

Let i be such that $3 \leq i \leq \ell$ and assume that the graphs G_1, G_2, \dots, G_{i-1} and the forests U_1, U_2, \dots, U_{i-1} have been defined already. The following algorithm defines the graph $G_i = (V_i, F_i)$ and the forest U_i .

Step 1: Initially, $F_i = \emptyset$ and $V_i = \emptyset$.

Step 2: For each edge (p, q) in $E_{i-1} \cup E_i$, do the following. Let x be the point stored at the root of the tree in U_{i-2} in which p is stored as a leaf. Let y be the point stored at the root of the tree in U_{i-2} in which q is stored as a leaf. If $x \neq y$, then insert the edge (x, y) into F_i and insert x and y into V_i , as shown in Fig. 1a-b.

After this step, we have obtained the graph $G_i = (V_i, F_i)$.

Step 3: For each connected component C of G_i , choose an arbitrary point z in C . Consider all trees T in U_{i-2} such that the points stored in their roots form the set C . Construct a tree whose root stores the index i and the point z and that has the roots of all these trees T as its children, as illustrated in Fig. 1b-c.

Step 4: The forest U_i consists of all trees that are obtained from Step 3 and all trees in U_{i-2} whose roots store a point that is not in V_i .

Lemma 2. *For each i with $1 \leq i \leq \ell$, the trees in the forest U_i are in one-to-one correspondence with the connected components of the graph with vertex set V and edge set $E_1 \cup E_2 \cup \dots \cup E_i$. Both U_ℓ and $U_{\ell-1}$ consist of one single tree.*

Proof. The first claim follows immediately from the way the forest U_i is defined. The second claim follows from the facts that E_ℓ is empty and the spanner G must be connected. \square

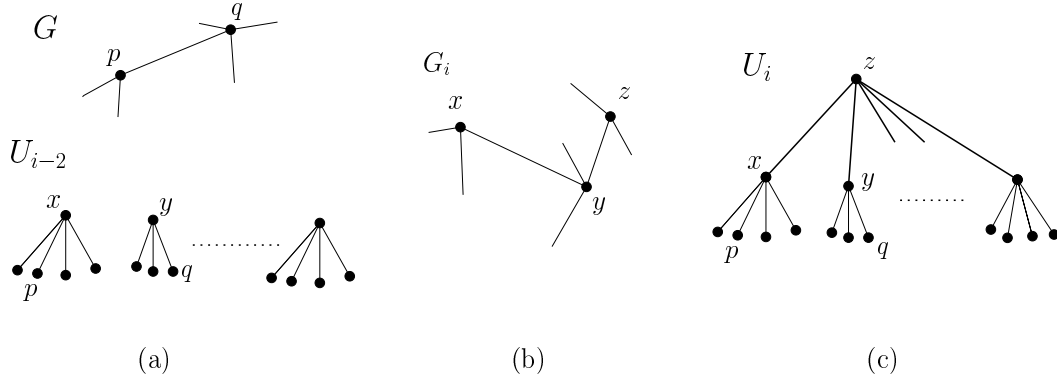


Fig. 1. Illustrating the construction of G_i and U_i .

3.2 Properties of the component graphs

The aim of this section is to prove that if x and y are two vertices of G_i such that $|xy|$ is within a factor of $n^{O(1)}$ of the longest edge in E_i then $\delta_{G_i}(x, y)$ closely approximates $\delta_G(x, y)$.

We start by showing two crucial properties.

Lemma 3. *The following properties hold.*

1. For any i with $1 \leq i \leq \ell$, let p and x be two points that are stored in the same tree of the forest U_i . Then $|px| < nL_i$.
2. For each i with $1 \leq i \leq \ell$, the length of each edge in G_i is less than $2L_i$.

Proof. If p and x are in the same tree of the forest U_i , then they are in the same connected component of an imaginary graph $\widehat{G}_i = (S, \cup_{j=1}^i E_j)$. In other words, they lie in the same connected component, if you include all edges of length at most L_i . Thus there is a path in the original graph G connecting them by edges of length at most L_i . Since such a path cannot have more than $n - 1$ edges, the first claim follows.

The proof of the second claim is by induction on i . This claim clearly holds if $i = 1$ and $i = 2$. Let $3 \leq i \leq \ell$, and assume that the second claim holds for the graphs G_1, G_2, \dots, G_{i-1} .

Let (a, b) be an arbitrary edge of G_i . Then $a \neq b$ and there is an edge (p, q) in $E_{i-1} \cup E_i$ where U_{i-2} contains two distinct trees T_a and T_b such that (i) the root of T_a stores a , (ii) p is stored in T_a , (iii) the root of T_b stores b , and (iv) q is stored in T_b . By the first claim, we have $|pa| < nL_{i-2}$ and $|qb| < nL_{i-2}$. Hence,

$$|ab| \leq |ap| + |pq| + |qb| < |pq| + 2nL_{i-2}.$$

Since $(p, q) \in E_{i-1} \cup E_i$, we have $|pq| < L_i$. Moreover, we have $L_{i-2} \leq L_i/n^{2c}$. Therefore, we obtain the following upper bound on the length of the edge (a, b) :

$$|ab| < |pq| + 2nL_{i-2} < L_i + 2L_i/n^{2c-1} < 2L_i,$$

completing the proof of the lemma. □

Now we are ready to prove the main results of this section, Lemma 4 and Lemma 5.

Lemma 4. *Let i be an index such that $1 \leq i \leq \ell$, and let x and y be two vertices of G_i such that $L_i/n^{c+4} \leq |xy| < L_i/t$. Then*

$$\delta_{G_i}(x, y) \leq (1 + \epsilon) \cdot \delta_G(x, y).$$

Proof. If $i \leq 2$, then we have $\delta_{G_i}(x, y) = \delta_G(x, y)$. So assume that $i \geq 3$. Let $P = (x = x_0, x_1, \dots, x_k = y)$ be a shortest path in G between x and y . Since G is a t -spanner, the length of P is less than or equal to $t|xy|$, which is less than L_i , as stated in the lemma. Hence, $|x_j x_{j+1}| < L_i$ for each j with $0 \leq j \leq k-1$.

We will convert P to a path Q between x and y in the graph G_i . It will then be shown that the length of Q is less than or equal to $1 + \epsilon$ times the length of P . During the conversion, we will maintain the following invariant.

Invariant: The subpath $(x = x_0, x_1, \dots, x_j)$ has been converted into a path $(x = y_0, y_1, \dots, y_{k'})$ in G_i , as illustrated in Fig. 2. The point $y_{k'}$ is stored at the root of the tree in the forest U_{i-2} that stores x_j .

We start the conversion by setting $j := 0$, $k' := 0$, and $y_0 := x_0$. Since x_0 is a vertex of G_i , the invariant holds at this moment.

Assume that $j < k$. If x_j and x_{j+1} are stored in the same tree of the forest U_{i-2} , then we set $j := j + 1$. Observe that the invariant is maintained in this case.

Now assume that x_j and x_{j+1} are stored in different trees of U_{i-2} . Since $|x_j x_{j+1}| < L_i$, the edge (x_j, x_{j+1}) is contained in $E_1 \cup \dots \cup E_i$. Since x_j and x_{j+1} are stored in different trees of U_{i-2} , this edge cannot be contained in $E_1 \cup \dots \cup E_{i-2}$. Hence, (x_j, x_{j+1}) is contained in $E_{i-1} \cup E_i$. Let $y_{k'+1}$ be the point stored at the root of the tree in U_{i-2} that contains x_{j+1} . Then $(y_{k'}, y_{k'+1})$ is an edge of G_i . Hence, if we set $j := j + 1$ and $k' := k' + 1$, then the invariant still holds.

We continue extending the path in G_i until $j = k$. Observe that the last vertex of Q is equal to $x_k = y$. Therefore, we have obtained the path Q between x and y in the graph G_i . It remains to estimate the length of Q . Consider the edge $(y_{k'}, y_{k'+1})$. By the triangle inequality, we have

$$|y_{k'} y_{k'+1}| \leq |y_{k'} x_j| + |x_j x_{j+1}| + |x_{j+1} y_{k'+1}|.$$

By Lemma 3, both $|y_{k'} x_j|$ and $|x_{j+1} y_{k'+1}|$ are less than

$$nL_{i-2} \leq L_i/n^{2c-1} \leq |xy|/n^{c-5},$$

which implies that

$$|y_{k'} y_{k'+1}| \leq |x_j x_{j+1}| + 2|xy|/n^{c-5}.$$

Since the path Q contains less than n edges, it follows that the length of Q is less than the length of P plus $2|xy|/n^{c-6}$. Therefore,

$$\delta_{G_i}(x, y) < \delta_G(x, y) + \frac{2}{n^{c-6}}|xy| \leq \delta_G(x, y) + \epsilon|xy| \leq (1 + \epsilon) \cdot \delta_G(x, y).$$

This completes the proof. \square

It remains to prove an upper bound on $\delta_G(x, y)$.

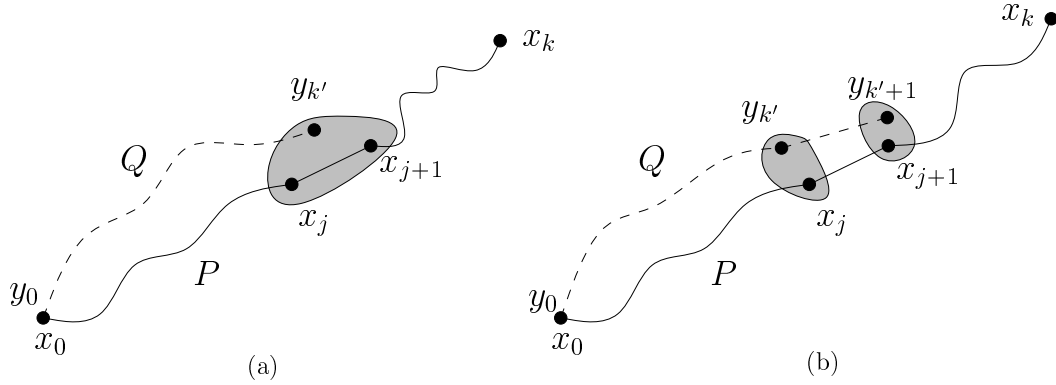


Fig. 2. Illustration for the proof of Lemma 4.

Lemma 5. *Let i be an index such that $1 \leq i \leq \ell$, and let x and y be two vertices of G_i such that $L_i/n^{c+4} \leq |xy| < L_i/t$. Then*

$$\delta_G(x, y) \leq (1 + \epsilon) \cdot \delta_{G_i}(x, y).$$

Proof. We have $\delta_G(x, y) = \delta_{G_i}(x, y)$ for $i \in \{1, 2\}$. So we may assume that $i \geq 3$. Let (a, b) be an arbitrary edge of G_i . It follows from the definition of G_i that there is an edge (p, q) in G such that (i) $(p, q) \in E_{i-1} \cup E_i$, (ii) a is stored at the root of the tree in the forest U_{i-2} that contains p , and (iii) b is stored at the root of the tree in the forest U_{i-2} that contains q . By Lemma 3, both $|ap|$ and $|bq|$ are less than $nL_{i-2} \leq L_i/n^{2c-1}$. Therefore,

$$|pq| \leq |pa| + |ab| + |bq| \leq |ab| + 2L_i/n^{2c-1}.$$

Since G is a t -spanner, we have

$$\delta_G(a, p) \leq t|ap| \leq tL_i/n^{2c-1} \quad \text{and} \quad \delta_G(q, b) \leq t|bq| \leq tL_i/n^{2c-1}.$$

By combining these inequalities, it follows that

$$\delta_G(a, b) \leq \delta_G(a, p) + |pq| + \delta_G(q, b) \leq |ab| + 2(t+1)L_i/n^{2c-1}. \quad (1)$$

Now consider the points x and y . It follows from Lemma 4 that x and y are connected by a path in G_i . Let k be the number of edges on a shortest path in G_i between x and y . If we apply the inequality (1) to each edge on this path, then we obtain

$$\delta_G(x, y) \leq \delta_{G_i}(x, y) + k \cdot 2(t+1)L_i/n^{2c-1} \leq \delta_{G_i}(x, y) + 2(t+1)L_i/n^{2c-2}.$$

Since $L_i \leq n^{c+4}|xy|$ and $|xy| \leq \delta_{G_i}(x, y)$, it follows that

$$\delta_G(x, y) \leq \delta_{G_i}(x, y) + 2(t+1)|xy|/n^{c-6} \leq (1 + \epsilon)\delta_{G_i}(x, y).$$

This completes the proof. \square

3.3 Extending the component graphs to partial spanners

Let i be an index such that $1 \leq i \leq \ell$, and let x and y be two vertices of G_i such that $L_i/n^{c+4} \leq |xy| < L_i/t$. Then it follows from Lemma 4 and the fact that G is a t -spanner that

$$\delta_{G_i}(x, y) \leq (1 + \epsilon) \cdot \delta_G(x, y) \leq (1 + \epsilon)t|xy|. \quad (2)$$

We would like to apply Theorem 3 to the graph G_i . This is, however, only possible if *each* pair of vertices of G_i having Euclidean distance less than L_i/t is connected by a $((1 + \epsilon)t)$ -spanner path. Observe that this is true if $i = 1$ and $i = 2$. In order to achieve this property for all i , we proceed as follows.

For each i with $3 \leq i \leq \ell$, let G'_i be a $((1 + \epsilon)t)$ -spanner for the vertex set V_i of G_i , having $O(|V_i|)$ edges. Such a spanner can be constructed in time $O(|V_i| \log |V_i|)$ using, for example, the algorithms by Salowe [30] or by Callahan and Kosaraju [10]. Let G''_i be the graph with vertex set V_i whose edge set is the union of the edge set of G_i and the set of all edges of G'_i having length at most L_i/n^{c+3} .

Lemma 6. *Let i be an index with $1 \leq i \leq \ell$, and let x and y be two vertices of G''_i such that $|xy| < L_i/t$. Then*

1. $\delta_{G''_i}(x, y) \leq (1 + \epsilon)t|xy|$, and
2. if $L_i/n^{c+1} \leq |xy|$, then $\frac{\delta_G(x, y)}{1 + \epsilon} \leq \delta_{G''_i}(x, y) \leq (1 + \epsilon) \cdot \delta_G(x, y)$.

Proof. The claims are true if $i \in \{1, 2\}$. Assume that $i \geq 3$. If $L_i/n^{c+4} \leq |xy|$, then the first claim follows from (2) and the fact that G_i is a subgraph of G''_i . So we may assume that $|xy| < L_i/n^{c+4}$. Since G'_i is a $((1 + \epsilon)t)$ -spanner for V_i , we have

$$\delta_{G'_i}(x, y) \leq (1 + \epsilon)t|xy| < (1 + \epsilon)tL_i/n^{c+4} \leq L_i/n^{c+3}.$$

Hence, the shortest path in G'_i between x and y is completely contained in G''_i . Therefore, we have

$$\delta_{G''_i}(x, y) \leq \delta_{G'_i}(x, y) < (1 + \epsilon)t|xy|,$$

proving the first claim.

To prove the second claim, assume that $L_i/n^{c+1} \leq |xy|$ as stated. Since G_i is a subgraph of G''_i , we have $\delta_{G''_i}(x, y) \leq \delta_{G_i}(x, y)$. By Lemma 4, we have $\delta_{G_i}(x, y) \leq (1 + \epsilon) \cdot \delta_G(x, y)$, so it remains to prove that $\delta_G(x, y) \leq (1 + \epsilon) \cdot \delta_{G''_i}(x, y)$.

Let $x = x_0, x_1, \dots, x_k = y$ be a shortest path in G''_i between x and y . (The rightmost inequality in the second claim implies that this path exists.) Let j be any index with $0 \leq j \leq k - 1$ and consider the edge (x_j, x_{j+1}) in G''_i . Either (x_j, x_{j+1}) is an edge of G_i or an edge of G'_i . First assume that (x_j, x_{j+1}) is an edge of G_i . Then it follows from the proof of Lemma 5 (see (1)) that

$$\delta_G(x_j, x_{j+1}) \leq |x_j x_{j+1}| + 2(t + 1)L_i/n^{2c-1}.$$

If (x_j, x_{j+1}) is not an edge of G_i , then it must be an edge of G'_i and $|x_j x_{j+1}| \leq L_i/n^{c+3}$. In this case, we have

$$\delta_G(x_j, x_{j+1}) \leq t|x_j x_{j+1}| \leq tL_i/n^{c+3}.$$

Hence, we always have

$$\delta_G(x_j, x_{j+1}) \leq |x_j x_{j+1}| + 2(t+1)L_i/n^{2c-1} + tL_i/n^{c+3}.$$

It follows that

$$\delta_G(x, y) \leq \sum_{j=0}^k \delta_G(x_j, x_{j+1}) \leq \delta_{G_i''}(x, y) + k(2(t+1)L_i/n^{2c-1} + tL_i/n^{c+3}).$$

Since $L_i \leq n^{c+1}|xy|$ and $k \leq n$, we obtain

$$\delta_G(x, y) \leq \delta_{G_i''}(x, y) + 2(t+1)|xy|/n^{c-3} + t|xy|/n \leq \delta_{G_i''}(x, y) + \epsilon|xy|.$$

Since $|xy| \leq \delta_{G_i''}(x, y)$, the proof is complete. \square

3.4 The graphs G_i'' approximate G

In this section we will prove that the graphs G_i'' approximate the input graph G . It will be shown that for any two points p and q in V one can, with the help of the trees $U_{\ell-1}$ and U_ℓ , find a graph G_i'' and two vertices x and y of G_i'' such that $\delta_{G_i''}(x, y)$ closely approximates $\delta_G(p, q)$. The query structure will be given in Section 3.5.

Recall from Lemma 1 that each of $U_{\ell-1}$ and U_ℓ is one single tree. Therefore, for any two points p and q in V , the lowest common ancestor of the leaves in $U_{\ell-1}$ or U_ℓ storing p and q is well-defined.

Lemma 7. *Let p and q be two distinct points of V , let U be the tree $U_{\ell-1}$ or U_ℓ , and let u be the lowest common ancestor of the leaves in U storing p and q . Let v and w be the two children of u that contain p and q in their subtrees, respectively, and let x and y be the points of V that are stored in v and w , respectively, as illustrated in Fig. 3a. Finally, let i be the index that is stored with u . If $i \geq 3$, then following inequalities hold.*

1. $|xy| < 2nL_{i-2} + |pq|$.
2. $|pq| < 2nL_{i-2} + |xy|$.
3. $|pq| \geq L_{i-2}/t$.
4. $|xy| \geq L_{i-2}/t$.

Proof. Let j be the index stored with v . Then v is the root of a tree in the forest U_j and $j \leq i-2$. Hence, by Lemma 3, $|xp| < nL_j \leq nL_{i-2}$. In a similar way, we obtain $|qy| < nL_{i-2}$. It follows that

$$|xy| \leq |xp| + |pq| + |qy| < 2nL_{i-2} + |pq|,$$

proving the first claim. The second claim can be proved in the same way.

To prove the third claim, assume the opposite, i.e., that $|pq| < L_{i-2}/t$. Since G is a t -spanner, we have $\delta_G(p, q) \leq t|pq| < L_{i-2}$. Hence, p and q are connected by a path in the graph with vertex set V and edge set $E_1 \cup E_2 \cup \dots \cup E_{i-2}$. But then, by Lemma 2, p and q are stored in the same tree in the forest U_{i-2} , which is a contradiction. The fourth claim can be proved in the same way. \square

By using the above inequalities we can now prove the following lemma.

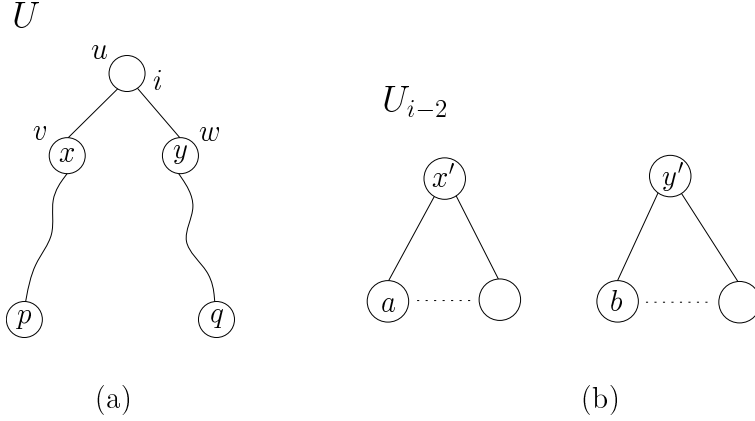


Fig. 3. Illustration of Lemma 7 and Lemma 8.

Lemma 8. *Let p and q be two distinct points of V , let U be the tree $U_{\ell-1}$ or U_ℓ , and let u be the lowest common ancestor of the leaves in U storing p and q . Let v and w be the two children of u that contain p and q in their subtrees, respectively, and let x and y be the points of V that are stored in v and w , respectively, as illustrated in Fig. 3a. Finally, let i be the index that is stored with u . If $L_i/n^{c+1} \leq |xy| < L_i/t$, then*

1. $\delta_G(p, q) \leq (1 + 2\epsilon) \cdot \delta_{G_i''}(x, y)$, and
2. $\delta_{G_i''}(x, y) \leq (1 + \epsilon)^2 \cdot \delta_G(p, q)$.

Proof. We assume that $i \geq 3$ (and leave the case when $i \in \{1, 2\}$ to the reader). First observe that x and y are vertices of the graph G_i'' . By Lemma 6, we have $\delta_G(x, y) \leq (1 + \epsilon) \cdot \delta_{G_i''}(x, y)$. By Lemma 3, we have $|px| < nL_{i-2} \leq L_i/n^{2c-1}$ and $|yq| < nL_{i-2} \leq L_i/n^{2c-1}$. By using these inequalities and the fact that G is a t -spanner, we obtain

$$\begin{aligned}
\delta_G(p, q) &\leq \delta_G(p, x) + \delta_G(x, y) + \delta_G(y, q) \\
&\leq t|px| + (1 + \epsilon) \cdot \delta_{G_i''}(x, y) + t|yq| \\
&< 2tL_i/n^{2c-1} + (1 + \epsilon) \cdot \delta_{G_i''}(x, y) \\
&\leq 2t|xy|/n^{c-2} + (1 + \epsilon) \cdot \delta_{G_i''}(x, y) \\
&\leq (2t/n^{c-2} + (1 + \epsilon)) \cdot \delta_{G_i''}(x, y) \\
&\leq (1 + 2\epsilon) \cdot \delta_{G_i''}(x, y).
\end{aligned}$$

The proof of the second claim is similar:

$$\begin{aligned}
\delta_{G_i''}(x, y) &\leq (1 + \epsilon) \cdot \delta_G(x, y) \\
&\leq (1 + \epsilon) (\delta_G(x, p) + \delta_G(p, q) + \delta_G(q, y)) \\
&\leq (1 + \epsilon) (t|xp| + \delta_G(p, q) + t|yq|) \\
&< (1 + \epsilon) (2t|xy|/n^{c-2} + \delta_G(p, q)).
\end{aligned}$$

By Lemma 7, we have $|xy| < 2nL_{i-2} + |pq|$ and $|pq| \geq L_{i-2}/t$. This implies that $|xy| \leq (2tn + 1)|pq|$. Hence,

$$\delta_{G_i''}(x, y) \leq (1 + \epsilon) (2t(2tn + 1)|pq|/n^{c-2} + \delta_G(p, q)).$$

Since $|pq| \leq \delta_G(p, q)$, it follows that $\delta_{G''}(x, y) \leq (1 + \epsilon)^2 \cdot \delta_G(p, q)$. \square

Lemma 9. *Let p and q be two distinct points of V .*

1. *Let u be the lowest common ancestor of the leaves in $U_{\ell-1}$ storing p and q , as illustrated in Fig. 4. Let v and w be the two children of u that contain p and q in their subtrees, respectively, and let x and y be the points of V that are stored in v and w , respectively. Finally, let i be the index that is stored with u .*
2. *Let u' be the lowest common ancestor of the leaves in U_ℓ storing p and q . Let v' and w' be the two children of u' that contain p and q in their subtrees, respectively, and let x' and y' be the points of V that are stored in v' and w' , respectively. Finally, let i' be the index that is stored with u' .*

Then $i' = i + 1$ or $i' = i - 1$.

Proof. Observe that $U_{\ell-1}$ stores only even indices and U_ℓ stores only odd indices, or vice versa. We assume that $i \geq 3$ (and leave the case when $i \in \{1, 2\}$ to the reader). We may assume without loss of generality that $i' \geq i + 1$. Hence, we have to prove that $i' = i + 1$. Assume the opposite, that $i' \geq i + 3$. Then $L_i \leq L_{i'-3} \leq L_{i'-2}/n^c$. By Lemma 7, we have $L_{i'-2} \leq t|pq|$, $L_{i-2} \leq t|xy|$ and $|pq| \leq 2nL_{i-2} + |xy|$. It follows that

$$\begin{aligned} L_i &\leq t|pq|/n^c \\ &\leq t(2nL_{i-2} + |xy|)/n^c \\ &\leq t(2tn|xy| + |xy|)/n^c \\ &= t(2tn + 1)|xy|/n^c. \end{aligned}$$

According to Lemma 3, we have $|xy| < nL_i$. Therefore,

$$L_i \leq t(2tn + 1)|xy|/n^c \leq t(2tn + 1)L_i/n^{c-1} < L_i,$$

which is a contradiction. Hence, we have proved that $i' = i + 1$. \square

Lemma 10. *Let p and q be two distinct points of V .*

1. *Let u be the lowest common ancestor of the leaves in $U_{\ell-1}$ storing p and q , as illustrated in Fig. 4. Let v and w be the two children of u that contain p and q in their subtrees, respectively, and let x and y be the points of V that are stored in v and w , respectively. Finally, let i be the index that is stored with u .*
2. *Let u' be the lowest common ancestor of the leaves in U_ℓ storing p and q . Let v' and w' be the two children of u' that contain p and q in their subtrees, respectively, and let x' and y' be the points of V that are stored in v' and w' , respectively. Finally, let i' be the index that is stored with u' .*

Then

$$L_i/n^{c+1} \leq |xy| < L_i/t \quad \text{or} \quad L_{i'}/n^{c+1} \leq |x'y'| < L_{i'}/t$$

Proof. We assume that $i \geq 3$ and $i' \geq 3$ (and leave the other cases to the reader). By Lemma 9, we may assume without loss of generality that $i' = i + 1$. Since, by Lemma 7, $|xy| \leq 2nL_{i-2} + |pq|$ and $|pq| \leq 2nL_{i-1} + |x'y'|$, we have

$$|xy| \leq 2nL_{i-2} + 2nL_{i-1} + |x'y'| \leq 3nL_{i-1} + |x'y'|.$$

Hence,

$$|xy| \leq 3L_i/n^{c-1} + |x'y'|. \quad (3)$$

In a similar way, we obtain the inequality

$$|x'y'| \leq 3L_i/n^{c-1} + |xy|. \quad (4)$$

By Lemma 3, we have $|xy| < nL_i$. Therefore,

$$\begin{aligned} |x'y'| &\leq (3/n^{c-1} + n) L_i \\ &\leq (3/n^{2c-1} + 1/n^{c-1}) L_{i+1} \\ &< L_{i+1}/t \\ &= L_{i'}/t. \end{aligned}$$

If $|x'y'| \geq L_{i+1}/n^{c+2}$, then the lemma holds. So from now on, we assume that

$$|x'y'| < L_{i+1}/n^{c+1}. \quad (5)$$

Let L be the length of a longest edge on a shortest path between x' and y' in the graph G . Since $L \leq \delta_G(x', y') \leq t|x'y'|$, it follows that $L/t \leq |x'y'|$. Let j be the index such that L is contained in the interval I_j . Then $L_j/n^c \leq L$ and, therefore,

$$L_j/n^{c+1} \leq L_j/(tn^c) \leq |x'y'|. \quad (6)$$

By combining (5) and (6), it follows that

$$L_j \leq n^{c+1}|x'y'| < L_{i+1},$$

which implies that $j \leq i$. We claim that $j = i$. To prove this, assume that $j \leq i-1$. Then x' and y' are connected by a path in the graph with vertex set V and edge set $E_1 \cup E_2 \cup \dots \cup E_{i-1}$. Hence, by Lemma 2, x' and y' are stored in the same tree in the forest U_{i-1} , which is a contradiction.

Observe that, since $j = i$, the points x' and y' are stored in the same tree in the forest U_i .

Since $L \in I_j = I_i$, the edge set $E_j = E_i$ is non-empty. Therefore, by Lemma 1, we have $L_{i+1} = n^c L_i$. Then it follows from (3) and (5) that

$$|xy| < 3L_i/n^{c-1} + L_{i+1}/n^{c+1} = 3L_i/n^{c-1} + L_i/n < L_i/t.$$

It remains to prove that $L_i/n^{c+1} \leq |xy|$. We will prove this inequality by contradiction. So we assume that

$$|xy| < L_i/n^{c+1}.$$

Let L' be the length of a longest edge on a shortest path between x and y in the graph G , and let j' be the index such that $L' \in I_{j'}$. An argument similar to the one used to obtain (6) shows that $L_{j'}/n^{c+1} \leq |xy|$ and, hence, $L_{j'} \leq n^{c+1}|xy| < L_i$, which shows that $j' \leq i-1$. If $j' \leq i-2$, then x and y must be contained in the same tree in the forest U_{i-2} , which we know is not the case. Therefore, $j' = i-1$ and, hence, the points x and y are stored in the same tree in the forest U_{i-1} . It follows that

1. x' and y' are stored in the same tree in the forest U_i , and
2. x and y are stored in the same tree in the forest U_{i-1} .

By the assumptions in the lemma, p and x are stored in the same tree in U_{i-2} , and q and y are stored in the same tree in U_{i-2} . Let T be the tree in U_{i-1} that stores x and y . By Lemma 2, the subset of V stored in T is the union of one or more subsets of V that are stored in trees in U_{i-2} . Therefore, p and q are both stored in T . Which implies that the lowest common ancestor of the leaves in U_ℓ storing p and q stores an index that is less than or equal to $i - 1$. This is a contradiction. \square

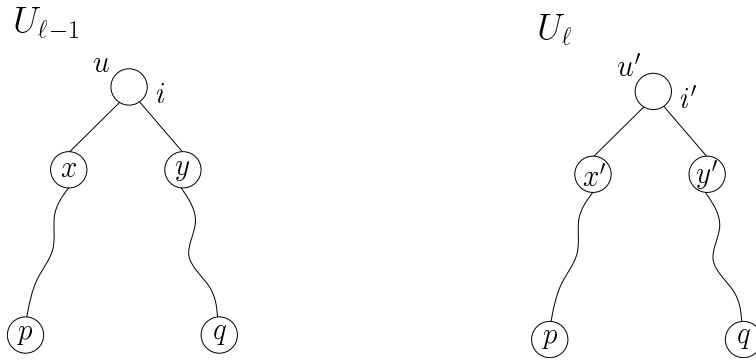


Fig. 4. Illustration of Lemmas 9-10.

Let us summarize this section. Let p and q be two points in V , and consider the two trees $U_{\ell-1}$ and U_ℓ , as shown in Fig. 4. From Lemma 10 it follows that one of i and i' satisfies the conditions of Lemma 8 and hence, at least one of $\delta_{G''_i}(x, y)$ and $\delta_{G''_{i'}}(x, y)$ is a close approximation of $\delta_G(p, q)$. Note also that G''_i and $G''_{i'}$ satisfy the requirements for Theorem 3, this follows from Lemma 6.

3.5 Answering an approximate shortest path query

We are given a t -spanner $G = (V, E)$ and an $\epsilon > 0$. The data structure that we will use consists of:

- The sequence L_1, \dots, L_ℓ , as described in Section 3.1.
- The two trees $U_{\ell-1}$ and U_ℓ , as described in Section 3.1. The trees have been preprocessed in linear time such that lowest common ancestor queries can be answered in constant time. This was first shown by Harel and Tarjan [24], see also Schieber and Vishkin [31], Gusfield [23] and, Bender and Farach-Colton [6].
- Finally, we also have the data structure of Theorem 3 for each graph G''_i , $1 \leq i \leq \ell$. In Lemma 6 we proved that G''_i is an (L_i/t) -partial $(1 + \epsilon)t$ -spanner for V_i , hence we can apply Theorem 3 to G''_i .

The query algorithm is as follows. Let p and q be two distinct points of V .

Step 1: Compute the lowest common ancestor u of the leaves in $U_{\ell-1}$ storing p and q . Let v and w be the children of u that contain p and q in their subtrees, respectively, and let x and

y be the points of V that are stored in v and w , respectively. Finally, let i be the index that is stored with u , see Fig. 4.

Step 2: Compute the lowest common ancestor u' of the leaves in U_ℓ storing p and q . Let v' and w' be the children of u' that contain p and q in their subtrees, respectively, and let x' and y' be the points of V that are stored in v' and w' , respectively. Finally, let i' be the index that is stored with u' .

Step 3: If $L_i/n^{c+1} \leq |xy| < L_i/t$, then use the algorithm of Theorem 3 to compute a $(1 + \epsilon)$ -approximation Δ to $\delta_{G_i''}(x, y)$. Otherwise, use the algorithm of Theorem 3 to compute a $(1 + \epsilon)$ -approximation Δ to $\delta_{G_{i'}''}(x, y)$.

It follows from Lemmas 8 and 10 that

$$\delta_G(p, q)/(1 + 2\epsilon) \leq \Delta \leq (1 + \epsilon)^3 \cdot \delta_G(p, q).$$

Hence returning $\Delta' = (1 + 2\epsilon)\Delta$ and replacing ϵ with $\epsilon/6$ we get

$$\delta_G(p, q) \leq \Delta' \leq (1 + \epsilon) \cdot \delta_G(p, q).$$

The three steps can be computed in constant time using the above data structures.

3.6 Complexity of the data structures

We consider the preprocessing. The sequences E_i , $1 \leq i \leq \ell$, and I_i , $1 \leq i \leq \ell$, can be computed in $O(|E| \log n) = O(n \log n)$ time.

By using a separate union-find data structure, we can compute the sequences $G_i = (V_i, F_i)$, $1 \leq i \leq \ell$, and U_i , $1 \leq i \leq \ell$, in time that is proportional to

$$|E| \log n + \sum_{i=1}^{\ell} (|V_i| + |F_i|).$$

Since G_i does not contain vertices of degree zero, we have $|V_i| \leq 2|F_i|$. Also, it is clear from the algorithm that constructs G_i that $|F_i| \leq |E_{i-1}| + |E_i|$. Therefore, the time to compute all graphs G_i and all forests U_i is proportional to

$$|E| \log n + \sum_{i=1}^{\ell} |E_i| = O(|E| \log n) = O(n \log n).$$

Using the algorithm of Callahan and Kosaraju [10], the sequence of spanners G'_i , $1 \leq i \leq \ell$, can be computed in time that is proportional to

$$\sum_{i=1}^{\ell} |V_i| \log n = O(|E| \log n) = O(n \log n).$$

The time needed to compute the sequence of graphs G''_i , $1 \leq i \leq \ell$, is proportional to

$$\sum_{i=1}^{\ell} (|V_i| + |F_i|) = O(|E|) = O(n).$$

Finally, by Theorem 3, the total time to preprocess all graphs G_i'' is proportional to

$$\sum_{i=1}^{\ell} |V_i| \log |V_i| = O(|E| \log n) = O(n \log n).$$

Putting together the results we obtain Theorem 1 that we restate below.

Theorem 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. In $O(n \log n)$ time we can preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q in V , we can in constant time compute a $(1 + \varepsilon)$ -approximation of the shortest-path distance in G between p and q .

In the traditional algebraic model of computation (without indirect addressing) a similar result can be obtained.

Theorem 4. 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. In $O(n \log^2 n)$ time we can preprocess G into a data structure of size $O(n \log n)$, such that for any two points p and q in V , we can in $O(\log \log n)$ time compute a $(1 + \varepsilon)$ -approximation of the shortest-path distance in G between p and q .

In the above two theorems, we assume that the number of edges m of G is $O(n)$. If this is not the case, then we apply the algorithm in [22] to compute a graph G' which is a $(1 + \varepsilon)$ -spanner of G and that has $O(n)$ edges. Then we use G' to approximate shortest path distances. In the algebraic model of computation, G' can be computed in $O(m \log \log n + n \log n)$ time. If we add indirect-addressing as a unit-time operation, then G' can be computed in $O(m + n \log n)$ time.

4 Applications

The data structure of Theorem 1 has several nice properties, and we believe that it can be applied to a number of basic problems. In this section we consider some examples where the utilization of the data structure immediately improves the time complexity, and in some cases also the approximation factor, of existing results.

4.1 Shortest paths in planar polygonal domains with obstacles

Consider a polygonal domain consisting of a collection \mathcal{F} of polygonal obstacles in the plane, and let V be the set of vertices of these obstacles. The *visibility graph* of \mathcal{F} is the graph G with vertex set V , and in which any two vertices p and q are connected by an edge if and only if the line segment joining p and q does not intersect the interior of any obstacle. We denote by $\delta_G(p, q)$, the Euclidean length of a shortest path between p and q in the graph G . For any real number $t > 1$, we say that the collection \mathcal{F} is *t -rounded* if $\delta_G(p, q)$ is at most t times the Euclidean distance between p and q , for any two points p and q in V . In other words, the visibility graph is a t -spanner for the complete geometric graph on the point set V (where the obstacles are ignored).

Arikati *et al.* [2] have shown how to compute, in $O(n \log n)$ time, a $(1 + \varepsilon)$ -spanner G' of G , for any given constant $\varepsilon > 0$. Let p and q be two points of V , and assume that t is a

constant. Since G' is a $(1 + \varepsilon)t$ -spanner of V , our results above imply that we can compute, in $O(1)$ time, a $(1 + \varepsilon)$ -approximation of the shortest path P in G' between p and q . The length of this path P is at most $(1 + \varepsilon)^2$ times the length of a shortest obstacle-avoiding path between p and q .

Using conical Voronoi diagrams, Clarkson [13] and Chen [11] have shown that, for any two points p and q in the plane, the problem of computing a $(1 + \varepsilon)$ -approximation of the shortest obstacle-avoiding path between p and q can be reduced to the computation of a constant number of shortest path queries in the visibility graph. This reduction takes $O(\log n)$ time. (See also [2]). Hence, a $(1 + \varepsilon)^3$ -approximation of the shortest obstacle-avoiding path can be computed in $O(\log n)$ time.

Theorem 5. *Let \mathcal{F} be a t -rounded collection of polygonal obstacles in the plane of total complexity n , where t is a positive constant. One can preprocess \mathcal{F} in $O(n \log n)$ time into a data structure of size $O(n \log n)$ that can answer obstacle avoiding $(1 + \varepsilon)$ -approximate shortest distance queries in time $O(\log n)$.*

4.2 Approximate closest pair queries

Given a geometric graph $G = (V, E)$ on n points and $O(n)$ edges, such that G is a t -spanner for V , for some constant $t > 1$, it is often of interest to answer various closest pair queries where distances are measured according to distances in G . We show that our results can be applied to give approximate solutions to such queries.

The monochromatic case: For any subset S of V , we define

$$\delta_G(S) := \min\{\delta_G(p, q) : p, q \in S, p \neq q\}.$$

In a query, we get a set $S \subseteq V$, and want to compute two points x and y in V that are a $(1 + \varepsilon)$ -approximate closest pair in S , i.e., $\delta_G(S) \leq \delta_G(x, y) \leq (1 + \varepsilon)\delta_G(S)$. Here, ε is a fixed positive real constant.

Consider a well-separated pair decomposition (WSPD), as defined in Appendix 6, $\{A_i, B_i\}$, $1 \leq i \leq \ell$, where $\ell = O(|S|)$, for the set S , with separation constant $s > 2t$. For each i , $1 \leq i \leq \ell$, for which both A_i and B_i are singleton sets, let Δ_i be a $(1 + \varepsilon)$ -approximation to the length of a shortest path in G between a_i and b_i , which are the only points of A_i and B_i , respectively. Let i be an index for which Δ_i is minimum. We claim that the points $x := a_i$ and $y := b_i$ form a $(1 + \varepsilon)$ -approximate closest pair in S .

To prove this, let p and q be two points of S for which $\delta_G(p, q) = \delta_G(S)$, and let j be the index such that (i) $p \in A_j$ and $q \in B_j$, or (ii) $q \in A_j$ and $p \in B_j$. We may assume w.l.o.g. that (i) holds. The first claim is that both A_j and B_j are singleton sets, i.e., $A_j = \{p\}$ and $B_j = \{q\}$. (If this claim holds, then Δ_j is well-defined.) Indeed, assume the set A_j contains a point p' different from p . Then Lemma 11 and the fact that G is a t -spanner for V imply that

$$\delta_G(p, p') \leq t|pp'| \leq t(2/s)|pq| \leq t(2/s)\delta_G(p, q) < \delta_G(p, q),$$

which is a contradiction. Now we can easily complete the proof that x and y form a $(1 + \varepsilon)$ -approximate closest pair in S :

$$\delta_G(S) \leq \delta_G(x, y) \leq \Delta_i \leq \Delta_j \leq (1 + \varepsilon)\delta_G(p, q) = (1 + \varepsilon)\delta_G(S).$$

Hence, we have reduced the problem of computing a $(1 + \varepsilon)$ -approximate closest pair in S to computing a WSPD for S and answering $O(|S|)$ approximate shortest path queries in G .

Theorem 6. *Let $G = (V, E)$ be a geometric graph on n points and $O(n)$ edges, such that G is a t -spanner for V , for some constant $t > 1$. One can preprocess G in time $O(n \log n)$ into a data structure of size $O(n \log n)$ such that given a subset S of V , a $(1 + \varepsilon)$ -approximate monochromatic closest pair query can be answered in time $O(|S| \log |S|)$.*

The bichromatic case: For any two disjoint subsets X and Y of V , we define

$$\delta_G(X, Y) := \min\{\delta_G(p, q) : p \in X, q \in Y\}.$$

In a query, we get disjoint sets X and Y and want to compute a $(1 + \varepsilon)$ -approximate bichromatic closest pair, i.e., a point $x \in X$ and a point $y \in Y$ such that $\delta_G(X, Y) \leq \delta_G(x, y) \leq (1 + \varepsilon) \cdot \delta_G(X, Y)$. Again, ε is a fixed positive real constant.

Let $\{A_i, B_i\}$, $1 \leq i \leq \ell$, be a WSPD for the set $X \cup Y$, where $\ell = O(|X \cup Y|)$, with separation constant s , where $s > \max\{2t, 4t/\varepsilon\}$. For each i , $1 \leq i \leq \ell$, for which A_i contains one or more points of X but no points of Y , and B_i contains one or more points of Y but no points of X , let a_i be an arbitrary point of $A_i \cap X$, let b_i be an arbitrary point of $B_i \cap Y$, and let Δ_i be a $(1 + \varepsilon)$ -approximation to the length of a shortest path in G between a_i and b_i . Let i be an index for which Δ_i is minimum. We claim that the points $x := a_i$ and $y := b_i$ form a $(1 + \varepsilon)$ -approximate bichromatic closest pair in $X \cup Y$.

To prove this, let $p \in X$ and $q \in Y$ be points such that $\delta_G(p, q) = \delta_G(X, Y)$. Let j be the index such that (i) $p \in A_j$ and $q \in B_j$, or (ii) $q \in A_j$ and $p \in B_j$. We may assume w.l.o.g. that (i) holds. As in the monochromatic case, it can be shown (using the fact that $s > 2t$) that $A_j \cap Y = \emptyset$ and $B_j \cap X = \emptyset$. Therefore, the value Δ_j is well-defined. We have

$$\begin{aligned} \delta_G(a_j, b_j) &\leq \delta_G(a_j, p) + \delta_G(p, q) + \delta_G(q, b_j) \\ &\leq t|a_j p| + \delta_G(p, q) + t|q b_j| \\ &\leq t(2/s)|pq| + \delta_G(p, q) + t(2/s)|pq| \\ &\leq (1 + 4t/s) \cdot \delta_G(p, q). \end{aligned}$$

Also,

$$\delta_G(X, Y) \leq \delta_G(x, y) \leq \delta_G(a_j, b_j) \leq \Delta_j \leq \Delta_i \leq (1 + \varepsilon) \cdot \delta_G(a_j, b_j).$$

Combining these inequalities, it follows that

$$\delta_G(X, Y) \leq \Delta_i \leq (1 + \varepsilon)(1 + 4t/s) \cdot \delta_G(X, Y) \leq (1 + \varepsilon)^2 \cdot \delta_G(X, Y),$$

where the last inequality follows from the fact that $s \geq 4t/\varepsilon$.

Hence, we have reduced the problem of computing a $(1 + \varepsilon)$ -approximate bichromatic closest pair to computing a WSPD for $X \cup Y$ and answering $O(|X| + |Y|)$ approximate shortest path queries in G .

Theorem 7. *Let $G = (V, E)$ be a geometric graph on n points and $O(n)$ edges, such that G is a t -spanner for V , for some constant $t > 1$. One can preprocess G in time $O(n \log n)$ into a data structure of size $O(n \log n)$ such that given two disjoint subsets X and Y of V an $(1 + \varepsilon)$ -approximate bichromatic closest pair query can be answered in time $O((|X| + |Y|) \log(|X| + |Y|))$.*

4.3 Approximating the dilation of a geometric graph

Given a geometric graph $G = (V, E)$ on n points and $O(n)$ edges, it was shown by Narasimhan and Smid [28] that the problem of computing a $(1 + \varepsilon)$ -approximation to the dilation t of G (for any given $\varepsilon > 0$), can be reduced to the problem of computing $O(n)$ shortest path queries after computing a well-separated decomposition that takes $O(n \log n)$ time.

Now assume that we are given a constant C which is an upper bound on the dilation t of G . Using the current result, we can answer the $O(n)$ shortest path queries in $O(n)$ time after $O(n \log n)$ time preprocessing computation, giving a total computation time of $O(n \log n)$. This improves the existing time complexity for planar graphs from $O(n\sqrt{n})$ to $O(n \log n)$ time. It results in a considerable improvement (both in terms of time and approximability) for arbitrary geometric graphs, for which the time complexity decreases from $O(n^{1+\frac{1}{1+\varepsilon}} \log^2 n)$ with $(2(1 + \varepsilon))$ -approximation factor to $O(n \log n)$ with $(1 + \varepsilon)$ -approximation factor.

Theorem 8. *Given a geometric graph on n vertices with $O(n)$ edges, and given a constant C that is an upper bound on the dilation t of G , one can compute a $(1 + \varepsilon)$ -approximation to t in time $O(n \log n)$.*

5 Efficient bucketing of distances

In this section, we develop the algorithmic tool that was used in Section 2.3.

Let V be a set of n points in the open hypercube $(0, n^k)^d$, where k is a positive integer constant and d is the number of dimensions. Our goal is to preprocess the points of V into a data structure, such that for any two points p and q in V for which $|pq| \geq 1$, we can efficiently answer the query of computing the integer $\text{BINDEX}(p, q)$ defined as

$$\text{BINDEX}(p, q) := \lfloor \log |pq| \rfloor.$$

Observe that $\text{BINDEX}(p, q)$ is the unique non-negative integer i such that the distance $|pq|$ is in the interval $[2^i, 2^{i+1})$, and that $0 \leq \text{BINDEX}(p, q) \leq \lfloor (1/2) \log d + k \log n \rfloor$.

In a computation model that has the floor and logarithm function as unit-time operations, such a query can clearly be answered in $O(1)$ time. The main result of this section is a data structure, having size $O(n)$, that can be built in $O(n \log n)$ time, and that can be used to answer queries in $O(1)$ time. We start by presenting this data structure for the one-dimensional case. In Section 5.3, we show how to use this result to solve the d -dimensional case.

5.1 The one-dimensional case

We will assume that V is a set of n real numbers in the interval $(0, n^k)$. For any two elements $x, y \in V$ with $|x - y| \geq 1$, we have $\text{BINDEX}(x, y) = \lfloor \log |x - y| \rfloor$, which is an integer between zero and $\lfloor k \log n \rfloor$. We assume w.l.o.g. that n is a power of two.

Let T_0 be the perfectly balanced binary search tree whose leaves store—from left to right—the intervals $[j, j + 1)$, $0 \leq j < n^k$. With each internal node u of T_0 , we store the interval $I(u)$, which is the union of the intervals stored at the leaves in the subtree rooted at u . Observe that $I(u)$ has the form $[a, b)$, for some real numbers a and b , where $b - a$ is a power of two. Distribute the elements of V over the leaves of T_0 . That is, we store each element x of V in the unique leaf whose interval contains x .

This tree T_0 has n^k leaves, and each of them stores a possibly empty subset of V . Let T be the tree obtained from T_0 by performing the following *compression steps* as long as possible:

- Delete the subtree rooted at any node u for which the interval $I(u)$ does not contain any element of V .
- For any node u having only one child v , delete u and make v the child of u 's parent.

The resulting tree T does not depend on the order in which the compression steps are made. Also, T has at most n leaves, and each internal node has exactly two children. Hence, T has a total of at most $2n - 1$ nodes. Since the height of T_0 is at most $k \log n$ the height of T is $O(\log n)$, since k is a constant.

We show how the compressed tree T can be constructed in a top-down manner, without first constructing T_0 . In a generic step, we have a subset V' of V and an interval $[a, b)$, where $b - a$ is a power of two, such that $V' \subseteq [a, b)$, and we want to compute the compressed tree $T(V')$ for V' .

- If $b - a = 1$ or V' contains only one element, then $T(V')$ consists of only one node u storing the interval $I(u) := [a, b)$ and the element(s) of V' .
- If $b - a \geq 2$, $|V'| \geq 2$ and $V' \subseteq [a, (a + b)/2)$, then $T(V')$ is the output of the recursive call for V' and the interval $[a, (a + b)/2)$.
- If $b - a \geq 2$, $|V'| \geq 2$ and $V' \subseteq [(a + b)/2, b)$, then $T(V')$ is the output of the recursive call for V' and the interval $[(a + b)/2, b)$.
- Otherwise, V' is partitioned into two sets $V'_1 := \{x \in V' : x < (a + b)/2\}$ and $V'_2 := \{x \in V' : x \geq (a + b)/2\}$. In this case, $T(V')$ consists of a node u storing the interval $I(u) := [a, b)$ and whose left (resp. right) subtree is the output of the recursive call for the set V'_1 and the interval $[a, (a + b)/2)$ resp. the set V'_2 and the interval $[(a + b)/2, b)$.

The complete compressed tree $T = T(V)$ is built by running this algorithm on the set V and the interval $[0, n^k)$. We can easily extend the algorithm such that each node u stores the non-negative integer i where the interval $I(u)$ has length 2^i . The running time to build this tree is $O(n \log n)$.

5.2 Answer a query

Let us see how we can use this tree to answer queries. Let x and y be two elements of V with $|x - y| \geq 1$. Recall that we want to compute the integer $\text{BINDEXT}(x, y) = \lfloor \log |x - y| \rfloor$. We may assume w.l.o.g. that $x < y$. We observe that $\text{BINDEXT}(x, y)$ is the exponent of the length of the largest interval whose length is a power of two and that fits in the interval $[x, y]$.

Let u be the lowest common ancestor of the leaves of T that store x and y . Note that x and y are stored at different leaves. Let $a \geq 0$, $b \geq 2$, and $j \geq 1$ be the integers such that $I(u) = [a, b)$ and $b - a = 2^j$. Observe that $x \in [a, a + 2^{j-1})$ and $y \in [a + 2^{j-1}, b)$. If at least one of x and y is “far” away from the mid-point $c := a + 2^{j-1}$, then j differs from $\text{BINDEXT}(x, y)$ by a “small” additive constant. It may happen, however, that both x and y are “close” to c . Let us assume that $|c - x| \leq |y - c|$. Then $|y - c| < |y - x| \leq 2|y - c|$. Hence, if j' is the exponent of the largest interval whose length is a power of two and that fits in the interval $[c, y]$, then j' differs from $\text{BINDEXT}(x, y)$ by a “small” additive constant. This suggests that we find the lowest common ancestor v of the leaves of T whose intervals contain c and y . This does not, in general, give us a good approximation to j' , because c and y can be “close” to each other in the interval $I(v)$. Assume, however, that c is an element of V . Let w be the right child of u . Hence, $I(w) = [c, b)$, c is stored in the leftmost leaf of the subtree rooted at w , and y is

stored in the subtree of w . Consider again the lowest common ancestor v of the leaves storing c and y . The nodes on the path starting in w and ending in v store the intervals

$$[c, b), [c, c + 2^{j-2}), [c, c + 2^{j-3}), \dots, [c, c + 2^{j'+1}),$$

where j' is as above. Note that this does not necessarily hold if c does not belong to the set V .

This suggests the following data structure for solving our query problem. Let V' be the union of V and the set of all mid-points of the intervals $I(u)$ over all internal nodes u of T . Observe that V' contains at most $2n - 1$ elements. We build a new compressed tree T' using the algorithm given above, but for the set V' .

Given two elements x and y in V with $|x - y| \geq 1$ and $x < y$, we first test if $y - x = 1$ or $1 < y - x < 2$. In the first case, we have $\text{INDEX}(x, y) = 0$, whereas $\text{INDEX}(x, y) = 1$ in the second case. Assume that $y - x \geq 2$. Compute the lowest common ancestor u of the leaves of T' that store x and y . Let c be the mid-point of the interval $I(u)$. If $|c - x| \leq |y - c|$, then we compute the lowest common ancestor v of the leaves storing c and y . Let j be the positive integer such that $I(v)$ has length 2^j . Since $y - c \geq 1$, the elements y and c are stored at different leaves of T . Then $2^{j-1} \leq |y - c| < 2^j$ and, hence, $2^{j-1} < |y - x| \leq 2^{j+1}$. This implies that $j - 1 \leq \text{INDEX}(x, y) \leq j + 1$. Since we know the values of j (stored with node v) and 2^j (the length of the interval $I(v)$), we can now easily compute $\text{INDEX}(x, y)$ in constant time.

We cannot use this data structure if $|c - x| > |y - c|$. In order to handle this case, we also build a compressed tree using intervals of the form $(a, b]$ instead of $[a, b)$.

To summarize, we have shown that after an $O(n \log n)$ -time preprocessing, we can reduce the problem of computing $\text{INDEX}(x, y)$ to answering two lowest common ancestor queries in a tree having size $O(n)$, plus an $O(1)$ -time computation. Harel and Tarjan [24] showed that, any tree can be preprocessed in linear time such that lowest common ancestor queries can be answered in $O(1)$ time. See also Schieber and Vishkin [31], Gusfield [23] and Bender and Farach-Colton [6]. Hence, we have proved the following result.

Theorem 9. *Let V be a set of n real numbers that are contained in the interval $(0, n^k)$, for some positive integer constant k . We can preprocess V in $O(n \log n)$ time into a data structure of size $O(n)$, such that for any two elements x and y of V , with $|x - y| \geq 1$, we can compute $\text{INDEX}(x, y) = \lfloor \log |y - x| \rfloor$ in $O(1)$ time.*

It should be clear that this theorem also holds if we allow queries with elements $x, y \in V$ such that $|x - y| \geq \delta$, for some fixed constant $\delta > 0$.

5.3 The d -dimensional case

Now assume that V is a d -dimensional set of points, where d is a constant. Let $p = (p_1, p_2, \dots, p_d)$ and $q = (q_1, q_2, \dots, q_d)$ be any two points of V with $|pq| \geq 1$, let j be such that $|p_j - q_j|$ is maximum, and let $i = \lfloor \log |p_j - q_j| \rfloor$. Since

$$|p_j - q_j| \leq |pq| \leq \sqrt{d} |p_j - q_j|,$$

we have

$$i \leq \text{INDEX}(p, q) \leq \frac{1}{2} \log d + i.$$

This suggests the following solution. For each ℓ , $1 \leq \ell \leq d$, we build the data structure of Theorem 9 for the set of ℓ -th coordinates of all points of V .

Given two distinct points p and q of V , we compute the index j such that $|p_j - q_j|$ is maximum. Then we use the algorithm of Theorem 9 to compute the integer $i = \lfloor \log |p_j - q_j| \rfloor$. Note that this algorithm also gives us the value 2^i . Given i and 2^i , we then compute $\text{BINDEX}(p, q)$ in $O(\log \log d)$ time. Observe that we can indeed apply Theorem 9, because $|p_j - q_j| \geq 1/\sqrt{d}$. This gives the following result.

Theorem 10. *Let V be a set of n points in \mathbb{R}^d that are contained in the hypercube $(0, n^k)^d$, for some positive integer constant k . We can preprocess V in $O(n \log n)$ time into a data structure of size $O(n)$, such that for any two points p and q of V , with $|pq| \geq 1$, we can compute*

$$\text{BINDEX}(p, q) = \lfloor \log |pq| \rfloor$$

in $O(1)$ time.

Until now, the values BINDEX were based on the binary logarithm. Let ε be a positive real constant, and assume that we want to compute the integer

$$\text{BINDEX}_\varepsilon(p, q) := \lfloor \log_{1+\varepsilon} |pq| \rfloor = \lfloor \log |pq| / \log(1 + \varepsilon) \rfloor,$$

where p and q are two points in V with $|pq| \geq 1$. Hence, $\text{BINDEX}_\varepsilon(p, q)$ is the integer i such that $(1 + \varepsilon)^i \leq |pq| < (1 + \varepsilon)^{i+1}$. Let $j := \text{BINDEX}(p, q)$. Then a straightforward calculation shows that

$$\left\lfloor \frac{j}{\log(1 + \varepsilon)} \right\rfloor \leq \text{BINDEX}_\varepsilon(p, q) \leq \left\lceil \frac{j + 1}{\log(1 + \varepsilon)} \right\rceil - 1. \quad (7)$$

Hence, if we know j , then we can compute $\text{BINDEX}_\varepsilon(p, q)$ in $O(1 + 1/\log(1 + \varepsilon))$ time, provided that we know the values $\lfloor j/\log(1 + \varepsilon) \rfloor$ and $(1 + \varepsilon)^i$ for all i in the range given in (7). Note that j is a non-negative integer that is bounded by $O(\log n)$, and the same holds for i . In an $O(n)$ -time preprocessing step, we can easily compute two arrays of length $O(\log n)$, containing all possible values of $\lfloor j/\log(1 + \varepsilon) \rfloor$ and $(1 + \varepsilon)^i$. Then given j , we can use this array to compute $\text{BINDEX}_\varepsilon(p, q)$ in time $O(1/\log(1 + \varepsilon))$.

Corollary 1. *Let V be a set of n points in \mathbb{R}^d that are contained in the hypercube $(0, n^k)^d$, for some positive integer constant k , and let ε be a positive real constant. We can preprocess V in $O(n \log n)$ time, such that for any two points p and q of V , with $|pq| \geq 1$, we can in constant time compute*

$$\text{BINDEX}_\varepsilon(p, q) = \lfloor \log_{1+\varepsilon} |pq| \rfloor.$$

6 Concluding remarks

We have presented the first data structure which supports $(1 + \varepsilon)$ -approximate shortest path queries in constant time for geometric t -spanners, hence functions as an approximate distance oracle. In the process we develop several tools that we believe are useful for other geometric problems. We also give several applications for our data structure, for example, closest pair

queries, shortest path queries between vertices in a planar polygonal domain, and efficiently computing the approximate dilation of geometric graphs.

Even though the results in this paper are restricted to geometric graphs with constant dilation we believe that the results are of great importance since many naturally occurring geometric graphs have constant dilation. As we already pointed out in the introduction, Keil and Gutwin [25] showed that this is true for the Delaunay triangulation, which is used in numerous applications. Also, different kinds of transportation networks have small dilation.

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(4):1167–1181, 1999.
2. 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.
3. S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In Proc. *35th IEEE Symposium on Foundations of Computer Science*, 703–712, 1994.
4. S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33–54, 1997.
5. S. Baswana and S. Sen. Approximate Distance Oracles for Unweighted Graphs in $O(n^2 \log n)$ Time. In Proc. *15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 271–280, 2004.
6. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In Proc. *4th Latin American Theoretical Informatics*, LNCS 1776, pp. 88–94, 2000.
7. M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica* 34(1): 81-97, 2002).
8. P. B. Callahan. Dealing with higher dimensions: the well-separated pair decomposition and its applications. Ph.D. Thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1995.
9. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.
10. P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In Proc. *4th ACM-SIAM Symposium on Discrete Algorithms*, pp. 291–300, 1993.
11. D. Z. Chen. On the all-pairs Euclidean short path problem. In Proc. *6th ACM-SIAM Symposium on Discrete Algorithms*, pp. 292–301, 1995.
12. 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(4):1223–1246, 2000.
13. K. L. Clarkson. Approximation algorithms for shortest path motion planning. In Proc. *19th ACM Symposium on Theory of Computing*, pp. 56–65, 1987.
14. 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*, pp. 215–224, 1999.
15. E. Cohen. Fast algorithms for constructing t -spanners and paths with stretch t . *SIAM Journal on Computing*, 28(1):210–236, 1998.
16. G. Das and D. Joseph. Which triangulations approximate the complete graph? In Proc. *International Symposium on Optimal Algorithms*, Lecture Notes in Computer Science, Vol. 401, Springer Verlag, Berlin, pp. 168–192, 1989.
17. G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7(4):297–315, 1997.
18. D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.
19. J. Gao, L. J. Guibas, J. Hershberger, L. Zhang and A. Zhu. Discrete Mobile Centers. *Discrete & Computational Geometry* 30(1):45–63, 2003.
20. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479-1500, 2002.
21. J. Gudmundsson, C. Levcopoulos, G. Narasimhan and M. Smid. Approximate distance oracles for geometric graphs. In Proc. *13th ACM-SIAM Symposium on Discrete Algorithms*, pp. 828–837, 2002.
22. J. Gudmundsson, G. Narasimhan and M. Smid. Fast pruning of spanners. Submitted, 2004. <http://www.win.tue.nl/~hgudmund/gns-fps-04.pdf>

23. D. Gusfield. Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge, UK, 1997.
24. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
25. J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete Computational Geometry*, 7:13–28, 1992.
26. J. B. Kruskal, Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proc. Amer. Math. Soc.*, 7(1956):48–50, 1956.
27. J. S. B. Mitchell. Shortest paths and networks. In *Handbook of Discrete and Computational Geometry*, pp. 445–466. CRC Press LLC, 1997.
28. G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2000.
29. M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *Journal of Algorithms*, 21(3):629–656, 1996.
30. J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry and Applications*, 1(2):99–107, 1991.
31. B. Schieber and U. Vishkin. On finding lowest common ancestors: simplifications and parallelisations. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.
32. M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. 33rd ACM Symposium on Theory of Computing*, pp. 183–192, 2001.
33. A. C. Yao. On constructing minimum spanning trees in k -dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Appendix A: The well-separated pair decomposition

We recall the definition and basic facts about the well-separated pair decomposition. The details can be found in Callahan and Kosaraju [8, 9].

Definition 1. Let $s > 0$ be a real number, and let A and B be two finite sets of points in \mathbb{R}^d . We say that A and B are well-separated w.r.t. s , if there are two disjoint d -dimensional balls C_A and C_B , having the same radius, such that (i) C_A contains all points of A , (ii) C_B contains all points of B , and (iii) the distance between C_A and C_B is at least equal to s times the radius of C_A .

The parameter s will be referred to as the *separation constant*. The following lemma follows easily from Definition 1.

Lemma 11. Let A and B be two finite sets of points that are well-separated w.r.t. s , let x and p be points of A , and let y and q be points of B . Then (i) $|xy| \leq (1 + 2/s) \cdot |xq|$, (ii) $|xy| \leq (1 + 4/s) \cdot |pq|$, and (iii) $|px| \leq (2/s) \cdot |pq|$.

Definition 2 ([9]). Let V be a set of n points in \mathbb{R}^d , and $s > 0$ a real number. A well-separated pair decomposition (WSPD) for V (w.r.t. s) is a sequence of pairs of non-empty subsets of V , $\{A_1, B_1\}, \{A_2, B_2\}, \dots, \{A_\ell, B_\ell\}$, such that

1. $A_i \cap B_i = \emptyset$, for all $i = 1, 2, \dots, \ell$,
2. for any two distinct points p and q of V , there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $q \in A_i$ and $p \in B_i$,
3. A_i and B_i are well-separated w.r.t. s , for all $i = 1, 2, \dots, \ell$.

The integer ℓ is called the size of the WSPD.

Callahan and Kosaraju show that a WSPD of size $\ell = O(n)$ can be computed in $O(n \log n)$ time. Their algorithm uses a binary tree T , called the *split tree*, which is similar to a kd -tree. We briefly describe the main idea. They start by computing the bounding box of V , which is successively split by d -dimensional hyperplanes, each of which is orthogonal to one of the axes. If a box is split, they take care that each of the two resulting boxes contains at least one point of V . As soon as a box contains exactly one point, the process stops (for this box). The resulting binary tree T stores the points of V at its leaves; one leaf per point. Also, each node u of T is associated with a subset of V . We denote this subset by V_u ; it is the set of all points of V that are stored in the subtree of u .

The split tree T can be computed in $O(n \log n)$ time. Callahan and Kosaraju show that, given T , a WSPD of size $\ell = O(n)$ can be computed in $O(n)$ time. Each pair $\{A_i, B_i\}$ in this WSPD is represented by two nodes u_i and v_i of T , i.e., we have $A_i = V_{u_i}$ and $B_i = V_{v_i}$.