Shortest-Path Queries in Geometric Networks

Eunjin Oh
Department of Computer Science and Engineering, POSETCH, Pohang, South Korea
eunjin.oh@postech.ac.kr

Abstract

A Euclidean t-spanner for a point set $V \subset \mathbb{R}^d$ is a graph such that, for any two points p and q in V, the distance between p and q in the graph is at most t times the Euclidean distance between p and q. Gudmundsson et al. [TALG 2008] presented a data structure for answering ϵ-approximate distance queries in a Euclidean spanner in constant time, but it seems unlikely that one can report the path itself using this data structure. In this paper, we present a data structure of size $O(n \log n)$ that answers ϵ-approximate shortest-path queries in time linear in the size of the output.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Shortest path, Euclidean spanner, data structure

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.52

Funding This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2020R1C1C1012742).

1 Introduction

Computing the shortest path between two vertices in a graph is a fundamental problem that has numerous applications such as route planning, geographic information systems, intelligent transportation systems, and web-search ranking [25]. Due to its various applications, the shortest path problem and its variants have been extensively studied for the last several decades. In a query variant, the goal is to process a given graph so that the shortest path between two query vertices and its length can be computed efficiently. Since a data structure for computing the exact distance requires large space, researchers have focused on designing data structures for answering *approximate* distance and path queries. We call a path in a graph G between two vertices u and v an *ϵ-approximate path* for a constant $\epsilon > 0$ if its length is at most $(1+\epsilon)d_G(u,v)$, where $d_G(u,v)$ denotes the distance in G between u and v. Similarly, any value between $d_G(u,v)$ and $(1+\epsilon)d_G(u,v)$ is called an *ϵ-approximate distance* between u and v.

In this paper, we present a data structure for answering *approximate shortest-path queries* in a geometric network which is called a *Euclidean spanner*.

Data structure for shortest-path & distance queries. For a weighted graph, Thorup and Zwick [27] showed that any data structure for answering $(2k+1)$-approximate distance queries in $O(1)$ time must use $\Omega(n^{1+1/k})$ space assuming the 1963 girth conjecture of Erdős, where n denotes the size of the graph. In addition to this, several lower bounds on the space requirements of data structures for approximate distance queries have been presented under several different conjectures. For details, refer to [26].

Thorup and Zwick [27] presented a data structure of size $O(kn^{1+1/k})$ for answering $(2k-1)$-approximate distance queries in $O(k)$ time for any integer $k > 0$. To emphasize that the query time is constant, they call their data structure an approximate distance *oracle*. A lot of data structures for answering the shortest-path and distance queries have been presented [1, 22, 23, 24, 27].

For special classes of graphs, we can obtain data structures with faster query times and low space complexities. For planar graphs, there exists a data structure of near-linear space for computing the exact distance between any two query vertices in polylogarithmic time [6, 10]. Also, approximate distance oracles for planar graphs, bounded-genus, and minor-free graphs are known [5, 11, 19]. Geometric versions of the approximate distance and shortest path problem also have been considered. There are numerous results on the shortest path problem and its query variant in polygonal domains [15, 16, 18], disk intersection graphs [4, 9], and Euclidean spanners [12].

Euclidean spanner. In this paper, we consider the shortest path problem for a geometric network which is called a *Euclidean spanner* for a set of points in d-dimensional Euclidean space. Let V be a set of n points in d-dimensional Euclidean space for a constant $d \geq 1$. A graph $G = (V, E)$ is called a t-spanner for V if, for any two points p and q in V, the distance in G between p and q is at most t times the Euclidean distance between p and q. Euclidean spanners have various applications including pattern recognition, function approximation, and broadcasting in communication networks [21]. A Euclidean spanner that has small size, bounded degree, small diameter, and small total weight can be computed efficiently [8, 20].

1.1 Previous Results

Our problem is closely related to the problem of constructing a data structure for answering approximate *distance* queries in a Euclidean spanner, which was introduced by Gudmundsson et al. [12]. They presented an approximate distance oracle for a Euclidean spanner in d-dimensional space. More specifically, given a d-dimensional Euclidean t-spanner G and a constant $\epsilon > 0$, they present a data structure of size $O(n \log n)$, which can be computed in $O(m + n \log n)$, so that given any two vertices p and q, an ϵ-approximate distance in G between p and q of V can be computed in constant time, where n denotes the number of vertices and m denotes the number of edges. Here, all the big-Oh notations hide constants depending on d, t and ϵ.

Distance query vs. Shortest-path query. As mentioned in [26], most distance oracles can be used to compute not only the distances but also the actual paths. This is because after having computed the distance, most distance oracles have an implicit representation of the path such that each edge can be output efficiently.

However, it seems unclear if the data structure by Gudmundsson et al. [12] can be used for reporting an approximate path in a Euclidean spanner. Indeed, the authors of [12] also mentioned this in [12, 14]. The main difficulty here is that they convert G into graphs G_i's such that the edges of each G_i are not necessarily contained in G, for $i = 1, \ldots, m$ for $m \leq n$. After converting G, they handle G_i's instead of G. Their query algorithm returns the length of a path π of G_i, but such a path is not necessarily a path of G. Using the fact that G is a Euclidean t-spanner, they showed that the length of π approximates the distance in G between two query points. However, it is not difficult to construct an example that π is not a path of G, that is, π contains edges not contained in the edge set of G, and π connects two vertices which are not the query points.

1.2 Our Results

In this paper, we present a data structure for answering shortest-path queries approximately and efficiently on a Euclidean spanner for a point set in d-dimensional Euclidean space. More specifically, our data structure has size of $O(n \log n)$ and can be constructed in $O(m + n \log n)$

time, where n denotes the number of vertices and m denotes the number of edges of the spanner. The data structure allows us to returns the edges of an ϵ-approximate path in constant time per edge. As in [12], all the big-Oh notations hide constants depending on d, t and ϵ.

Why do we need the actual path? For some applications, it is necessary to obtain the actual path as well as the distance. Imagine that we want to design an efficient navigation system. Lots of road networks can be represented as Euclidean t-spanners for a small constant $t > 1$ [21]. For instance, a part of the southern Scandinavian railroad network is a 1.85-spanner [21]. To design an efficient navigation system, given a starting position and a destination, we are required to find a *shortest path* (or an *approximate shortest path*) in the road network efficiently. The data structure of [12] tells us *how long* it takes from the starting position to the destination, but it does not give *how* to get there. This is not sufficient for navigation systems.

Actual path vs. Approximate path. Our algorithm returns an approximate path in time linear in its complexity. It is possible that an approximate path has a larger complexity than the actual path. However, it seems not an serious issue for many applications because our algorithm returns the edges one by one in constant time per edge.

Imagine that we use a navigation system on a highway. Our goal is to find the path from the starting point to the destination to minimize the travel time. The travel time depends on the sum of lengths of the edges of the path, not on the number of edges of the path. Given the starting point and a destination, our algorithm process them in constant time, and then returns the edges one by one in constant time per edge. To traverse the path, it sufficient to have the next edge of the current edge on the path. Since a path lying between two interchanges in a highway, which is represented as an edge of a graph, is sufficiently long, the computation time is subsumed by the travel time. Therefore, even if an approximate path has a larger complexity than the actual path, the travel time is within $(1 + \epsilon)$ times the optimal travel time in this case.

1.3 Preliminaries

Let $d_G(u, v)$ denote the distance in a graph G between two vertices u and v, and let $|pq|$ denote the Euclidean distance between two points p and q. For two values a and b in $\mathbb{R}$, we use $[a, b]$ to denote the set of all values lying between a and b including a and b, $[a, b)$ to denote $[a, b] \setminus \{b\}$, and (a, b) to denote $[a, b] \setminus \{a, b\}$. With a slight abuse of notation, for a finite set A, we use $|A|$ to denote the cardinality of A. For a path π in a Euclidean space, we use $w(\pi)$ to denote the length of π.

A subgraph H of a graph G is called a *t-spanner* of G if the vertex set of H coincides with the vertex set of G, and $d_H(v, u) \leq t d_G(v, u)$ for any two vertices v and u in H. We call a graph $G = (V, E)$ a *geometric graph* if each vertex of V corresponds to a point in a Euclidean space and each edge pq has weight $|pq|$. Let V be a set of n points in d-dimensional Euclidean space for a constant d. For a value $t \geq 1$, a geometric graph G is called a *t-spanner for V* if it is a t-spanner of the complete geometric graph whose vertex set is V, that is, $d_G(p, q) \leq t|pq|$ for any two points p and q in V. Similarly, for two values $L > 0$ and $t > 1$, a geometric graph G is called an *L-partial t-spanner* for V if $d_G(p, q) \leq t|pq|$ for any two points p and q in V with $|pq| \leq L$.

For a fixed $\epsilon > 0$, our goal is to construct a data structure on a t-spanner G for a point set V in d-dimensional Euclidean space so that, given any two points p and q in V, an ϵ-approximate path and its length can be computed efficiently. To make the description easier, we assume that G has $O(n)$ edges as in [12], where n is the number of vertices of G. If it is not the case, we compute a $(1+\epsilon)$-spanner G' of G of size $O(n)$ using the following lemma, and construct a data structure for G'. Here, notice that G' is a subgraph of G by definition.

▶ **Lemma 1** ([13]). *For a t-spanner G with n vertices and m edges, we can compute a $(1+\epsilon)$-spanner of G with $O(n)$ edges in $O(m+n\log n)$ time.*

The model of computation we use in this paper is the same as the one in [12], which is the traditional algebraic computation model with the added power of indirect addressing.

2 Overall Data Structure and Query Algorithm

Given a t-spanner $G=(V,E)$ with n vertices and $O(n)$ edges, and a constant $\epsilon>0$, we can construct a data structure of size $O(n\log n)$ in $O(m+n\log n)$ time so that the ϵ-approximate path between any two points can be computed in constant time per edge.

Overall structure of [12]. The data structure of [12] consists of several substructures. Each substructure is constructed for a value L in $\mathbb{R}$ (which might be a function of n) and a subset W_L of V. This allows us to compute an approximate path between two query points p and q contained in W_L such that their Euclidean distance lies in $[L/n, L/t]$. The size of the substructure for L is near linear in the complexity of W_L.

They partition the edge set with respect to their lengths such that the edge lengths within each subset differ by a factor of $n^{O(1)}$ of each other. Let $\mathcal{L}=\langle L_1, L_2, \ldots, L_r\rangle$ be the lengths of the longest edges contained in the subsets. For each index i with $1\le i\le r$, they choose a small subset W_i of V so that for any two query points p and q, there are two vertices x and y in W_i, and an index i such that $|xy|\in[L_i/n, L_i/t]$ and $|px|+|yq|$ is sufficiently small. Using this property, for any two query points p and q, they find an index i and two vertices x and y of W_i satisfying the properties mentioned above, and compute an approximate distance between x and y instead of the distance between p and q.

Overall structure of our data structure. As in [12], we partition the edge set with respect to their lengths, but we choose a subset V_i of V for each index in a way different from [12]. Then we show that if the Euclidean distance between two query points p and q lies in $[L_i/n, L_i/t]$, there are two vertices x and y in V_i such that $|px|+|yq|$ is sufficiently small. Hence, it suffices to compute the paths between x and p, between y and q, and between x and y. Notice the difference between ours and [12]: In [12], for any two query points p and q, there are two vertices x and y of W_i for some index i with $|xy|\in[L_i/n, L_i/t]$ such that $d_{W_i}(x,y)$ approximates $d_G(p,q)$. However, in our case, if $|pq|\in[L_i/n, L_i/t]$ for an index i, then there are two vertices x and y in V_i such that $d_{V_i}(x,y)$ approximates $d_G(p,q)$.

In our case, there might be two points p and q such that $|pq|\notin[L_i/n, L_i/t]$ for any index i. This means that not all queries can be answered using our data structure stated above. To handle this issue, we construct five different partitions (sequences) $\mathcal{L}_1,\ldots,\mathcal{L}_5$ of the edge lengths such that, for any two points p and q, one of the five sequences contains a value L with $|pq|\in[L/n, L/t]$. Then we can use the data structure constructed for the partition (sequence) to compute an ϵ-approximate path between p and q. Figure 1 illustrates the overall structure of our data structure.

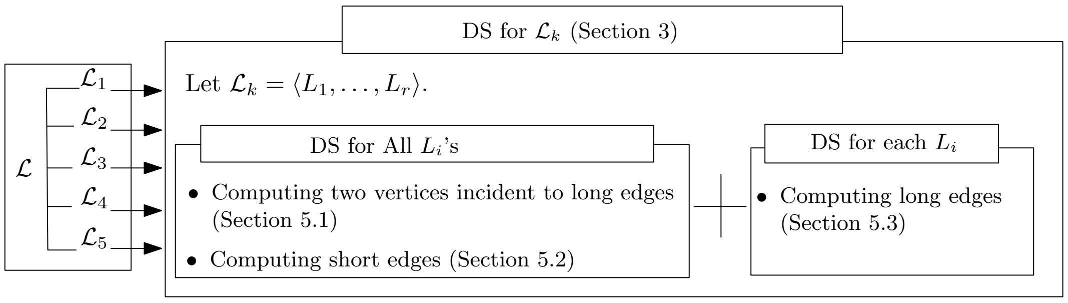
DS for $\mathcal{L}_k$ (Section 3)
Let $\mathcal{L}_k = \langle L_1, \ldots, L_r \rangle$.
$\mathcal{L}$
$\mathcal{L}_1$
$\mathcal{L}_2$
$\mathcal{L}_3$
$\mathcal{L}_4$
$\mathcal{L}_5$
DS for All L_i's
• Computing two vertices incident to long edges (Section 5.1)
• Computing short edges (Section 5.2)
DS for each L_i
• Computing long edges (Section 5.3)

Figure 1 Illustration for the overall data structure. Given two query points p and q, the query algorithm finds one element, say L_i, in one sequence, say $\mathcal{L}_k$. In Section 6.1, we show how to find the first and last bridge points, $\bar{p}$ and $\bar{q}$, from p to q along an approximate path in G. In Section 6.2, we show how to compute short edges in a path between p and q. Then the sequence of the long edges and short edges computed so far forms an approximate path in G between p and q. In Section 6.3, we show how to compute the long edges an approximate path between $\bar{p}$ and $\bar{q}$.

Classifying lengths. Let $\mathcal{L} = \langle L_1, L_2, \ldots, L_{r-1} \rangle$ be a sequence of values in $\mathbb{R}$ such that L_1 is the length of a shortest edge of G, and L_i is n^2 times the length of a shortest edge among all edges of length larger than L_{i-1} for an integer $i \geq 2$, and $[L_1, L_{r-1})$ contains all edge lengths. Then for a technical reason, we add $L_r = n^2 L_{r-1}$ at the end of $\mathcal{L}$. Note that $L_i \geq n^2 L_{i-1}$ for every integer i with $2 \leq i \leq r$. Moreover, for any two vertices of G, their distance in G lies in $[L_1, L_r)$ because a shortest path in G consists of at most n edges, and thus the length of any edge in G also lies in $[L_1, L_r)$ because G is a t-spanner for a constant t. For a sequence $\mathcal{L}'$ of lengths, let $\bar{\mathcal{L}}'$ be the union of $[L'/n, L'/t]$ over all lengths L' in $\mathcal{L}'$. Our goal is to obtain five sequences $\mathcal{L}_1, \ldots, \mathcal{L}_5$ such that the distance in G between any two vertices lies in the union of $\bar{\mathcal{L}}_1, \ldots, \bar{\mathcal{L}}_5$.

For a value x in $\mathbb{R}$, we use $x\mathcal{L}$ to denote the sequence $\langle xL_1, xL_2, \ldots, xL_r \rangle$. Consider the following five sequences:

$$\mathcal{L}_1 = t\mathcal{L}, \quad \mathcal{L}_2 = (t^2/n)\mathcal{L}, \quad \mathcal{L}_3 = (t^3/n^2)\mathcal{L}, \quad \mathcal{L}_4 = nt\mathcal{L}, \quad \mathcal{L}_5 = t^2\mathcal{L}$$

The following lemma shows that given two points p and q, one of the five sequences contains a value L such that $|pq| \in [L/n, L/t]$. Also, as we will see in Section 3, we can find such a sequence and such a value L in constant time.

▶ **Lemma 2.** *For any two points p and q, the distance in G between p and q is contained in the union of $\bar{\mathcal{L}}_1, \ldots, \bar{\mathcal{L}}_5$.*

Proof. Let p and q be any two points in V. By the definition of $\mathcal{L}$, there is an integer i with $1 < i \leq r$ such that $|pq| \in [L_{i-1}, L_i]$. Consider the partition of $[L_{i-1}, L_i)$ into six intervals such that the endpoints of the intervals are

$$L_{i-1}, \quad tL_{i-1}, \quad nL_{i-1}, \quad t^3L_i/n^3, \quad t^2L_i/n^2, \quad tL_i/n, \quad L_i.$$

We claim that $|pq|$ does not lies in $(nL_{i-1}, t^3L_i/n^3)$, for a sufficiently large n. Assume to the contrary that $|pq| \in (nL_{i-1}, t^3L_i/n^3)$. Then $d_G(p,q) \leq t|pq| < t^4L_i/n^3 < L_i/n^2$ for a sufficiently large n, and thus all edges in a shortest path in G between p and q have length less than L_i/n^2. Recall that L_i/n^2 is the length of a shortest edge of G among all edges of length larger than L_{i-1}. Thus by construction of $\mathcal{L}$, there is no edge of G whose length lies in $(L_{i-1}, L_i/n^2)$. Therefore, all edges in a shortest path in G between p and q have length at most L_{i-1}, and thus $|pq| \leq d_G(p,q) \leq nL_{i-1}$. This contradicts $|pq| \in (nL_{i-1}, t^3L_i/n^3)$, and therefore, $|pq|$ lies on one of the five (closed) intervals excluding $(nL_{i-1}, t^3L_i/n^3)$. By construction, if $|pq|$ lies in the kth interval among the five intervals, then it lies in $\bar{\mathcal{L}}_{6-k}$. ◀

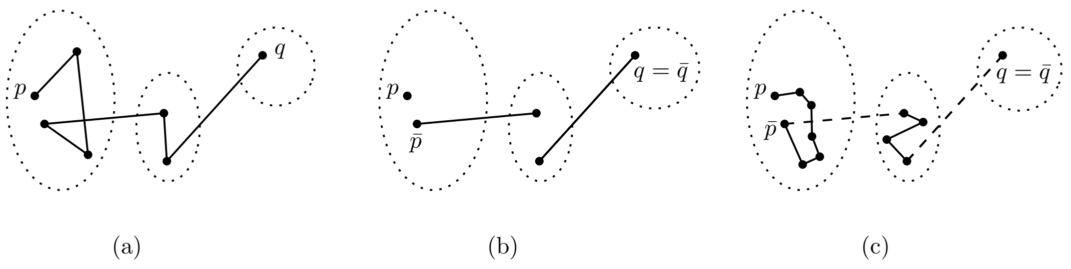

Figure 2 (a) A shortest path π_G in G between p and q. The interior of each dashed curve is a connected component of G_{i-2}. (b) Two bridges. The points $\bar{p}$ and $\bar{q}$ are the fist points in π_G incident to the bridges from p and q, respectively. (c) The bridges, and edges of G_{i-2} from an ϵ-approximate path in G between p and q.

3 Finding the Index i with $|pq| \in [L_{i-1}, L_i)$

Let $\mathcal{L} = \langle L_1, \dots, L_r \rangle$ be the sequence of values defined in Section 2. The following two lemmas give an algorithm for computing i with $|pq| \in [L_{i-1}, L_i)$ in constant time for any two query points p and q.

▶ **Lemma 3.** *If $L_{i-1} \le |pq| < L_i$, the length of a longest edge of the path in M between p and q lies in $[L_{i-2}, L_{i+1})$, where M denotes a minimum spanning tree of G.*

Proof. Let p and q be two vertices of V with $L_{i-1} \le |pq| < L_i$ for an integer i. Since G is a t-spanner, $d_G(p,q) \le t|pq| < tL_i$. Therefore, every edge in a shortest path in G between p and q has length at most $tL_i < L_{i+1}$. By the cycle property of the minimum spanning tree, all edges of the path $\pi_M(p,q)$ in M between p and q have length less than L_{i+1}.

Now we show that a longest edge of $\pi_M(p,q)$ has length at least L_{i-2}. Assume to the contrary that all edges of $\pi_M(p,q)$ have length less than L_{i-2}. Since the number of vertices of $\pi_M(p,q)$ is at most n, the length of $\pi_M(p,q)$ is less than $nL_{i-2} < L_{i-1}$. Note that $d_G(p,q)$ is at most the length of $\pi_M(p,q)$ because M is a subgraph of G. Therefore, we have $d_G(p,q) < nL_{i-2} < L_{i-1}$, which contradicts that $L_{i-1} \le |pq| \le d_G(p,q)$. ◀

▶ **Lemma 4.** *After an $O(n \log n)$-time preprocessing, we can compute the index i with $L_{i-1} \le |pq| < L_i$ in constant time for any two points of V.*

Proof. We sort all edges of G in increasing order with respect to the edge weights, and compute $\mathcal{L}$ in $O(n \log n)$ time. Also, we compute a minimum spanning tree M of G in $O(n \log n)$ time. Recall that G has $O(n)$ edges.

We construct a binary tree T such that each non-leaf node is associated with an edge of M, and each leaf node is associated with a vertex of M as follows. If M consists of a single vertex, T consists of a single node associated with the vertex of M. Otherwise, we create a new node which is the root of T, and associate the root with a longest edge of M. The subgraph of M obtained by removing the longest edge of M consists of two connected components. Then we define a binary tree for each connected component recursively, and merge the two binary trees by making their roots the children of the root of T. For a vertex of T associated with an edge e of M, we store the index $i(e)$ such that the length of e lies in $[L_i, L_{i+1})$. We can construct T and all indices $i(\cdot)$ in $O(n \log n)$ time in total.

Then for two points p and q, a longest edge of the path in M between p and q is stored in the lowest common ancestor of the leaf nodes of M associated with p and q, respectively, by construction. To answer this query, we construct the data structure in [17] so that the lowest common ancestor of any two nodes of T can be computed in constant time.

In this way, we can compute $i(e)$ in constant time, where e is a longest edge of the path in M between p and q. By Lemma 3, if $L_{i-1} \leq |pq| < L_i$ for some index i, the length of e lies in $[L_{i-2}, L_{i+1})$. That is, $i(e) \in [i-2, i)$. If $L_{i-1} \leq |pq| < L_i$, i is either $i(e)+2$ or $i(e)+1$, and we can check if a given integer is i or not in constant time. Therefore, we can compute i with $L_{i-1} \leq |pq| < L_i$ in constant time. ◀

Therefore, we can obtain the index i with $L_{i-1} \leq |pq| < L_i$ in constant time, and then we can find the sequence $\mathcal{L}_j$ with $|pq| \in \bar{\mathcal{L}}_j$ in constant time. With a slight abuse of notation, we let $\mathcal{L} = \langle L_1, L_2, \ldots, L_r \rangle$ be $\mathcal{L}_j$. Note that $L_i \geq n^2 L_{i-1}$ for every integer i with $2 \leq i \leq r$, and the length of every edge of G is in $[L_1, L_{r-1})$. In the following sections, we present a data structure of size $O(n \log n)$ so that an ϵ-approximate path between p and q can be computed in constant time per edge for any two vertices p and q in V with $|pq| \in \bar{\mathcal{L}}$.

4 Long and Short Edges of a Shortest Path in G

We decompose E into sets $E_1, E_2, \ldots, E_r$ such that E_i is the set of edges of G whose lengths lie in $[L_{i-1}, L_i)$, where $L_0 = 0$. The data structure for $\mathcal{L}$ also consists of several substructures. Each substructure is constructed for each subset E_i. It handles query points p and q with $|pq| \in [L_i/n, L_i/t]$. A shortest path π_G in G between p and q can be partitioned into several pieces so that each piece is a maximal subpath of π_G contained in the same connected component of the subgraph of G induced by the edges in $E_1 \cup E_2 \cup \ldots \cup E_i$. We call the edges of π_G connecting two pieces of π_G *bridges*. See Figure 2(a,b). Our strategy is to compute the bridges of an approximate path first, and then connect each of them by a (not necessarily shortest) path of G. See Figure 2(c). Since the total length of the bridges is much larger than the length of the other part of π_G, it suffices to connect two bridges by any path of G.

To use this observation, we construct a graph S_i of size $O(|E_{i-1} \cup E_i|)$ for each i such that S_i contains all edges of $E_{i-1} \cup E_i$ and some short edges connecting the edges of $E_{i-1} \cup E_i$. The number of short edges is $O(|E_{i-1} \cup E_i|)$, and short edges are not necessarily contained in E. We can modify the data structure of [12] to compute *long edges* of an approximate path in S_i in the case that the Euclidean distance between the query points is larger than the length of a longest edge of S_i. Using this data structure, we can compute the bridges of π_G for query points p and q. Then we compute the edges of E connecting the bridges. For an illustration, see Figure 1.

In the following, for a fixed index i, we show how to construct a graph S_i, which is a $2L_i$-partial $t(1+\epsilon)$-spanner for V. Using S_i, we show how to obtain an ϵ-approximate path of G between two query points. We use G_i to denote the subgraph of G with the vertex set V and the edge set $E_1 \cup E_2 \cup \ldots \cup E_i$. Notice that G_i is an L_i-partial t-spanner for V because G is a t-spanner for V due to Lemma 5. Then our goal in this section can be restated as follows: construct a data structure for G_i of size $O(|E_i \cup E_{i-1}|)$ so that given two points p and q, an ϵ-approximate path in G_i can be found efficiently assuming that $L_i/n \leq |pq| < L_i/t$.

▶ **Lemma 5.** *We have $d_G(p,q) = d_{G_i}(p,q)$ for any two points p and q with $|pq| < L_i/t$.*

Proof. Since $d_G(p,q) \leq t|pq| < L_i$, the length of any edge of a shortest path in G between p and q is less than L_i. Therefore, every edge of the shortest path is contained in G_i, and thus $d_G(p,q) = d_{G_i}(p,q)$. ◀

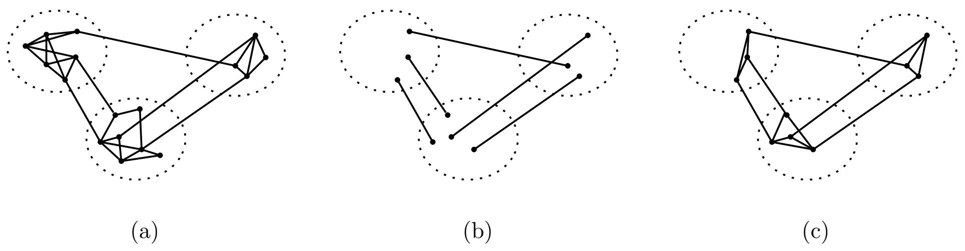

Figure 3 (a) The vertices lying in the interior of each dashed circle form a connected component of G_{i-2}. (b) The five bridges, and ten bridge points. (c) The line segments from a partial spanner S_i. An edge of S_i is either a bridge or an edge of a Euclidean $(1+\epsilon)$-spanner for a connected component of G_{i-2}.

4.1 Construction of S_i

We say an edge is *long* if its length is at least L_{i-2}, and an edge (which is not necessarily in G_i) is *short*, otherwise. Then we have the following lemma.

▶ **Lemma 6.** *A shortest path in G_i between two points contains a long edge if their Euclidean distance is at least L_i/n.*

Proof. Let p and q be two vertices of G_i with $|pq| > L_i/n$. Suppose that all edges of the shortest path are short, that is, their lengths are less than L_{i-2}. Then $d_{G_i}(p,q) < nL_{i-2} \leq L_i/n^3 < |pq|$. This contradicts that $|pq| \leq d_{G_i}(p,q)$ for any two points p and q in V. ◀

We define a graph $S_i = (V_i, F_i)$ as follows. A long edge of G_i is called a *bridge* if their endpoints lie on different connected components of G_{i-2}. In this case, their endpoints are called *bridge points*. Let B_i be the set of bridges, and V_i denote the set of bridge points. Note that $B_i \subseteq E_{i-1} \cup E_i$ and $|V_i| = O(|E_{i-1} \cup E_i|)$. We define the vertex set of S_i as V_i. See Figure 3(a,b).

The edge set F_i of S_i is defined as follows. For a connected component γ of G_{i-2}, let $V_i(\gamma)$ be the set of points of V_i which are contained in γ. We construct a Euclidean $(1+\epsilon)$-spanner for $V_i(\gamma)$ of size $O(|V_i(\gamma)|)$ in $O(|V_i(\gamma)| \log |V_i(\gamma)|)$ time using the algorithm in [8]. Then we let F_i be the set of all edges of B_i and all edges of the Euclidean spanners of length at most $2L_i$ for all connected components of G_{i-2}. See Figure 3(c).

▶ **Lemma 7.** *The complexity of S_i is $O(|E_i \cup E_{i-1}|)$.*

Proof. Since all bridges for G_{i-2} are contained in $E_i \cup E_{i-1}$, the number of vertices of V_i is $O(|E_i \cup E_{i-1}|)$. An edge of S_i is either a bridge for G_{i-2} or an edge of the Euclidean $(1+\epsilon)$-spanner for $V_i(\gamma)$ for a connected component γ of G_{i-2}. Since the sum of $|V_i(\gamma)|$ for all connected components γ is $O(|V_i|) = O(|E_i \cup E_{i-1|}|)$, the number of edges of S_i is also $O(|E_i \cup E_{i-1}|)$. ◀

4.2 Properties of S_i

For any two query points p and q with $|pq| \in [L_i/n, L_i/t]$, all bridges in an approximate path in G between p and q are contained in the edge set of S_i. We compute the bridges of an ϵ-approximate path in S_i between two vertices, and then compute several paths in G_{i-2}. The concatenation of those bridges and paths form an ϵ-approximate path between p and q.

In this subsection, we show that S_i is a $2L_i$-partial $t(1+\epsilon)$-spanner for V_i. This property is used for designing a data structure for computing the bridges of an ϵ-approximate path in S_i between two vertices, which is described in Section 6.3.

▶ **Lemma 8.** *S_i is a $2L_i$-partial $t(1+\epsilon)$-spanner for V_i.*

Proof. The lemma can be restated as follows by definition: For any two vertices u and v in V_i with $|uv| \le 2L_i$, we have $d_{S_i}(u,v) \le t(1+\epsilon)|uv|$.

Consider two vertices u and v in V_i with $|uv| \le 2L_i$ which are contained in the same connected component γ of G_{i-2}. Recall that the edges of a Euclidean $(1+\epsilon)$-spanner for $V_i(\gamma)$ of length at most $2L_i$ are contained in the edge set of S_i. Therefore, for any two vertices u and v in $V_i(\gamma)$ with $|uv| \le 2L_i$, their distance in S_i is at most $(1+\epsilon)|uv| \le t(1+\epsilon)|uv|$ since $t > 1$. Therefore, in the following, we consider two points u and v in V_i with $|uv| \le 2L_i$ which are contained in different connected components of G_{i-2}.

Consider a shortest path π in G between u and v. Since u and v are not connected in G_{i-2}, π contains at least one bridge in B_i. Consider all bridges of B_i contained in π. Let $\langle w_1, \ldots, w_k \rangle$ be the sequence of the endpoints of the bridges sorted along π from u to v. For each j, either $w_j w_{j+1}$ is a bridge of B_i, or w_j and w_{j+1} are connected in G_{i-2}. Also, let $w_0 = u$ and $w_{k+1} = v$.

We construct a path in S_i between u and v as follows. It consists of $k+1$ subpaths: $\pi_0, \pi_1, \ldots, \pi_k$. Let j be an index j with $0 \le j \le k$. If $w_j w_{j+1}$ is a bridge in B_i, we let π_j be the bridge. Otherwise, we let π_j be a shortest path between w_j and w_{j+1} in the Euclidean spanner for $V_i(\gamma)$, where γ is the connected component of G_{i-2} containing both w_j and w_{j+1}. Then we let π_S be the concatenation of $\pi_0, \ldots, \pi_k$.

Clearly, π_S is a path in S_i connecting u and v. Also, $d_{S_i}(u,v)$ is at most the length of π_S, which is the sum of the lengths of $\pi_0, \ldots, \pi_{k+1}$. If π_j is a bridge, its length is $|w_j w_{j+1}|$. Otherwise, its length is at most $(1+\epsilon)|w_j w_{j+1}|$ because it is a shortest path in a Euclidean spanner. Since w_j is a vertex of a shortest path in G for every integer j with $0 \le j \le k+1$, the distance in G between u and v is at least the sum of $|w_j w_{j+1}|$ for all $0 \le j \le k$. Also, since G is a Euclidean t-spanner for V, the distance in G between u and v is at most $t|uv|$. By combining all of them, we have

$$d_{S_i}(u,v) \le \sum_{j=1}^{k} d_{S_i}(w_j, w_{j+1}) \le \sum_{j=1}^{k} (1+\epsilon)|w_j w_{j+1}| \le (1+\epsilon) d_G(u,v) \le t(1+\epsilon)|uv|,$$

which completes the proof of the lemma. ◀

▶ **Corollary 9.** *For two points $u, v \in V_i$ with $|uv| \le L_i/t$, we have $d_{S_i}(u,v) \le (1+\epsilon) d_G(u,v)$.*

Also, by construction of S_i, we have the following lemma.

▶ **Lemma 10.** *A longest edge of S_i has length at most $2L_i$.*

Notice that, even though the distance in S_i between two points is within $(1+\epsilon)$ times their distance in G, a shortest path in S_i between the two points is not necessarily a path in G. This is because S_i contains edges which are not contained in E. Moreover, it is possible that p or q is not contained in V_i, where i is the index with $|pq| \in [L_i/n, L_i/t]$.

5 Specifying an Approximate Path

Consider any two points p and q in V with $|pq| \in [L_i/n, L_i/t]$. In this section, we specify an ϵ-approximate path we are going to compute among all ϵ-approximate paths between p and q in G. We call such a path an *approximate G-path*.

Let $\bar{p}$ and $\bar{q}$ be any two vertices of V_i which are contained in the same connected components of G_{i-2} as p and q, respectively. Notice that p and q are not necessarily contained in V_i. If $p \in V_i$ and $q \in V_i$, then $\bar{p}$ and $\bar{q}$ might be p and q, respectively. Let $\pi(p, \bar{p})$ be any path in G_{i-2} between p and $\bar{p}$. Similarly, let $\pi(q, \bar{q})$ be any path in G_{i-2} between q and $\bar{q}$. Now we connect $\bar{p}$ and $\bar{q}$ by a path in G_i as follows. Let $\pi_S(\bar{p}, \bar{q})$ be an ϵ-approximate path in S_i between $\bar{p}$ and $\bar{q}$. Note that $\pi_S(\bar{p}, \bar{q})$ might contain an edge not contained in E. Consider the bridges in B_i contained in $\pi_S(\bar{p}, \bar{q})$. Let $\langle w_1, \ldots, w_r \rangle$ be the sequence of the endpoints of the bridges sorted along $\pi_S(\bar{p}, \bar{q})$.

In the following, we construct a path $\pi_\epsilon(p, q)$ of G connecting $p, \bar{p}, w_1, w_2, \ldots, w_r, q, \bar{q}$ in order whose length is at most $(1 + 3\epsilon)d_G(p, q)$. For each integer $j = 1, \ldots, r$, if $w_j w_{j+1}$ is a bridge, it is an edge of G. In this case, we connect w_j and w_{j+1} by the bridge. If $w_j w_{j+1}$ is not a bridge, w_j and w_{j+1} are contained in the same connected component γ of G_{i-2}. In this case, we connect w_j and w_{j+1} by an arbitrary path in G_{i-2}. The concatenation of $\pi(p, \bar{p})$, paths (or bridges) connecting w_j and w_{j+1}, and $\pi(q, \bar{q})$ is called an ϵ-approximate G-path, denoted by $\pi_\epsilon(p, q)$. By construction, it is a path in G connecting $p, \bar{p}, w_1, w_2, \ldots, w_r, \bar{q}, q$ in order. Notice that all bridges of $\pi_\epsilon(p, q)$ are long, and all non-bridge edges are short.

▶ **Lemma 11.** *The length of $\pi_\epsilon(p, q)$ is at most $(1 + 3\epsilon)d_G(p, q)$.*

Proof. Recall that $L_i/n \leq |pq| \leq L_i/t$. Let $\langle w_1, \ldots, w_r \rangle$ be the sequence of the endpoints of the bridges in $\pi_S(\bar{p}, \bar{q})$ from $\bar{p}$ to $\bar{q}$. Let $w_0 = p$ and $w_{k+1} = q$. Consider an edge $w_j w_{j+1}$ with $0 \leq j \leq k$. If it is a bridge, it is contained in $\pi_\epsilon(p, q)$. In this case, the length of the part of $\pi_\epsilon(p, q)$ from w_j to w_{j+1} is $|w_j w_{j+1}|$. If it is not a bridge, w_j and w_{j+1} are contained in the same connected component of G_{i-2}. Since $L_i/L_{i-1} \geq n^2$ and the number of vertices of G_{i-2} is at most n, the length of a path connecting w_j and w_{j+1} in G_{i-2} is at most $L_{i-2}n \leq L_{i-1}/n$. Therefore, the length of the part of $\pi_\epsilon(p, q)$ from w_j to w_{j+1} is at most L_{i-1}/n.

Therefore, we have the following inequalities.

$$\begin{aligned}
w(\pi_\epsilon(p, q)) &\leq w(\pi_S(\bar{p}, \bar{q})) + L_{i-1} \\
&\leq (1 + \epsilon)d_G(\bar{p}, \bar{q}) + L_{i-1} \\
&\leq (1 + \epsilon)(d_G(p, q) + 2L_{i-1}/n) + L_{i-1} \\
&\leq (1 + \epsilon)d_G(p, q) + 2L_{i-1} \\
&\leq (1 + \epsilon)d_G(p, q) + 2\epsilon d_G(p, q),
\end{aligned}$$

where $w(\pi)$ denotes the length of a path π.

The first inequality holds because the total length of bridges of $\pi_\epsilon(p, q)$ is at most $|\pi_S(\bar{p}, \bar{q})|$, and the length of the remaining part is at most L_{i-1}. The second inequality holds by Corollary 9. The third inequality holds because a path in G_{i-2} connecting p and $\bar{p}$ (and q and $\bar{q}$) has length at most L_{i-1}/n. The last inequality holds due to $L_i/n \leq |pq| \leq d_G(p, q)$. Since ϵ is a constant, for a large n, it holds $L_{i-1} \leq L_i/n^2 \leq d_G(p, q)/n \leq \epsilon d_G(p, q)$. Therefore, the length of $\pi_\epsilon(p, q)$ is at most $(1 + 3\epsilon)d_G(p, q)$. ◀

The following lemma will be used for computing the long edges of $\pi_\epsilon(p, q)$ in Section 6.3.

▶ **Lemma 12.** *The Euclidean distance between $\bar{p}$ and $\bar{q}$ lies in $[L_i/(2n), 2L_i)$.*

Proof. Since p and $\bar{p}$ are connected in G_{i-2} by definition, the distance in G between them is at most nL_{i-2}, and thus $|p\bar{p}|$ is at most nL_{i-2}. Similarly, $|q\bar{q}|$ is at most nL_{i-2}. Recall that $|pq| \in [L_i/n, L_i/t]$.

By the triangle inequality, we have $|\bar{p}\bar{q}| \leq |\bar{p}p| + |pq| + |\bar{q}q| \leq L_i/t + 2nL_{i-2} \leq 2L_i$, which shows the upper bound. Similarly, by the triangle inequality, we have $|\bar{p}\bar{q}| \geq |pq| - |\bar{p}p| - |\bar{q}q| \geq L_i/n - 2nL_{i-2} \geq L_i/(2n)$, which shows the lower bound. ◀

6 Computing ϵ-Approximate G-Paths

In this section, we present a data structure and a query algorithm for computing an ϵ-approximate G-path for two query points p and q with $|pq| \in [L_i/n, L_i/t]$ assuming that we are given the index i. This data structure consists of three substructures: one for computing $\bar{p}$ and $\bar{q}$, one for computing the short edges of the path, and one for computing the long edges (bridges) of the path.

More specifically, we first construct a data structure in $O(n \log n)$ time, which is described in Section 6.1, so that given two points p and q, and an index i, we can compute $\bar{p}$ and $\bar{q}$ in constant time. Also, we construct a data structure in $O(n \log n)$ time so that for any two points p and q, and an index $i-1$, we can report the edges of an arbitrary path in G_{i-2} between p and q in constant time per edge. This data structure is described in Section 6.2. Note that these data structures do not depend on an index, and thus they can be used for all indices i. In addition to this, we construct a data structure on S_i which allows us to find all bridges in an ϵ-approximate path in S_i between $\bar{p}$ and $\bar{q}$. This data structure is described in Section 6.3, and it is a slight modification of the data structure of [12]. It can be constructed in $O(|E_i \cup E_{i-1}| \log n)$ time for each index i, and thus the total construction time for all indices is $O(n \log n)$ time.

▶ **Lemma 13.** *Assuming the three substructures mentioned above have been constructed, for any two query points p and q, we can compute $\pi_\epsilon(p,q)$ in constant time per edge.*

Proof. We first find $\bar{p}$ and $\bar{q}$ in constant time using the data structure described in Section 6.1. Then we compute all bridges of an ϵ-approximate path $\pi_S(p,q)$ in S_i in constant time per edge using the data structure described in Section 6.3. Notice that the bridges are contained in $\pi_\epsilon(p,q)$. Let $\langle w_1, \ldots, w_r \rangle$ be the sequence of the endpoints of the bridges sorted along $\pi_S(p,q)$. The bridge points are reported by the query algorithm described in Section 6.3 one by one from w_1 to w_r in constant time per edge. After w_{j+1} is reported by the algorithm, we compute the part of $\pi_\epsilon(p,q)$ lying between w_j and w_{j+1}. If $w_j w_{j+1}$ is not a bridge. we compute a path in G_{i-2} connecting w_j and w_{j+1} using the data structure described in Section 6.3 in constant time per edge. The path obtained by concatenating all paths we computed so far is an ϵ-approximate G-path. Therefore, we can compute the ϵ-approximate G-path in constant time per edge. ◀

6.1 Computing $\bar{p}$ for an Index i and Points p and q

In this section, we present a data structure for the following type of queries: Given two points p and q of V and an index i with $|pq| \in [L_i/n, L_i/t]$, we want to find vertices $\bar{p}$ (and $\bar{q}$) of V_i such that p and $\bar{p}$ (and q and $\bar{q}$) are contained in the same connected component of G_{i-2}. To do this, we use a minimum spanning tree M of G, which can be computed in $O(n \log n)$ time. Recall that we assume that the number of edges of G is $O(n)$. By properties of the minimum spanning tree, we have the following observation. We use $\pi_T(u,v)$ to denote the path in T between u and v for a tree T.

▶ **Observation 14.** *A longest edge in the path in M between two points p and q has length at most $\ell > 0$ if and only if there is a path in G between p and q whose longest edge has length at most ℓ.*

Data Structure. We sort all edges of G in increasing order with respect to their lengths and compute a minimum spanning tree M of G in $O(n \log n)$ time. Then we construct a tree T such that each non-leaf node stores an index, each leaf node of T stores a vertex of V, and each edge of T stores two vertices of V. If M consists of a single vertex, T consists of a single node which stores the vertex of M. Otherwise, consider all edges of M whose lengths lie in $[L_{s-1}, L_s)$, where s is an index such that the length of a longest edge of M lies in $[L_{s-1}, L_s)$. Imagine that we remove such edges from M. Then there are several connected components. For each connected component γ, we construct a tree $T(\gamma)$ recursively. Then we merge $T(\gamma)$'s for all connected components γ by making a new node and connecting it with the roots of $T(\gamma)$'s. Then we store the index s at the new node. Also, for the edge between the root of each connected component γ and the new node, we store two arbitrary vertices in V_i and V_{i+1} contained in γ. Notice that if a node v of T stores an index i, the vertices of M stored in the leaf nodes of the subtree rooted at v are connected in G_i.

Then for any two points p and q, the lowest common ancestor of the leaf nodes of T corresponding to p and q stores the index s such that the length of a longest edge of $\pi_M(p, q)$ lies in $[L_{s-1}, L_s)$ by construction. To use this property in our query algorithm, we construct the data structure in [2, 17] so that the lowest common ancestor (LCA) of any two nodes of T can be computed in constant time. Also, we construct a *level ancestor data structure* on T so that the children of the LCA of p and q contained in $\pi_T(p, q)$ for any two vertices p and q can be computed in constant time. A level ancestor data structure on an n-vertex tree has size of $O(n)$, and it can be constructed in $O(n \log n)$ time [3].

▶ **Lemma 15.** *For two points of V with $|pq| \in [L_i/n, L_i/t]$, the index stored in the lowest common ancestor of the leaf nodes corresponding to p and q in T is either i or $i-1$.*

Proof. We first claim that p and q are connected in G_i, but not in G_{i-2}. Since $d_G(p, q) \leq t|pq| \leq L_i$, a shortest path in G between p and q consists of edges of length at most L_i, and thus p and q are connected in G_i. Also, p and q are not connected in G_{i-2}. Otherwise, $d_G(p, q) \leq nL_{i-2} < L_{i-1}$, which contradicts $L_i/n \leq |pq| \leq d_G(p, q)$.

The lowest common ancestor of the leaf nodes corresponding to p and q stores the index i if p and q are not connected in G_{i-1}, or stores the index $i-1$ if p and q are connected in G_{i-1}. This completes the proof of the lemma. ◀

Query Algorithm. To compute $\bar{p}$, we first find the lowest common ancestor v of the leaf nodes for p and q in T in constant time, and compute the child v' of v in $\pi_T(v, p)$, and the child v'' of v' in $\pi_T(v, p)$ in constant time using the level ancestor data structure. By Lemma 15, v stores i or $i-1$. If v stores $i-1$, by construction, the edge vv' of T stores a vertex in V_i which is connected by p in G_{i-2}. If v stores i, the edge $v'v''$ of T stores a vertex in V_i connected by p in G_{i-2}. Therefore, in any case, we can compute $\bar{p}$ in constant time. Similarly, we can compute $\bar{q}$ in constant time.

▶ **Lemma 16.** *Given a sequence $\mathcal{L} = \langle L_1, L_2, \ldots, L_r \rangle$ of values in $\mathbb{R}$ such that $L_i \geq n^2 L_{i-1}$ for every integer i with $2 \leq i \leq r$, and the length of every edge of G is in $[L_1, L_r)$, we can construct a data structure of size $O(n \log n)$ in $O(n \log n)$ time so that given any two query points p and q of V with $|pq| \in [L_i/n, L_i/t]$, we can find vertices $\bar{p}$ (and $\bar{q}$) of V_i connected by p (and q) in G_{i-2} in constant time.*

6.2 Computing Short Edges

In this section, we show how to construct a data structure so that for any two query points p and q, and an index i, a path in G_{i-2} between p and q can be found in constant time per edge. Here, a path is not necessarily a shortest path.

We use a minimum spanning tree M of G. Since M is a tree, we can compute the path in M connecting any two vertices p and q in constant time per edge. To do this, we choose an arbitrary vertex of M as the root of M, and construct the data structure in [17] so that the lowest common ancestor of any two vertices of M can be computed in constant time. Using this data structure, we compute the LCA u of p and q. Then we traverse M from p towards the root until we encounter u. Then from u, we traverse the path between u and p in constant time per edge. Due to the level ancestor data structure, we can compute the child of any node w in the path between w and q in constant time, and thus each edge of the path can be computed in constant time. In this way, we can compute the path in M from p to q in constant time per edge.

By Observation 14, the returned path is a path in G_{i-2} assuming that such a path exists.

▶ **Lemma 17.** *Given a t-spanner G for a point set V in d-dimensional Euclidean space, we can construct a data structure of size $O(n \log n)$ in $O(n \log n)$ time so that given any two query points p and q of V, and an index i, we can compute a path in G_{i-2} between p and q in constant time per edge, assuming such a path exists.*

6.3 Computing Long Edges

We construct a data structure on S_i so that for any two vertices u and v of V_i, the bridges of an ϵ-approximate path in S_i between u and v can be computed in constant time per bridge. We obtain this data structure by slightly modifying the data structure of [12], which allows us to compute an approximate distance in a $2L_i$-partial t-spanner in the case that the Euclidean distance between two query points lies in $[L_i/(2n), 2L_i)$. In our case, the query points for this data structure are $\bar{p}$ and $\bar{q}$, and their Euclidean distance lies in $[L_i/(2n), 2L_i)$ by Lemma 12. We give a brief sketch of the data structure. Details can be found in Section 6.3 in the appendices.

Brief sketch of [12]. A key idea is to partition $[L_i/(2n), 2L_i)$ into $O(\log n)$ subintervals each of the same length. Then a value in a subinterval is larger than the square of any other values in the subinterval. For each interval I, they compute a data structure on S_i such that, for any two vertices u and v of S_i with $|uv| \in tI$, the ϵ-approximate distance between them can be computed in constant time. Since for any two vertices u and v with $|uv| \in [L_i/(2n), 2L_i)$, there is a subinterval I such that $|uv| \in tI$. Therefore, they can use the sequence of $O(\log n)$ data structures to answer approximate distance queries.

The data structure constructed for each interval I is a *cluster graph*, which was introduced by Das et al. [7]. The vertex set of the cluster graph is the vertex set of S_i, and an edge of the cluster graph is called either an *inter-cluster* edge, or an *intra-cluster* edge. If we remove all inter-cluster edges from the cluster graph, it becomes a forest. A inter-cluster edge connects the roots of two trees of the forest. Das et al. [7] showed that if $uv \in tI$, the shortest path in the cluster graph between u and v consists of $O(1)$ inter-cluster edges and the shortest path between u (and v) and the root of the tree containing u (and v). Moreover, a vertex is incident to $O(1)$ inter-cluster edges. Also, they showed that the distance in the cluster graph between two points is an ϵ-approximate path between them in S_i. Using this property, one can compute an ϵ-approximate path in constant time.

Modification. It can be easily modified to support shortest-path queries. As in [12], we compute all inter-cluster edges in the shortest path between u and v in constant time. In the preprocessing phase, we compute for each inter-cluster edge, we store a path in S_i connecting the endpoints. Since an intra-cluster edge is an edge of S_i, we can compute an ϵ-approximate path between any two points in constant time.

Instead of computing all edges of the path, our goal is to compute the bridges in the path in constant time per bridge. To do this, for each intra-cluster edge, we store the pointer pointing to the closest bridge from the edge in the path between the node and the root. Also, instead of storing all edges of a path for each inter-cluster edge, we store the bridges in the path only. Then we can find all bridges in the path between v (and u) and the root of the tree in constant time per bridge.

References

1 Rachit Agarwal and P. Brighten Godfrey. Distance oracles for stretch less than 2. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, page 526–538, USA, 2013. Society for Industrial and Applied Mathematics.

2 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

3 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–21, 2004. Latin American Theoretical Informatics.

4 Timothy M. Chan and Dimitrios Skrepetos. Approximate Shortest Paths and Distance Oracles in Weighted Unit-Disk Graphs. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SoCG.2018.24.

5 Timothy M. Chan and Dimitrios Skrepetos. Faster approximate diameter and distance oracles in planar graphs. *Algorithmica*, 81(8):3075–3098, August 2019. doi:10.1007/s00453-019-00570-z.

6 Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 138–151, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316316.

7 Gautam Das and Giri Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, pages 132–139, New York, NY, USA, 1994. ACM. doi:10.1145/177424.177579.

8 Michael Elkin and Shay Solomon. Optimal Euclidean spanners: Really short, thin, and lanky. *J. ACM*, 62(5), November 2015. doi:10.1145/2819008.

9 Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 483–492, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/780542.780613.

10 Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 515–529, 2018. doi:10.1137/1.9781611975031.34.

11 Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\epsilon)$-approximate distance oracle for planar graphs. *Theoretical Computer Science*, 761:78–88, 2019.

12 Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate distance oracles for geometric spanners. *ACM Trans. Algorithms*, 4(1):10:1–10:34, March 2008. doi:10.1145/1328911.1328921.

13 Joachim Gudmundsson, Giri Narasimhan, and Michiel Smid. Fast pruning of geometric spanners. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, pages 508–520, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

14 Joachim Gudmundsson, Giri Narasimhan, and Michiel Smid. *Applications of Geometric Spanner Networks*, pages 86–90. Springer New York, New York, NY, 2016. `doi:10.1007/978-1-4939-2864-4_15`.

15 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

16 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

17 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. `doi:10.1137/0213024`.

18 John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

19 Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 135–146, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

20 H. Le and S. Solomon. Truly optimal euclidean spanners. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1078–1100, 2019.

21 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. `doi:10.1017/CBO9780511546884`.

22 M. Patrascu and L. Roditty. Distance oracles beyond the thorup-zwick bound. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 815–823, 2010.

23 M. Patrascu, L. Roditty, and M. Thorup. A new infinity of distance oracles for sparse graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 738–747, 2012.

24 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup–zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014.

25 Jose L. Santos. Real-world applications of shortest path algorithms. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 13-14, 2006*, volume 74 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. DIMACS/AMS, 2006. `doi:10.1090/dimacs/074/01`.

26 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4), March 2014. `doi:10.1145/2530531`.

27 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, January 2005. `doi:10.1145/1044731.1044732`.