Approximate Distance Oracles for Geometric Spanners

JOACHIM GUDMUNDSSON
National ICT Australia Ltd

and
CHRISTOS LEVCOPOULOS
Lund University

and
GIRI NARASIMHAN
Florida International University

and
MICHIEL SMID
Carleton University

Given an arbitrary real constant $\varepsilon > 0$, and a geometric graph G in d-dimensional Euclidean space with n points, $O(n)$ edges, and constant dilation, our main result is a data structure that answers $(1+\varepsilon)$-approximate shortest path length queries in constant time. The data structure can be constructed in $O(n \log n)$ 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. Finally, we show how to extend the main result to answer $(1+\varepsilon)$-approximate shortest path length queries in constant time for geometric spanner graphs with $m = \omega(n)$ edges. The resulting data structure can be constructed in $O(m + n \log n)$ time using $O(n \log n)$ space.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Shortest paths, geometric graphs, approximation algorithm, computational geometry, spanners

1. INTRODUCTION

The *shortest-path* (SP) problem for weighted graphs 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

Authors' addresses: J. Gudmundsson, National ICT Australia Ltd, Australia. NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.
C. Levcopoulos, Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.
G. Narasimhan, School of Computer Science, Florida International University, Miami, FL 33199, USA.
M. Smid, School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. This author was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

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, in many applications, it is sufficient to solve the problem approximately. The approximation version of the shortest path problem has been studied extensively; for example, see Aingworth et al. [1999], Cohen [1998], Dor et al. [2000].

For the approximate query version on undirected weighted graphs with n vertices and m edges, Thorup and Zwick [2001] present an algorithm that 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. Since the query time is essentially bounded by a constant, Thorup and Zwick refer to their queries as approximate *distance oracles*. It is not an approximation scheme in the true sense because the value k needs to be a positive integer. For the unweighted case, Baswana and Sen [2004] solved the same problem by showing a new algorithm with a preprocessing time of $O(n^2 \log n)$, which is an improvement over the result of Thorup and Zwick if km is asymptotically greater than $n \log n$. Experimental studies on practical algorithms for shortest path problems were studied by Wagner and Willhalm [2003], Wagner et al. [2004] and Pyrga et al. [2004]. A related result on answering distance queries in weighted planar graphs is a result from Arikati et al. [1996], where they showed that given an n-vertex planar graph G with non-negative real edge weights and an arbitrary number $1 \leq \rho \leq \sqrt{n}$, after preprocessing with $O(n^2/\rho)$ time and space, a distance query between any two vertices can be answered in $O(\rho)$ time. For approximate distance queries in planar graphs, the best result can be found in Thorup [2004], where it was shown that given an n-vertex planar graph G with non-negative real edge weights and an arbitrary real number $\varepsilon > 0$, after preprocessing with near-linear time and space, a $(1+\varepsilon)$-approximate distance query can be answered in $O(1/\varepsilon)$ time.

We focus on the geometric version of the approximate distance query 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. We do not assume that the geometric graphs are planar. There are no known results for answering shortest path queries in geometric graphs that are better than what is known for weighted graphs (i.e., the non-geometric cases mentioned above). In particular, for approximation factors less than 3, only the trivial solutions were known; namely, either computing and storing a complete distance matrix, or leaving the graph untouched and answering each query with a single source shortest path computation from scratch.

In the geometric case, good surveys can be found in Mitchell [1997], Mitchell [1998], Suri [1997]. However, most of the work reported are on variants of the problem. The main variants considered include answering shortest path queries in polygonal domains, where the paths have to lie within a polygonal region, which may or may not include obstacles. Other variants include answering shortest path queries on polyhedral surfaces. All these variants have been considered in a number of papers, including the ones by Guibas and Hershberger [1989] (simple polygons), Chen [1995] (polygonal domains with polygonal obstacles), Mitchell [1996], Kapoor et al. [1997], Hershberger and Suri [1999], and Chen et al. [2001] (polygonal domains with polygonal obstacles). Work on the approximate version of these variants can be found in Clarkson [1987], Chen [1995], Arikati et al. [1996], Chen et al. [2000], Har-Peled [1997], Agarwal et al. [1997], and Agarwal et al. [2000].

Note that while the problem of finding shortest paths in polygonal domains can be modeled as a problem on geometric graphs, it is not clear whether the problem of finding shortest paths in geometric graphs can be reduced to a corresponding problem in polygonal domains. Thus the geometric variants mentioned above do not appear to be useful to solve the distance query problem in geometric graphs. More importantly, note that none of the existing algorithms for these geometric variants answer shortest path length queries (either approximate or exact) in **constant** time with subquadratic space and preprocessing time. In this paper we optimally solve the approximate distance oracle problem for geometric spanners.

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 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 $\varepsilon > 0$. Assuming that $m = O(n)$, our main result is an algorithm that preprocesses G in $O(n \log n)$ time and space, after which shortest path length queries can be answered in constant time.

We remark that many "naturally occurring" geometric graphs are t-spanners, 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 Das and Joseph [1989], Keil and Gutwin [1992]. As a more practical example, the section of the southern Scandinavian railroad network shown in Figure 1 of Narasimhan and Smid [2000] 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.

In a preliminary conference version (Gudmundsson et al. [2002a]), we showed the basic ideas of this algorithm as applied to point sets with the additional constraint that the interpoint distances be within a polynomial ratio of each other. In a follow-up conference version (Gudmundsson et al. [2002b]), we showed how to eliminate the above constraint. This current paper is an archival version that combines both those results.

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

THEOREM 1.1. *Let $t > 1$ and $\varepsilon > 0$ be real constants. Let V be a set of n points in $\mathbb{R}^d$, and let $G = (V, E)$ be a t-spanner for V with $O(n)$ edges. The graph G can be preprocessed into a data structure of size $O(n \log n)$ in time $O(n \log n)$, such that for any pair of 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 $O(1)$ time. Note that all the big-Oh notations hide constants that depend on d, t and ε.*

The main data structure used is a series of "cluster graphs" of increasing coarseness each of which helps answer queries for pairs of points with interpoint distances of different scales. The idea of using cluster graphs to speed up geometric algorithms was first introduced by Das and Narasimhan [1997] and later used by Gudmundsson et al. [2002] to design an efficient algorithm to compute $(1 + \varepsilon)$-spanners. More recently, similar ideas have been used by Gao et al. [2003].

Another important contribution of this paper is a technical tool to "bucket" edge lengths in constant time, without using the floor function. This tool is used in our algorithm to quickly determine the approximate distance between points in Euclidean space, which in turn is used 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. The heart of this bucketing tool is the use of a constant-time lowest common ancestor (LCA) computation. The idea of using LCA to estimate distances between points has perhaps been used in some indirect form in the work of Bartal [1996] on probabilistic embeddings of metric spaces in trees. Bartal defined the concept of a k-HST (hierarchically well-separated tree), which is a probabilistic hierarchical clustering produced with the property that vertices that are closer to each other have a higher probability of having a LCA that is closer. Other related data structures include Arora's randomly translated quadtrees, see Arora [1997], Arora [1998], and the quadtree-based data structure used by Chan [1998] (also see the survey Indyk [2001]), both of which had properties similar to that of k-HSTs, but in the geometric context. We provide the simplest deterministic method to use LCA for the purpose of distance estimation or rounding in the geometric context.

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, a concept that was introduced in Overmars and van der Stappen [1996] and de Berg et al. [2002]. It is clear that *skinny* obstacles force long paths around them and it was argued in these two papers that for most practical situations, it is enough to be able to handle *fat* obstacles. 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.

Finally, using a recent result of Gudmundsson et al. [2005], we show in Corollary 6.1 that the result of Theorem 1.1 can be extended to geometric spanner graphs with superlinear edges.

Our model of computation is the traditional algebraic computation tree model with the added power of indirect addressing. In particular, our algorithms do not use the non-algebraic floor function as a unit-time operation. (Recall, that by adding the floor function, we obtain a strictly stronger model of computation. For example, the MAXGAP-problem can be solved in $O(n)$ time if the floor function is available, whereas this problem has an $\Omega(n \log n)$ lower bound in the algebraic model; see Gonzalez [1975], Lee and Wu [1986], and Preparata and Shamos [1988].)

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. Given a constant $t > 1$, G is a t-spanner for V if $\delta_G(p, q) \leqslant t|pq|$ for any two points p and q of V. If P is a path in G between p and q having length Δ with $\delta_G(p, q) \leqslant \Delta \leqslant (1 + \varepsilon)\delta_G(p, q)$, then P is said to be a $(1 + \varepsilon)$-approximate shortest path for p and q. We say that a subgraph G' of G is a t'-spanner of G, if $\delta_{G'}(p, q) \leqslant t' \cdot \delta_G(p, q)$ for any two points p and q of V. Finally, if $L > 0$ is a real number, 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 $\delta_G(p, q) \leqslant t \cdot |pq|$.

In Section 2, we start with an easier version of the problem. We present a data structure that answers approximate distance queries in a spanner G for pairs of points that are "sufficiently separated" from each other. In Section 3, the result is extended to the case when there is no restriction on how close the query points p and q are. 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. In Section 5, we present a tool for bucketing edge lengths without using the floor function, thus completing the proof of Theorem 1.1. This tool is used repeatedly in Sections 2 and 3. Finally, in Section 6, we extend Theorem 1.1 to geometric spanner graphs of arbitrary size.

2. APPROXIMATE DISTANCE QUERIES FOR "SUFFICIENTLY SEPARATED" POINTS

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 $t > 1$. We assume that $|E| = O(n)$. Let D be the length of the longest edge in G, We fix a positive constant ε. We also fix a real number $C > 2$, which may depend on n. In this section, we solve the $(1 + \varepsilon)$-approximate distance oracle problem for pairs of points $p, q \in V$ such that $|pq| > D/C$.

2.1 Preprocessing for approximate distance queries

The preprocessing of graph G, which is performed before approximate shortest path queries can be answered, is described below.

Preprocessing algorithm: *Run the* ImprovedGreedy *algorithm (see Gudmundsson et al. [2002]) with modifications as discussed below. The* ImprovedGreedy *algorithm maintains a graph G', which is a partial $(1+\varepsilon)$-spanner of G. During the process it creates a sequence of cluster graphs of the partial spanner, one in each iteration. These cluster graphs constitute the data structure for answering shortest path length queries.*

Below we provide details on how the ImprovedGreedy algorithm from Gudmundsson et al. [2002] is modified for the preprocessing algorithm. In the process, we also summarize the cluster graph data structure and its properties. The cluster graph data structure used here was first introduced by Das and Narasimhan [1997] and later used in Gudmundsson et al. [2002] to design an efficient, cluster-based algorithm to compute $(1 + \varepsilon)$-spanners.

2.1.1 *Generating the cluster graphs.* First, as described in Gudmundsson et al. [2002], the ImprovedGreedy algorithm takes as input a set of points in $\mathbb{R}^d$ and computes a t-spanner of the set of points. In order to achieve this, Step 1 of algorithm ImprovedGreedy (see Figure 2 of Gudmundsson et al. [2002]) constructs a linear-sized $\sqrt{t/t'}$-spanner graph, of which a $\sqrt{tt'}$-spanner is output by the rest of its computation.

In contrast, the input to the preprocessing algorithm is a linear-sized t-spanner graph of the set of points and we intend to end up with a $(1+\varepsilon)$-spanner of this graph (besides a sequence of cluster graphs). Thus, we

can skip Step 1 of ImprovedGreedy and proceed directly to Step 2 of the ImprovedGreedy algorithm by assuming that the value of $\sqrt{tt'}$ used in that algorithm is replaced in the preprocessing algorithm by $1+\varepsilon$.

Second, we assume that the length D of the longest edge in G is equal to C. If this is not the case, then we scale the coordinates of the points of V so that this is true. This makes some of our later discussions more convenient. This implies that for the query points p and q, we must have $|pq| > D/C = 1$.

Continuing with the steps of the ImprovedGreedy algorithm, a graph G' is initialized to $(V, \emptyset)$, and the edges of E are sorted according to their lengths. Then, the algorithm, acting in a greedy fashion, examines the edges of E one by one, in increasing order of length. When it 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 and, therefore, a $t(1+\varepsilon)$-spanner of the set V of points. In fact, it was also shown in Gudmundsson et al. [2002] that after processing all edges of length at most L, the graph G' at that time is a (L/t)-partial $t(1+\varepsilon)$-spanner of the point set, which means that it contains a $t(1+\varepsilon)$-spanner path between all pairs of points whose interpoint distance is at most L/t.

In order to speed up the above computation, the edges are batched together and then processed in phases. We partition the edge set E into subsets

$$E_0 := \{(p, q) \in E : |pq| \leqslant 1\}, \qquad \text{and}$$

$$E_i := \{(p, q) \in E : 2^{i-1} < |pq| \leqslant 2^i\}, \quad 1 \leqslant i \leqslant 1 + \lfloor \log C \rfloor.$$

The graph G' is initialized to (V, E_0), and the remaining subsets E_i, $i \geqslant 1$, are processed in $1 + \lfloor \log C \rfloor$ phases. At the beginning of phase i in which E_i is processed, the algorithm constructs a *cluster graph* H of the current partial spanner 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 2^i, 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'. Gudmundsson *et al.* proved that the final graph G' (i.e., after all subsets E_i, $i \geqslant 1$, have been processed) is a $(1+\varepsilon)$-spanner of G. They also showed that for each i, $1 \leqslant i \leqslant 1 + \lfloor \log C \rfloor$, the algorithm spends $O(n + |E_i|)$ time to process the edge set E_i. Hence the overall algorithm runs in

$$O\left(n \log n + \sum_{i \geqslant 1} (n + |E_i|)\right) = O\left(n \log n + n \log C + |E|\right) = O(n \log n + n \log C) \tag{1}$$

time. We now present some properties of the cluster graph in more detail.

2.1.2 *The cluster graph and its properties.* The cluster graph (of a given input graph with respect to a certain radius W) is a critical data structure used in the previous work and is also crucial for the algorithm in this paper. We start with a simple description of this structure. The detailed definition can be found in Section 3 of Das and Narasimhan [1997]. The basic idea is to construct a different graph (the cluster graph) whose vertex set is the same as the input graph, but some of the vertices are marked as *cluster centers*. A cluster center forms a *cluster* with all vertices that are within distance W from it in the input graph. The clusters *cover* the input graph, but no cluster center is in more than one cluster. The reader is encouraged to refer to further details in Das and Narasimhan [1997] on the edges in the cluster graph and their weights. A key property shown in Das and Narasimhan [1997] was that if the input graph is a partial spanner graph (such as the one maintained in the algorithm), then the cluster graph has a linear number of edges and its degree is bounded by a constant.

The following lemmas are important properties of the spanner graph and cluster graph maintained in each of the phases. Consider an arbitrary phase j of the modified ImprovedGreedy algorithm used in the preprocessing algorithm.

LEMMA 2.1. *After processing all edges in phase j, the spanner graph G' maintained at that time is a $(1+\varepsilon)$-spanner of the original graph G, for all pairs of vertices x and y, such that $(x,y) \in E_k, k \leqslant j$.*

PROOF. For every edge $(x,y) \in E_j$, the modified IMPROVEDGREEDY algorithm used in the preprocessing algorithm makes a call to algorithm SHORTPATH$(H, x, y, (1+\varepsilon)|xy|)$ (see Figure 3 in Das and Narasimhan [1997], where the term $\sqrt{tt'}$ appearing in the IMPROVEDGREEDY algorithm was replaced in our modified algorithm by $1+\varepsilon$). This call, as described in detail in Das and Narasimhan [1997], initiates a naive Dijkstra's shortest path algorithm from x in the cluster graph H (maintained in that phase) until all vertices within distance $(1+\varepsilon)|xy|$ are reached. If vertex y is not one of the vertices reached, algorithm SHORTPATH returns false and edge (x,y) is added to G' (the spanner graph maintained in that phase). Else, it reports true, and edge (x,y) is not added to G'. If the edge (x,y) is added to G', then clearly $\delta_{G'}(x,y) = \delta_G(x,y) = |xy|$. If it is not added, then because of the test in algorithm SHORTPATH, we know that $\delta_H(x,y) \leqslant (1+\varepsilon)|xy| = (1+\varepsilon)\delta_G(x,y)$. Since $\delta_{G'}(x,y) \leqslant \delta_H(x,y)$ (by Lemma 3 of Das and Narasimhan [1997]), we have shown that $\delta_{G'}(x,y) \leq (1+\varepsilon)\delta_G(x,y)$. Consequently, after phase j, the graph G' is a $(1+\varepsilon)$-spanner for all pairs of points corresponding to edges in E_j. In fact, it is a $(1+\varepsilon)$-spanner for all pairs of points corresponding to edges in $E_k, k \leqslant j$. □

We define the constant a to be

$$a := \max\{\lceil \log t \rceil, 1\}.$$

Let p and q be two points of V such that $|pq| > 1$ (i.e., they are sufficiently separated), and let $i = \lceil \log |pq| \rceil$. Consider the graph G' and the corresponding cluster graph H computed in the preprocessing algorithm at the moment when the greedy algorithm has just processed all edges of E_{i+a}. Thus graphs G' and H are considered after all edges of weight at most 2^{i+a} have been considered.

LEMMA 2.2.

$$\delta_G(p,q) \leqslant \delta_{G'}(p,q) \leqslant (1+\varepsilon) \cdot \delta_G(p,q).$$

PROOF. Graph G' is a subgraph of G and hence the first inequality.

Since G is a t-spanner of V, we know that there exists a path in G between p and q, whose length is at most $t|pq|$. In other words, $\delta_G(p,q) \leqslant t|pq| \leqslant t2^i \leqslant 2^{i+a}$, implying that each edge on the shortest path in G between p and q has length at most 2^{i+a}. Thus each edge on the shortest path must have been considered by phase $i+a$ of our modified IMPROVEDGREEDY algorithm. By Lemma 2.1, for each edge (x,y) on the shortest path, we have $\delta_{G'}(x,y) \leqslant (1+\varepsilon)|xy|$, thus proving that G' contains a $(1+\varepsilon)$-approximation to the shortest path between p and q in G. □

LEMMA 2.3.

$$\delta_G(p,q) \leqslant \delta_H(p,q) \leqslant (1+\varepsilon)^2 \cdot \delta_G(p,q)$$

PROOF. Lemma 2.2 shows that $\delta_G(p,q) \leqslant \delta_{G'}(p,q)$. Lemma 3 of Das and Narasimhan [1997] proves that $\delta_{G'}(p,q) \leqslant \delta_H(p,q)$. Putting the two statements together gives us the first inequality.

To prove the second inequality, choose a positive constant α such that $\alpha < 1/4$ and

$$\frac{1+6\alpha}{1-2\alpha} \leqslant 1+\varepsilon.$$

Then we know that

$$\delta_{G'}(p,q) \geqslant |pq| > 2^{i-1} > (1-2\alpha)2^{i-1}.$$

By replacing the symbol δ in Lemma 4 of Das and Narasimhan [1997] by α, we know that

$$\delta_H(p,q) \leqslant \left(\frac{1+6\alpha}{1-2\alpha}\right) \cdot \delta_{G'}(p,q) \leqslant \left(\frac{1+6\alpha}{1-2\alpha}\right)(1+\varepsilon) \cdot \delta_G(p,q) \leqslant (1+\varepsilon)^2 \delta_G(p,q),$$

where the second inequality follows from Lemma 2.2. □

2.1.3 *Complexity analysis.* As shown in (1), the entire preprocessing can be performed in $O(n \log n + n \log C)$ time. Finally, by Lemma 7 of Das and Narasimhan [1997], each cluster graph has n vertices and $O(n)$ edges. Since we compute and save $O(\log C)$ cluster graphs, one for each phase, the space complexity of the preprocessing is $O(n \log C)$.

2.2 Answering approximate distance queries

The two lemmas above prove that distances between vertices in the graphs G' and H are approximately the same as that in G, for pairs of points at least distance one apart. (Recall that we defined a to be the larger of $\lceil \log t \rceil$ and 1.) The important point to note is that if the queried pairs of points have interpoint distances that would have been processed in phase i of Step 2 above, then the graphs G' and H are those that are computed by the algorithm a phases later (in phase $i+a$). This implies that to answer approximate shortest path length queries in G, it is sufficient to perform the search in graphs G' or H. All three graphs share the same vertex set V and are sparse (i.e., have $O(n)$ edges). What we will show next is that the cluster graph H has the appropriate properties making it the ideal candidate to be used for the required search. As before, we consider two points p and q in V with $|pq| > 1$, and the positive integer $i = \lceil \log |pq| \rceil$.

LEMMA 2.4. *For any real number ℓ with $0 < \ell \leqslant (1+\varepsilon)^2 2^{i+a}$, it takes $O(1)$ time to decide if $\delta_H(p,q) \leqslant \ell$. If this is the case, then $\delta_H(p,q)$ can be computed in $O(1)$ time. The constants in the big-Oh notation depend only on d, t, and ε.*

PROOF. Recall that H is the cluster graph obtained at the end of phase $i+a$, i.e., after all edges of length at most 2^{i+a} have been processed. The lemma follows from the discussion following algorithm SHORT-PATH in Das and Narasimhan [1997] and from Section 3.3 in Gudmundsson et al. [2002]. □

The intuition for the above proof comes from the fact that graph H has constant degree with regard to "long" edges, and it can be shown that if there is a path between p and q in H of length at most ℓ, then it cannot use a sequence of "short" edges, and that if it uses a sequence of "long" edges, there can only be a constant number of them. Thus the queries are answered by searching in a constant-sized subgraph of H. For further details, we refer the reader to the earlier papers Das and Narasimhan [1997], Gudmundsson et al. [2002].

In order to facilitate answering approximate distance queries, our data structure simply consists of the collection of cluster graphs H_i, for $1 \leqslant i \leqslant 1 + \lfloor \log C \rfloor$, where H_i is the cluster graph at the moment when the greedy algorithm has just processed all edges of the subset E_i. 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 $|pq| > 1$.

Query algorithm: *Compute $i = \lceil \log |pq| \rceil$, and output $\delta_{H_{i+a}}(p,q)$.*

It follows from Lemmas 2.3 and 2.4 that $\Delta := \delta_{H_{i+a}}(p,q)$ can be computed in $O(1)$ time and satisfies $\delta_G(p,q) \leqslant \Delta \leqslant (1+\varepsilon)^2 \cdot \delta_G(p,q)$. What remains is to show how to compute i. This discussion is deferred to Section 5, where we use a second data structure to show in Corollary 5.3 that the integer i can be computed in $O(1)$ time.

Thus, our data structures allow us to answer $(1+\varepsilon)^2$-approximate shortest path length queries in constant time, for pairs of points with interpoint distance at least 1. If we replace ε by $\varepsilon/3$ in the entire construction, and observe that $(1+\varepsilon/3)^2 \leqslant 1+\varepsilon$, then we have proved the following result:

THEOREM 2.5. *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 $t > 1$) with $O(n)$ edges. Let D be the length of the longest edge in G, let ε be a positive real constant, and let $C > 2$ be a real number. In $O(n \log n + n \log C)$ time, we can preprocess G into a data structure of size $O(n \log C)$, such that for any two points p and q in V with $|pq| > D/C$, we can compute, in $O(1)$ time, a $(1+\varepsilon)$-approximation to the length of the shortest path in G between p and q.*

In the next section, we will need a generalization of Theorem 2.5. The following theorem states that Theorem 2.5 remains true even for partial spanners.

THEOREM 2.6. *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 $t > 1$) with $O(n)$ edges. Let $C > 2$ be a real number, and let ε be a positive real constant. In $O(n \log n + n \log C)$ time, we can preprocess G into a data structure of size $O(n \log C)$, such that for any two points p and q of V with $L/C \leqslant |pq| < L$, we can compute, in $O(1)$ time, a $(1+\varepsilon)$-approximation to the length of the shortest path in G between p and q.*

PROOF. Consider two points p and q in V, such that $|pq| < L$. Since G is an L-partial t-spanner, we have $\delta_G(p,q) \leqslant t|pq| < tL$, and, therefore, each edge on a shortest path in G between p and q has length less than

tL. Thus, in order to answer $(1+\varepsilon)$-approximate distance queries for points p and q with $L/C \leqslant |pq| < L$, we only have to consider the edges in G whose lengths are less than tL. Let G_0 denote the subgraph of G consisting of these edges, and let $D = tL$. Then, we want to answer $(1+\varepsilon)$-approximate distance queries in G_0, for pairs p, q of points in V with $D/(tC) \leqslant |pq| < L$. The proof of the theorem is now identical to that of Theorem 2.5. □

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 2.5 to query points that may not be "sufficiently separated". For the preprocessing to support arbitrary queries, an obvious approach is to construct a sequence of graphs that approximates G and fulfills the requirements of Theorem 2.6. In this section, we will show how this can be done.

The construction of the query structure consists of several steps. First we construct a hierarchy of so-called component graphs, denoted $G_1, \ldots, G_\ell$. The general idea is to partition E into subsets $E_1, \ldots, E_\ell$ such that the edge lengths within each subset differ by a factor of $n^{O(1)}$ of each other. For each index i, we consider each connected component in $(V, E_1 \cup E_2 \cup \ldots \cup E_i)$ as a supervertex, the set of supervertices 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 supervertices 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 supervertex representing the connected component containing p and y is the supervertex 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 next step of the construction is needed to fulfill the condition in Theorem 2.6 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 $G_1, \ldots, G_\ell$ 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:

(1) the pair of points p and x (and also the pair q and y) are much closer to each other than the pair p and q, and

(2) $|xy|$ is within a factor of $n^{O(1)}$ more than the edge lengths in E_i.

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

3.1 Constructing 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 with $O(n)$ edges. We fix a real constant $\varepsilon > 0$ and an integer constant $c > 6$. The following algorithm partitions the edge set E into subsets (some of which are empty).

Algorithm PARTITIONEDGES
$E' := E$;
$L :=$ length of a shortest edge in E';
$L_1 := n^c L$; $L_2 := n^c L_1$; $L_3 := n^c L_2$;
$E_1 :=$ set of all edges in E' whose lengths are less than L_1;
$E' := E' \setminus E_1$;
$E_2 :=$ set of all edges in E' whose lengths are less than L_2;
$E' := E' \setminus E_2$;
$E_3 :=$ set of all edges in E' whose lengths are less than L_3;
$E' := E' \setminus E_3$;
$i := 3$;
while $E_i \neq \emptyset$ or $E' \neq \emptyset$
do if $E_i \neq \emptyset$
  then $L_{i+1} := n^c L_i$
  else $L :=$ length of a shortest edge in E';
    $L_{i+1} := n^c L$

endif;
$E_{i+1} :=$ set of all edges in E' whose lengths are less than L_{i+1};
$E' := E' \setminus E_{i+1}$;
$i := i + 1$
endwhile;
$\ell := i$

Consider the sequence $L_1, L_2, \ldots, L_\ell$ of real numbers and the sequence $E_1, E_2, \ldots, E_\ell$ of edge sets that are computed by this algorithm. For each i with $1 \leqslant i \leqslant \ell$, we define the interval I_i by

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

The following lemma follows immediately from the above algorithm, and our assumption that the edge set E has size $O(n)$.

LEMMA 3.1. *The following properties hold.*

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

We now define a sequence $G_i = (V_i, F_i)$, $1 \leqslant i \leqslant \ell$, of Euclidean graphs, and a sequence U_i, $1 \leqslant i \leqslant \ell$, of forests over V. Observe that there may be an edge e in the edge set E_i and an edge e' in the neighboring edge set E_{i+1}, such that e and e' have almost the same length. Each edge in the next set E_{i+2}, however, has a length that is at least n^c times the length of e. In other words, edges in E_i and edges in E_{i+2} have lengths that differ by a factor of at least n^c. This property is crucial for the analysis that will be presented in the next subsections. Because of this, our recursive construction of the forest U_i will be in terms of the forest U_{i-2}.

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

Algorithm COMPUTECOMPONENTGRAPHS
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 x and y into the vertex set V_i, and insert the edge (x, y) into the edge set F_i. This situation is illustrated in Figures 1(a) and (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 the vertex set of C. Consider all trees T in U_{i-2} such that the points stored in their roots form the vertex set of 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. This situation is illustrated in Figures 1(b) and (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.*

The following lemma states that the connected components of G_i correspond to the trees in U_i.

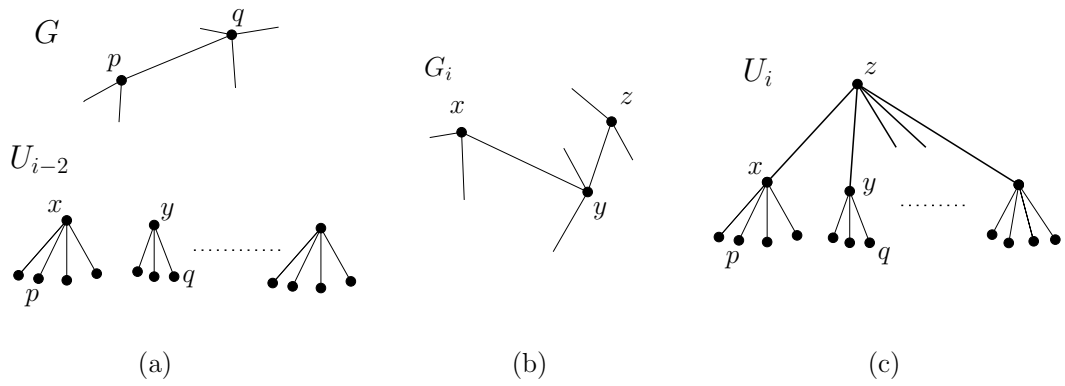

Fig. 1. Illustrating the construction of G_i and U_i.

LEMMA 3.2. *For each i with $1 \leqslant i \leqslant \ell$, 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 $E_1 \cup E_2 \cup \ldots \cup E_i$. 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 is a connected graph. □

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 in the following lemma:

LEMMA 3.3. *The following properties hold.*

(*1*) *For any i with $1 \leqslant i \leqslant \ell$, let T be a tree in the forest U_i, let x and p be arbitrary points stored in T. Then $|px| < nL_i$.*

(*2*) *For each i with $1 \leqslant i \leqslant \ell$, every edge in G_i has length less than $2L_i$.*

PROOF. Let G^i be the graph with vertex set V and edge set $E_1 \cup E_2 \cup \ldots \cup E_i$. If p and x are in the same tree of the forest U_i, then, by Lemma 3.2, they are in the same connected component of G^i. Since the length of each edge in G^i is less than L_i, and since any path between p and x in G^i contains less than n edges, it follows that $|px| \leqslant \delta_{G^i}(p, x) < nL_i$. This proves the first claim.

The second claim clearly holds if $i \in \{1, 2\}$. Assume that $3 \leqslant i \leqslant \ell$, and let (a, b) be an arbitrary edge of G_i. Then $a \neq b$ and there is an edge (p, q) in $E_{i-1} \cup E_i$ such that the forest U_{i-2} contains two distinct trees T' and T'' such that (i) the root of T' stores a, (ii) p is stored in T', (iii) the root of T'' stores b, and (iv) q is stored in T''. By the first claim, we have $|ap| < nL_{i-2}$ and $|qb| < nL_{i-2}$. Therefore,

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

Since $(p, q) \in E_{i-1} \cup E_i$, we have $|pq| < L_i$. Moreover, by Lemma 3.1, $L_{i-2} \leqslant L_i/n^{2c}$. It follows that

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

where the last inequality follows from the fact that $n^{2c-1} > 2$. Hence, we have shown that the length of every edge in G_i is less than $2L_i$. □

Now we are ready to prove the main results of this section, Lemmas 3.4 and 3.5.

LEMMA 3.4. *Let i be an index such that $1 \leqslant i \leqslant \ell$, and let x and y be two vertices of G_i such that $L_i/n^{c+4} \leqslant |xy| < L_i/t$. Then*

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

PROOF. Let $P = (x = x_0, x_1, \ldots, x_k = y)$ 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 assumed in this lemma. Hence, $|x_j x_{j+1}| < L_i$ for each j with $0 \leqslant j \leqslant k-1$. If $i \leqslant 2$, then we have $\delta_{G_i}(x,y) = \delta_G(x,y)$, and the lemma is true. So assume that $i \geqslant 3$.

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 $(1+\varepsilon)$ times the length of P. During the conversion, we will maintain the following invariant.

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

We start the conversion by setting $j := 0$, $k' := 0$, and $y_0 := x_0$. Since x_0 is a vertex of G_i, it must be a root of a tree in U_{i-2} and 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 $E_1 \cup \ldots \cup E_i$. Since x_j and x_{j+1} are stored in different trees of U_{i-2}, this edge cannot be contained in $E_1 \cup \ldots \cup E_{i-2}$. 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

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

By Lemma 3.3, both $|y_{k'} x_j|$ and $|x_{j+1} y_{k'+1}|$ are less than $nL_{i-2} \leqslant L_i/n^{2c-1} \leqslant |xy|/n^{c-5}$, which implies that

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

Since the path Q contains less than n edges, it follows that the length of Q is less than the length of P plus $2|xy|/n^{c-6}$. If $c > 6$, then for any fixed constant $\varepsilon > 0$ and n sufficiently large, we can make $2/n^{c-6}$ less than ε. Therefore,

$$\begin{aligned} \delta_{G_i}(x,y) &< \delta_G(x,y) + \frac{2}{n^{c-6}}|xy| \\ &\leqslant \delta_G(x,y) + \varepsilon|xy| \\ &\leqslant (1+\varepsilon) \cdot \delta_G(x,y). \end{aligned}$$

This completes the proof. □

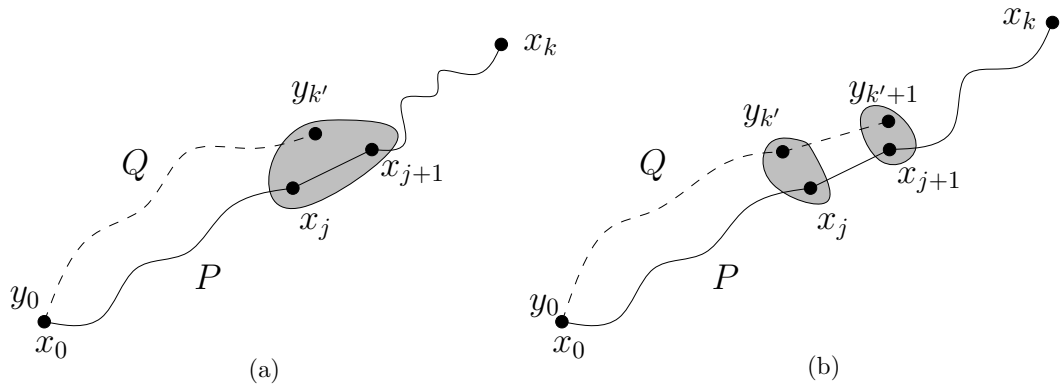

Fig. 2. Illustration for the proof of Lemma 3.4.

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

LEMMA 3.5. *Let i be an index such that $1 \leqslant i \leqslant \ell$, and let x and y be two vertices of G_i such that $L_i/n^{c+4} \leqslant |xy| < L_i/t$. Then*

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

PROOF. As in the previous lemma, we have $\delta_G(x, y) = \delta_{G_i}(x, y)$ for $i \in \{1, 2\}$. So we may assume that $i \geqslant 3$. 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 3.3, both $|ap|$ and $|bq|$ are less than $nL_{i-2} \leqslant L_i/n^{2c-1}$. Therefore,

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

Since G is a t-spanner, we have

$$\delta_G(a, p) \leqslant t|ap| \leqslant tL_i/n^{2c-1},$$

and

$$\delta_G(q, b) \leqslant t|qb| \leqslant tL_i/n^{2c-1}.$$

By combining these inequalities, it follows that

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

Now consider the points x and y. It follows from Lemma 3.4 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 (2) to each edge on this path, then we obtain

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

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

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

The last inequality comes about because $c > 6$, and for any $\varepsilon > 0$, for sufficiently large n, we can make $2(t + 1)/n^{c-6}$ less than ε. This completes the proof. □

3.3 Extending the component graphs to partial spanners

Let i be an index such that $1 \leqslant i \leqslant \ell$, and let x and y be two vertices of G_i such that $L_i/n^{c+4} \leqslant |xy| < L_i/t$. Then it follows from Lemma 3.4 and the fact that G is a t-spanner that

$$\delta_{G_i}(x, y) \leqslant (1 + \varepsilon) \cdot \delta_G(x, y) \leqslant (1 + \varepsilon)t|xy|. \tag{3}$$

We would like to apply Theorem 2.6 to the graph G_i. However, this is only possible if *each* pair of vertices of G_i having Euclidean distance less than L_i/t is connected by a $t(1 + \varepsilon)$-spanner path. Observe that this is true if $i = 1$ and $i = 2$. The following algorithm takes the sequence of component graphs and constructs a sequence of augmented component graphs that have the required property for all i. The required correctness proofs are detailed in Section 3.4.

Algorithm COMPUTEAUGMENTEDCOMPONENTGRAPHS
For each i with $3 \leqslant i \leqslant \ell$, let G'_i be a $((1 + \varepsilon)t)$-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 [1991] or by Callahan and Kosaraju [1993]. 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}.

3.4 Properties of the augmented component graphs

In this section we will prove that the sequence of graphs G''_i collectively 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 $\delta_G(p, q)$. First we start with the following lemma.

LEMMA 3.6. *Let i be an index with $1 \leqslant i \leqslant \ell$, and let x and y be two distinct vertices of G_i'' such that $|xy| < L_i/t$. Then*

(1) $\delta_{G_i''}(x,y) \leqslant (1+\varepsilon)t|xy|$, *and*

(2) if $L_i/n^{c+1} \leqslant |xy|$, *then* $\frac{\delta_G(x,y)}{1+\varepsilon} \leqslant \delta_{G_i''}(x,y) \leqslant (1+\varepsilon)\cdot\delta_G(x,y)$.

PROOF. The claims are true if $i \in \{1,2\}$. Assume that $i \geqslant 3$. If $L_i/n^{c+4} \leqslant |xy|$, then the first claim follows from inequality (3) 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 $((1+\varepsilon)t)$-spanner for V_i, we have

$$\delta_{G_i'}(x,y) \leqslant (1+\varepsilon)t|xy| < (1+\varepsilon)tL_i/n^{c+4} \leqslant L_i/n^{c+3}.$$

Hence, the shortest path in G_i' between x and y is completely contained in G_i''. Therefore, by the first inequality in the previous chain, we have

$$\delta_{G_i''}(x,y) \leqslant \delta_{G_i'}(x,y) \leqslant (1+\varepsilon)t|xy|,$$

proving the first claim.

To prove the second claim, assume that $L_i/n^{c+1} \leqslant |xy|$ as stated. Since G_i is a subgraph of G_i'', we have $\delta_{G_i''}(x,y) \leqslant \delta_{G_i}(x,y)$. By Lemma 3.4, we have $\delta_{G_i}(x,y) \leqslant (1+\varepsilon)\cdot\delta_G(x,y)$, thus proving the second part of the second claim. It thus remains to prove that $\delta_G(x,y) \leqslant (1+\varepsilon)\cdot\delta_{G_i''}(x,y)$.

Let $x = x_0, x_1, \ldots, x_k = y$ be a shortest path in G_i'' between x and y. (Since G_i'' contains a spanner on the vertices in V_i, such a path must exist.) Let j be any index with $0 \leqslant j \leqslant k-1$ 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 inequality (2) in the proof of Lemma 3.5 that

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

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

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

Hence, we always have

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

It follows that

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

Since $c > 6$, $L_i \leqslant n^{c+1}|xy|$, and $k \leqslant n$, we obtain

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

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

Recall from Lemma 3.1 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 and will be instrumental in answering the query.

LEMMA 3.7. *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 Figure 3(a). Finally, let i be the index that is stored with u. The following inequalities hold.*

(1) $|xy| < 2nL_{i-2} + |pq|$.

(2) $|pq| < 2nL_{i-2} + |xy|$.

(3) $|pq| \geqslant L_{i-2}/t$.

(4) $|xy| \geqslant L_{i-2}/t$.
(5) $|xy| < \left(n + 2/n^{2c-2}\right) L_i$.

PROOF. Let j be the index stored with v. Then v is the root of a tree in the forest U_j and $j \leqslant i-2$. Hence, by Lemma 3.3, $|xp| < nL_j \leqslant nL_{i-2}$. In a similar way, we obtain $|qy| < nL_{i-2}$. It follows that

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

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

To prove the third claim, assume the contrary, i.e., that $|pq| < L_{i-2}/t$. Since G is a t-spanner, we have $\delta_G(p,q) \leqslant t|pq| < L_{i-2}$. Hence, p and q are connected by a path in the graph with vertex set V and edge set $E_1 \cup E_2 \cup \ldots \cup E_{i-2}$. But then, by Lemma 3.2, p and q must have been stored in the same tree of the forest U_{i-2}, which is a contradiction. The fourth claim can be proved in the same way.

It remains to prove the fifth claim. Observe that v and w are children of u because the points x and y are vertices of G_i that are connected by a path in this graph. Consider any path in G_i between x and y, and let (x', y') be a longest edge on this path. There exists an edge $(a, b) \in E_{i-1} \cup E_i$ such that a is stored in the tree of U_{i-2} whose root stores x', and b is stored in the tree of U_{i-2} whose root stores y', as shown in Figure 3(b). By Lemma 3.3, we have $|ax'| < nL_{i-2}$ and $|by'| < nL_{i-2}$. It follows that

$$\begin{aligned} |xy| &\leqslant \delta_{G_i}(x,y) \\ &\leqslant n|x'y'| \\ &\leqslant n\left(|x'a| + |ab| + |by'|\right) \\ &< 2n^2 L_{i-2} + nL_i \\ &\leqslant 2L_i/n^{2c-2} + nL_i. \end{aligned}$$

This completes the proof of the lemma. □

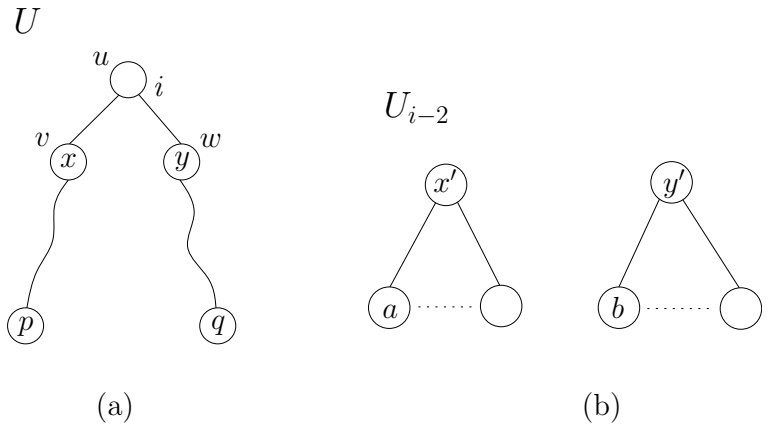

Fig. 3. Illustration of Lemma 3.7 and Lemma 3.8.

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

LEMMA 3.8. *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 Figure 3(a). Finally, let i be the index that is stored with u. If $L_i/n^{c+1} \leqslant |xy| < L_i/t$, then*

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

PROOF. We assume that $i \geqslant 3$. The case when $i \in \{1, 2\}$ is easy and left to the reader. First observe that x and y are vertices of the graph G''_i. By Lemma 3.6, we have $\delta_G(x, y) \leqslant (1 + \varepsilon) \cdot \delta_{G''_i}(x, y)$ and $\delta_{G''_i}(x, y) \leqslant (1+\varepsilon)\cdot\delta_G(x, y)$. By Lemma 3.3, we have $|px| < nL_{i-2} \leqslant L_i/n^{2c-1}$ and $|yq| < nL_{i-2} \leqslant L_i/n^{2c-1}$. By using these inequalities and the fact that G is a t-spanner, we obtain

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

The proof of the second claim is similar:

$$\begin{aligned}
\delta_{G''_i}(x, y) &\leqslant (1 + \varepsilon) \cdot \delta_G(x, y) \\
&\leqslant (1 + \varepsilon) \left(\delta_G(x, p) + \delta_G(p, q) + \delta_G(q, y)\right) \\
&\leqslant (1 + \varepsilon) \left(t|xp| + \delta_G(p, q) + t|qy|\right) \\
&< (1 + \varepsilon) \left(2t|xy|/n^{c-2} + \delta_G(p, q)\right).
\end{aligned}$$

By Lemma 3.7, we have $|xy| < 2nL_{i-2} + |pq|$ and $|pq| \geqslant L_{i-2}/t$. This implies that $|xy| \leqslant (2nt + 1)|pq|$. Hence,

$$\delta_{G''_i}(x, y) \leqslant (1 + \varepsilon) \left(2t(2nt + 1)|pq|/n^{c-2} + \delta_G(p, q)\right).$$

Since $|pq| \leqslant \delta_G(p, q)$, it follows that $\delta_{G''_i}(x, y) \leqslant (1 + \varepsilon)^2 \cdot \delta_G(p, q)$. □

For the following two lemmas, we assume the following situation, as illustrated in Figure 4. We assume that p and q are two distinct points from V and that the following two conditions are satisfied:

(1) Let u be the lowest common ancestor of the leaves in $U_{\ell-1}$ 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.

(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'.

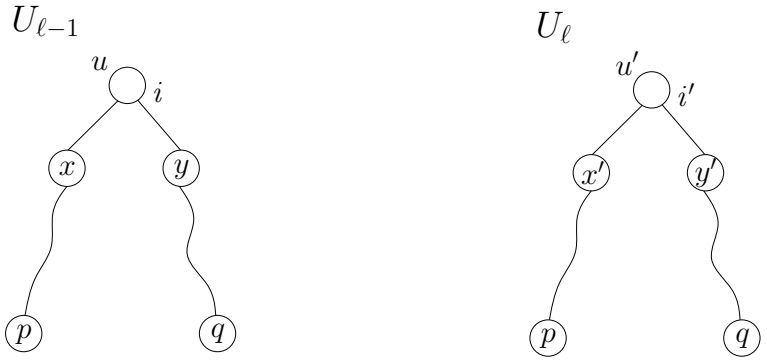

Fig. 4. Illustrations for Lemmas 3.9 and 3.10.

LEMMA 3.9. *If the above two conditions are satisfied, 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 $i \geqslant 3$; the case when $i \in \{1, 2\}$ is easy and left to the reader. We may assume without loss of generality that $i' \geqslant i+1$. Hence, we have to prove that $i' = i+1$. Assume to the contrary that $i' \geqslant i+3$. Then $L_i \leqslant L_{i'-3} \leqslant L_{i'-2}/n^c$. By Lemma 3.7, we have $L_{i'-2} \leqslant t|pq|$, $L_{i-2} \leqslant t|xy|$ and $|pq| \leqslant 2nL_{i-2} + |xy|$. It follows that

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

According to Lemma 3.7 it holds that $|xy| < \left(n + 2/n^{2c-2}\right) L_i$. But then,

$$\left(n + 2/n^{2c-2}\right) L_i \leqslant t\left(n + 2/n^{2c-2}\right)(2nt+1)\,|xy|/n^c < |xy|,$$

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

LEMMA 3.10. *If the same conditions as in Lemma 3.9 apply, then*

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

PROOF. As before, we assume that $i \geqslant 3$ and $i' \geqslant 3$, and leave the easier cases for the reader. By Lemma 3.9, we may assume without loss of generality that $i' = i+1$. Since, by Lemma 3.7, $|xy| \leqslant 2nL_{i-2} + |pq|$ and $|pq| \leqslant 2nL_{i-1} + |x'y'|$, we have

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

Hence,

$$|xy| \leqslant 3L_i/n^{c-1} + |x'y'|. \tag{4}$$

In a similar way, we obtain the inequality

$$|x'y'| \leqslant 3L_i/n^{c-1} + |xy|. \tag{5}$$

By Lemma 3.7, we have $|xy| < \left(n + 2/n^{2c-2}\right) L_i$. Therefore,

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

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

$$|x'y'| < L_{i+1}/n^{c+1}. \tag{6}$$

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

$$L_j/n^{c+1} \leqslant L_j/(tn^c) \leqslant |x'y'|. \tag{7}$$

By combining inequalities (6) and (7), it follows that

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

which implies that $j \leqslant i$. We claim that $j = i$. To prove this, assume that $j \leqslant i-1$. Then x' and y' are connected by a path in the graph with vertex set V and edge set $E_1 \cup E_2 \cup \ldots \cup E_{i-1}$. Hence, by Lemma 3.2, x' and y' are stored in the same tree in the forest U_{i-1}, which is a contradiction. This proves that $j = i$. Note 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 3.1, we have $L_{i+1} = n^c L_i$. Then it follows from inequalities (4) and (6) 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 $L_i/n^{c+1} \leqslant |xy|$. 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 (7) shows that $L_{j'}/n^{c+1} \leqslant |xy|$ and, hence, $L_{j'} \leqslant n^{c+1}|xy| < L_i$, which shows that $j' \leqslant i-1$. If $j' \leqslant i-2$, 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}. To summarize, we have shown 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 3.2, 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, implying 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 $i-1$, which is a contradiction. □

3.5 Preprocessing: tying it all together

Having completed the description of the preprocessing step needed to construct the query structure for supporting arbitrary shortest path length queries, we now summarize this preprocessing step.

Preprocessing algorithm:
Step 1: *Construct the sequence $L_1, \ldots, L_\ell$ using algorithm* PartitionEdges *(see Section 3.1).*
Step 2: *Construct the two trees $U_{\ell-1}$ and U_ℓ and the component graphs G_i, $1 \leqslant i \leqslant \ell$, using algorithm* ComputeComponentGraphs *(see Section 3.1).*
Step 3: *Preprocess the trees $U_{\ell-1}$ and U_ℓ in linear time such that lowest common ancestor queries can be answered in constant time. This can be done using an algorithm by Harel and Tarjan [1984]; see also Schieber and Vishkin [1988], Gusfield [1997] and Bender and Farach-Colton [2000].*
Step 4: *Compute the sequence of graphs G_i'', $1 \leqslant i \leqslant \ell$ using the algorithm* ComputeAugmentedComponentGraphs*, as described at the end of Section 3.3.*
Step 5: *Preprocess the graphs G_i'', $1 \leqslant i \leqslant \ell$ using the algorithm implied by Theorem 2.6.*

Step 5 needs some clarification. By Lemma 3.6, the graph G_i'' is an (L_i/t)-partial $(1+\varepsilon)t$-spanner for the set V_i. We apply Theorem 2.6, where, in this theorem, we replace G by G_i'', replace V by V_i, replace L by L_i/t, replace t by $(1+\varepsilon)t$, and replace C by n^{c+1}. Thus, Theorem 2.6 implies that we can answer $(1+\varepsilon)$-approximate shortest path queries in G_i'', for any two points x and y of V_i that satisfy $L_i/(tn^{c+1}) \leqslant |xy| < L_i/t$. The latter condition is met, if $L_i/n^{c+1} \leqslant |xy| < L_i/t$. By Theorem 2.6, the time to preprocess G_i'' is

$$O\left(|V_i| \log |V_i| + |V_i| \log C\right) = O(|V_i| \log n). \tag{8}$$

3.6 Answering approximate distance queries

Given an $\varepsilon > 0$, the query data structure consists of:

—The sequence $L_1, \ldots, L_\ell$, as described in Section 3.1.

—The two trees $U_{\ell-1}$ and U_ℓ, as described in Section 3.1.

—The data structure of Theorem 2.6 for each of the graphs G_i'', $1 \leqslant i \leqslant \ell$.

Let p and q be two points in V, and consider the two trees $U_{\ell-1}$ and U_ℓ, as shown in Figure 4. Let i and i' be the index values stored in the lowest common ancestor nodes of p and q in the trees $U_{\ell-1}$ and U_ℓ, respectively. From Lemma 3.10, it follows that either i or i' satisfies the conditions of Lemma 3.8 and hence, either $\delta_{G_i''}(x,y)$ or $\delta_{G_{i'}''}(x,y)$ is a close approximation of $\delta_G(p,q)$. Note also that from Lemma 3.6, G_i'' and $G_{i'}''$ satisfy the requirements for Theorem 2.6. Thus the query algorithm is as follows. Let p and q be two distinct points of V.

Query algorithm:

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 Figure 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 $L_i/n^{c+1} \leqslant |xy| < L_i/t$, then use the algorithm of Theorem 2.6 to compute a $(1+\varepsilon)$-approximation Δ to $\delta_{G''_i}(x,y)$. Otherwise, use the algorithm of Theorem 2.6 to compute a $(1+\varepsilon)$-approximation Δ to $\delta_{G''_{i'}}(x',y')$.*

It follows from Lemmas 3.8 and 3.10 that

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

By replacing $(1+2\varepsilon)\Delta$ by Δ', we get

$$\delta_G(p,q) \leqslant \Delta' \leqslant (1+2\varepsilon)(1+\varepsilon)^3 \cdot \delta_G(p,q).$$

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

3.7 Analyzing the complexity of the preprocessing step

Recall that we assume that $|E| = O(n)$. In Step 1 of the preprocessing, the sequences E_i, $1 \leqslant i \leqslant \ell$, and I_i, $1 \leqslant i \leqslant \ell$, 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 $G_i = (V_i, F_i)$, $1 \leqslant i \leqslant \ell$, and U_i, $1 \leqslant i \leqslant \ell$, 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 $|V_i| \leqslant 2|F_i|$. Also, it is clear from the algorithm that constructs G_i, that $|F_i| \leqslant |E_{i-1}| + |E_i|$. Therefore, the time to compute all graphs G_i and all forests U_i (and thus the time complexity of Step 2) is proportional to

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

Step 3 runs in $O(n)$ time (see Harel and Tarjan [1984]).

Using the algorithm of Callahan and Kosaraju [1993], the sequence of spanners G'_i, $1 \leqslant i \leqslant \ell$, can be computed in Step 4 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, $1 \leqslant i \leqslant \ell$, is proportional to

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

Finally, by (8), the total time to preprocess all graphs G''_i (Step 5) is proportional to

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

The discussion above shows how the data structure can be used to answer approximate shortest path queries, within a ratio of $(1+2\varepsilon)(1+\varepsilon)^3$. If we replace ε by $\varepsilon/6$ in the entire reduction, and observe that $(1+2\varepsilon/6)(1+\varepsilon/6)^3 \leqslant 1+\varepsilon$, then we obtain Theorem 1.1, which we restate below. (As mentioned before, the computation of the value $\lceil \log |pq| \rceil$, which is needed in the query algorithm of Section 2.2, will be given in Section 5.)

Theorem 1.1. *Let $t > 1$ and $\varepsilon > 0$ be real constants. Let V be a set of n points in $\mathbb{R}^d$, and let $G = (V, E)$ be a t-spanner for V with $O(n)$ edges. The graph G can be preprocessed into a data structure of size $O(n \log n)$ in time $O(n \log n)$, such that for any pair of 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 $O(1)$ time. Note that all the big-Oh notations hide constants that depend on d, t and ε.*

The proof of Theorem 1.1 is based on the fast implementation of the ImprovedGreedy algorithm of Gudmundsson et al. [2002] (as was discussed in Section 2). This algorithm works in the algebraic model of computation with indirect addressing. If we want an algorithm in the traditional algebraic model of computation (without indirect addressing), then we can replace algorithm ImprovedGreedy by the corresponding algorithm of Das and Narasimhan [1997], which has a running time of $O(n \log^2 n)$. In this model, the computation of the value $\lceil \log |pq| \rceil$, as well as lowest common ancestor computations, can be done in $O(\log \log n)$ time. Thus, we obtain the following result:

Theorem 3.11. *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. This algorithm works in the algebraic model of computation.*

4. APPLICATIONS

The data structure of Theorem 1.1 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 approximation algorithms.

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. [1996] 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 [1987] and Chen [1995] 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 Arikati et al. [1996]). Hence, a $(1 + \varepsilon)^3$-approximation of the shortest obstacle-avoiding path can be computed in $O(\log n)$ time.

Theorem 4.1. *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 path length queries in time $O(\log n)$. If the query points are vertices of $\mathcal{F}$, then the queries can be answered in $O(1)$ time.*

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.

4.2.1 *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., $\delta_G(S) \leqslant \delta_G(x,y) \leqslant (1+\varepsilon)\delta_G(S)$. Here, ε is a fixed positive real constant.

In what follows, we will be making use of the well-separated pair decompositions (WSPD) devised by Callahan [1995], Callahan and Kosaraju [1995]. Details about the decomposition may be found in the above references. Callahan and Kosaraju showed that a WSPD of size $\ell = O(n)$ can be computed in $O(n \log n)$ time. For our purpose, we make use of the following simple lemma from Callahan and Kosaraju [1993], which we restate here for convenience.

LEMMA 4.2. *Let $s > 0$ be a real number, 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) $|xy| \leqslant (1+2/s)\cdot|xq|$, (ii) $|xy| \leqslant (1+4/s)\cdot|pq|$, and (iii) $|px| \leqslant (2/s)\cdot|pq|$.*

Consider a well-separated pair decomposition (WSPD), $\{A_i, B_i\}$, $1 \leqslant i \leqslant \ell$, where $\ell = O(|S|)$, for the set S, with separation constant $s > 2t$. For each i, $1 \leqslant i \leqslant \ell$, 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 4.2, the fact that G is a t-spanner for V, and our assumption that $s > 2t$, imply that

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

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

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

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 4.3. *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 $O(|S| \log |S|)$.*

4.2.2 *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 $\delta_G(X,Y) \leqslant \delta_G(x,y) \leqslant (1+\varepsilon)\cdot\delta_G(X,Y)$. Again, ε is a fixed positive real constant.

Let $\{A_i, B_i\}$, $1 \leqslant i \leqslant \ell$, 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, $1 \leqslant i \leqslant \ell$, 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

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

Also,

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

Combining these inequalities, it follows that

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

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

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 4.4. *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 [2000] 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 results of this paper, 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 linear-sized 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 4.5. *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 2.2.

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 $|pq| \geqslant 1$, 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 $0 \leqslant \text{BINDEX}(p, q) \leqslant \lfloor (1/2) \log d + k \log n \rfloor$.

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. This structure, together with the algorithms that operate on it, work in the algebraic computation tree model with the added power of indirect addressing; in particular, they do not use the floor or logarithm functions. 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 $|x-y| \geqslant 1$, we have $\textsc{BIndex}(x, y) = \lfloor \log |x-y| \rfloor$, 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 $[j, j+1)$, $0 \leqslant j < n^k$. 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 $b-a \geqslant 2$, $|V'| \geqslant 2$ 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 $b-a \geqslant 2$, $|V'| \geqslant 2$ 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 $V'_2 := \{x \in V' : x \geqslant (a+b)/2\}$. In this case, $T(V')$ consists of a node u storing the interval $I(u) := [a, b)$. The left subtree of u is the output of the recursive call for the set V'_1 and the interval $[a, (a+b)/2)$, whereas the right subtree of u is the output of the recursive call for 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 Answering a query

Let us see how we can use this tree to answer queries. Let x and y be two elements of V with $|x-y| \geqslant 1$. Recall that we want to compute the integer $\textsc{BIndex}(x, y) = \lfloor \log |x-y| \rfloor$. We may assume w.l.o.g. that $x < y$. We observe that $\textsc{BIndex}(x, y)$ 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 $a \geqslant 0$, $b \geqslant 2$, and $j \geqslant 1$ 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 $\textsc{BIndex}(x, y)$ by a "small" additive constant. It may happen, however, that both x and y are "close" to c. Let us assume that $|c-x| \leqslant |y-c|$. Then $|y-c| < |y-x| \leqslant 2|y-c|$. 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 $\textsc{BIndex}(x, y)$ 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

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

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 $|x - y| \geqslant 1$ and $x < y$, we first test if $y - x = 1$ or $1 < y - x < 2$. In the first case, we have $\textsc{BIndex}(x, y) = 0$, whereas $\textsc{BIndex}(x, y) = 1$ in the second case. Assume that $y - x \geqslant 2$. 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 $|c - x| \leqslant |y - c|$, 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 $y - c \geqslant 1$, the elements y and c are stored at different leaves of T. Then $2^{j-1} \leqslant |y - c| < 2^j$ and, hence, $2^{j-1} < |y - x| \leqslant 2^{j+1}$. This implies that $j - 1 \leqslant \textsc{BIndex}(x, y) \leqslant j + 1$. 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 $\textsc{BIndex}(x, y)$ 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 $\textsc{BIndex}(x, y)$ to answering two lowest common ancestor queries in a tree having size $O(n)$, plus an $O(1)$-time computation. Harel and Tarjan [1984] 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 [1988], Gusfield [1997] and Bender and Farach-Colton [2000]. Hence, we have proved the following result.

THEOREM 5.1. *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 $|x - y| \geqslant 1$, we can compute $\textsc{BIndex}(x, y) = \lfloor \log |y - x| \rfloor$ 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 $|x - y| \geqslant \delta$, 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 $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ be any two points of V with $|pq| \geqslant 1$, let j be such that $|p_j - q_j|$ is maximum, and let $i = \lfloor \log |p_j - q_j| \rfloor$. Since

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

we have

$$i \leqslant \textsc{BIndex}(p, q) \leqslant \frac{1}{2} \log d + i.$$

This suggests the following solution. For each ℓ, $1 \leqslant \ell \leqslant d$, we build the data structure of Theorem 5.1 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 5.1 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 $\textsc{BIndex}(p, q)$ in $O(\log \log d)$ time. Observe that we can indeed apply Theorem 5.1, because $|p_j - q_j| \geqslant 1/\sqrt{d}$. This gives the following result.

THEOREM 5.2. *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 $|pq| \geqslant 1$, we can compute*

$$\textsc{BIndex}(p, q) = \lfloor \log |pq| \rfloor$$

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 $|pq| \geqslant 1$. Hence, $\text{BINDEX}_\varepsilon(p,q)$ is the integer i such that $(1+\varepsilon)^i \leqslant |pq| < (1+\varepsilon)^{i+1}$. Let $j := \text{BINDEX}(p,q)$. Then a straightforward calculation shows that

$$\left\lfloor \frac{j}{\log(1+\varepsilon)} \right\rfloor \leqslant \text{BINDEX}_\varepsilon(p,q) \leqslant \left\lceil \frac{j+1}{\log(1+\varepsilon)} \right\rceil - 1. \tag{9}$$

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 (9). 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 5.3. *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 $|pq| \geqslant 1$, we can in constant time compute*

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

6. EXTENSIONS TO GEOMETRIC SPANNER GRAPHS WITH SUPERLINEAR SIZE

The distance oracle algorithm presented so far only applies to geometric spanner graphs with a linear number of edges. The result can be extended to geometric spanner graphs with superlinear size.

In a recent conference paper, Gudmundsson et al. [2005] have proved that a geometric t-spanner G on n points in $\mathbb{R}^d$ and m edges can be "pruned" to obtain a $(1+\varepsilon)$-spanner G' of G with $O(n)$ edges. Hence, G' is a $t(1+\varepsilon)$-spanner on the same points. Here ε is any given positive constant, and the time complexity of the algorithm was shown to be $O(m + n \log n)$. The corresponding time complexity in the traditional algebraic model of computation (without indirect addressing) is $O(m \log\log n + n \log n)$.

The above result has the following consequence for the results in this paper. Given a geometric t-spanner with m edges, where m is superlinear in n, i.e., $m = \omega(n)$, we can apply the pruning algorithm of Gudmundsson et al. [2005] to first obtain a $t(1+\varepsilon)$-spanner on the input points before applying the algorithm presented here. Hence the following two corollaries are easy consequences.

COROLLARY 6.1. *Let $t > 1$ and $\varepsilon > 0$ be real constants. Let V be a set of n points in $\mathbb{R}^d$, and let $G = (V, E)$ be a t-spanner for V with m edges. The graph G can be preprocessed into a data structure of size $O(n \log n)$ in time $O(m + n \log n)$, such that for any pair of 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 $O(1)$ time. Note that all the big-Oh notations hide constants that depend on d, t and ε.*

In the traditional algebraic model of computation (without indirect addressing) the corresponding weaker result can be stated as follows.

COROLLARY 6.2. *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 m edges. In $O(m \log\log n + n \log^2 n)$ 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. This algorithm works in the algebraic model of computation.*

7. CONCLUDING REMARKS

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

shortest path length queries between vertices in a planar polygonal domain, and efficiently computing the approximate dilation of geometric graphs. Because of the wide applicability of spanners, we expect that many more applications will be discovered.

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 [1992] have shown that this is true for the Delaunay triangulation, which is used in numerous applications. Also, different kinds of transportation networks have small dilations.

Finally, we note two problems that remain open. First, it is not clear if the space utilization of the algorithms in this paper can be reduced from $O(n \log n)$ to $O(n)$ without sacrificing the speed of the algorithms. Secondly, we have encountered technical problems in extending the algorithms to report the approximate shortest path between the given query points. This is caused by the fact that for given query points, p and q, Step 3 of the query algorithm actually reports the length of the shortest path between a pair of points x and y, which may be distinct from the pair of points p and q.

REFERENCES

Agarwal, P. K., Har-Peled, S., and Karia, M. 2000. Computing approximate shortest paths on convex polytopes. In *Proceedings of the 16th ACM Symposium on Computational Geometry*. 270–279.

Agarwal, P. K., Har-Peled, S., Sharir, M., and Varadarajan, K. R. 1997. Approximate shortest paths on a convex polytope in three dimensions. *Journal of the ACM 44*, 567–584.

Aingworth, D., Chekuri, C., Indyk, P., and Motwani, R. 1999. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing 28*, 1167–1181.

Arikati, S., Chen, D. Z., Chew, L. P., Das, G., Smid, M., and Zaroliagis, C. D. 1996. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 4th Annual European Symposium on Algorithms*. Lecture Notes in Computer Science, vol. 1136. Springer-Verlag, Berlin, 514–528.

Arora, S. 1997. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*. 554–563.

Arora, S. 1998. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM 45*, 753–782.

Bartal, Y. 1996. Probabilistic approximations of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*. 184–193.

Baswana, S. and Sen, S. 2004. Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*. 271–280.

Bender, M. A. and Farach-Colton, M. 2000. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*. Lecture Notes in Computer Science, vol. 1776. Springer-Verlag, Berlin, 88–94.

Callahan, P. B. 1995. Ph.D. thesis. Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland.

Callahan, P. B. and Kosaraju, S. R. 1993. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms*. 291–300.

Callahan, P. B. and Kosaraju, S. R. 1995. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM 42*, 67–90.

Chan, T. M. 1998. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry 20*, 359–373.

Chen, D. Z. 1995. On the all-pairs Euclidean short path problem. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*. 292–301.

Chen, D. Z., Daescu, O., and Klenk, K. S. 2001. On geometric path query problems. *International Journal of Computational Geometry & Applications 11*, 617–645.

Chen, D. Z., Klenk, K. S., and Tu, H.-Y. T. 2000. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM Journal on Computing 29*, 1223–1246.

Clarkson, K. L. 1987. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. 56–65.

Cohen, E. 1998. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing 28*, 210–236.

Das, G. and Joseph, D. 1989. Which triangulations approximate the complete graph? In *Proceedings of the International Symposium on Optimal Algorithms*. Lecture Notes in Computer Science, vol. 401. Springer-Verlag, Berlin, 168–192.

Das, G. and Narasimhan, G. 1997. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry & Applications 7*, 297–315.

de Berg, M., Katz, M. J., van der Stappen, A. F., and Vleugels, J. 2002. Realistic input models for geometric algorithms. *Algorithmica 34*, 81–97.

Dor, D., Halperin, S., and Zwick, U. 2000. All-pairs almost shortest paths. *SIAM Journal on Computing 29*, 1740–1759.

Gao, J., Guibas, L. J., Hershberger, J., Zhang, L., and Zhu, A. 2003. Discrete mobile centers. *Discrete & Computational Geometry 30*, 45–63.

Gonzalez, T. 1975. Algorithms on sets and related problems. Technical Report, Department of Computer Science, University of Oklahoma, Norman.

Gudmundsson, J., Levcopoulos, C., and Narasimhan, G. 2002. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM Journal on Computing 31*, 1479–1500.

Gudmundsson, J., Levcopoulos, C., Narasimhan, G., and Smid, M. 2002a. Approximate distance oracles for geometric graphs. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*. 828–837.

Gudmundsson, J., Levcopoulos, C., Narasimhan, G., and Smid, M. 2002b. Approximate distance oracles revisited. In *Proceedings of the 13th International Symposium on Algorithms and Computation*. Lecture Notes in Computer Science, vol. 2518. Springer-Verlag, Berlin, 357–368.

Gudmundsson, J., Narasimhan, G., and Smid, M. 2005. Fast pruning of geometric spanners. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, vol. 3404. Springer-Verlag, Berlin, 508–520.

Guibas, L. J. and Hershberger, J. 1989. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences 39*, 126–152.

Gusfield, D. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

Har-Peled, S. 1997. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. In *Proceedings of the 13th ACM Symposium on Computational Geometry*. 359–365.

Harel, D. and Tarjan, R. E. 1984. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing 13*, 338–355.

Hershberger, J. and Suri, S. 1999. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing 28*, 2215–2256.

Indyk, P. 2001. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*. 10–33.

Kapoor, S., Maheshwari, S. N., and Mitchell, J. S. B. 1997. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete & Computational Geometry 18*, 377–383.

Keil, J. M. and Gutwin, C. A. 1992. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry 7*, 13–28.

Lee, D. T. and Wu, Y. F. 1986. Geometric complexity of some location problems. *Algorithmica 1*, 193–211.

Mitchell, J. S. B. 1996. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry & Applications 6*, 309–332.

Mitchell, J. S. B. 1997. Shortest paths and networks. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL, Chapter 24, 445–466.

Mitchell, J. S. B. 1998. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science, Amsterdam, Chapter 24, 633–701.

Narasimhan, G. and Smid, M. 2000. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing 30*, 978–989.

Overmars, M. H. and van der Stappen, A. F. 1996. Range searching and point location among fat objects. *Journal of Algorithms 21*, 629–656.

Preparata, F. P. and Shamos, M. I. 1988. *Computational Geometry: An Introduction*. Springer-Verlag, Berlin.

Pyrga, E., Schulz, F., Wagner, D., and Zaroliagis, C. 2004. Experimental comparison of shortest path approaches for timetable information. In *Proceedings of the 6th Workshop Algorithm Engineering and Experiments*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, 88–99.

Salowe, J. S. 1991. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications 1*, 99–107.

Schieber, B. and Vishkin, U. 1988. On finding lowest common ancestors: simplifications and parallelisations. *SIAM Journal on Computing 17*, 327–334.

Suri, S. 1997. Polygons. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL, Chapter 23, 429–444.

Thorup, M. 2004. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM 51*, 993–1024.

Thorup, M. and Zwick, U. 2001. Approximate distance oracles. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*. 183–192.

Wagner, D. and Willhalm, T. 2003. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *Proceedings of the 11th Annual European Symposium on Algorithms*. Lecture Notes in Computer Science, vol. 2832. Springer-Verlag, Berlin, 776–787.

Wagner, D., Willhalm, T., and Zaroliagis, C. 2004. Dynamic shortest path containers. In *Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways*. Electronic Notes in Theoretical Computer Science, vol. 92. Elsevier, 65–84.

Received XX; accepted YY