Approximate Distance Oracles for Geometric Graphs

Joachim Gudmundsson* Christos Levcopoulos† Giri Narasimhan‡ Michiel Smid§

Abstract

Given a geometric t-spanner graph G in $\mathbb{E}^d$ with n points and m edges, with edge lengths that lie within a polynomial (in n) factor of each other. Then, after $O(m+n\log n)$ preprocessing, we present an approximation scheme to answer $(1+\varepsilon)$-approximate shortest path queries in $O(1)$ time. The data structure uses $O(n\log n)$ space.

1 Introduction

The *shortest-path* (SP) problem for weighted graphs with n vertices and m edges is a fundamental problem for which efficient solutions can now be found in any standard algorithms text. The approximation version of this problem has been studied extensively, see [1, 11, 14]. In numerous algorithms, query versions frequently appear as subroutines. In such a query, we are given two vertices and have to compute or approximate the shortest path between them. The latest in a series of results for undirected weighted graphs is by Thorup and Zwick [23]; their algorithm computes $(2k-1)$-approximate solutions to the query version of the SP problem in $O(k)$ time, using a data structure that takes (expected) time $O(kmn^{1/k})$ to construct and utilizes $O(kn^{1+1/k})$ space. It is not an approximation scheme in the true sense because the value k needs to be a positive integer. Since the query time is essentially bounded by a constant, Thorup and Zwick refer to their queries as approximate *distance oracles.*

We focus on the geometric version of this problem. A geometric graph has vertices corresponding to points in $\mathbb{R}^d$ and edge weights from a Euclidean metric. Again considerable previous work exists on the shortest path and related problems. A good survey can be found in [19]. Also see [2, 7, 8, 9]. Note that none of the existing results answer shortest path queries in constant time, with subquadratic space and preprocessing time.

We consider geometric graphs that are t-spanners for some (possibly large) constant t. A graph $G=(V,E)$ is said to be a *t-spanner* for V, if for any two points p and q in V, there exists a path in G between p and q of length at most t times the Euclidean distance between p and q. Given a geometric t-spanner $G=(V,E)$ on n vertices and m edges, we consider the problem of finding $(1+\varepsilon)$-approximate solutions to shortest path length queries. We present an algorithm that preprocesses G in $O(n\log n+m)$ time, after which such queries can be answered in $O(1)$ time, for any given real constant $\varepsilon>0$. Also, a $(1+\varepsilon)$-approximate shortest path can be reported in time proportional to the number of its edges. We remark that many "naturally occurring" geometric graphs are t-spanners, for some constant $t>1$, thus justifying the interest in the restricted inputs considered in this paper. For example, Keil and Gutwin [18] have shown that the Delaunay triangulation of a set of points in the plane is a t-spanner for $t=\frac{2\pi}{3\cos\pi/6}$. We impose yet another restriction on our inputs, i.e., that the interpoint distances be confined to a polynomial range. While this too can be argued to be true of most "naturally occurring" geometric graphs, we are working to eliminate this restriction.

We believe that the data structure presented can be applied to a large number of problems. In

*Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. E-mail: joachim@cs.uu.nl. Supported by The Swedish Foundation for International Cooperation in Research and Higher Education.

†Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. E-mail: christos@cs.lth.se.

‡School of Computer Science, Florida International University, Miami, FL 33199, USA. E-mail: giri@fiu.edu.

§School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6. E-mail: michiel@scs.carleton.ca.

this paper we give several examples. For example, consider the problem of approximating a shortest obstacle-avoiding path between two vertices in a planar polygonal domain with obstacles, having a total of n vertices. In accordance with the spanner constraints imposed above, we consider restricted planar polygonal domains, which we will refer to as t-rounded domains. A polygonal domain P is said to be *t-rounded* if for any two vertices p and q, the length of the shortest obstacle-avoiding path between p and q is at most t times the Euclidean distance between them. The concept of t-roundedness is similar in spirit to that of domains with *fat* obstacles or with bounded *aspect ratio* obstacles, see [13, 21]. 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 $O(1)$ time for any two vertices of the domain, and in $O(\log n)$ time for any two points in the plane.

We also consider interesting query versions of *closest pair* problems and show how to answer such queries efficiently. Finally, we significantly improve on the time complexity of algorithms to compute the approximate stretch factor of geometric graphs.

The main data structure used is a series of "cluster graphs" of increasing coarseness each of which helps answer queries for different size edges. The idea of cluster graphs to speed up geometric algorithms was first introduced by Das and Narasimhan [12] and later used by Gudmundsson *et al.* [15] to design an efficient algorithm to compute $(1+\varepsilon)$-spanners.

In this paper, we develop two interesting tools. First an $O(m \log n)$-time algorithm for constructing a $(1+\varepsilon)$-spanner of a given geometric t-spanner. If the ratio between the longest and shortest edge in the given graph is polynomially bounded then a $(1+\varepsilon)$-spanner can be built in time $O(m + n \log n)$. Secondly, we design a tool to bucket edge lengths without using the floor functions. We believe that these tools (especially the latter) are of independent interest as general devices to speed up geometric algorithms.

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

2 A bucketing tool

In this section, we develop an algorithmic tool that will be used repeatedly later in this paper.

THEOREM 2.1. *Let V be a set of n points in $\mathbb{R}^d$ that are contained in the hypercube $(0, n^k)^d$, for some-positive integer constant k. 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 $|xy| \geqslant 1$, we can compute*

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

in $O(1)$ time.

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. In the purely algebraic model, a query can be answered in $O(\log \log n)$ time. The main result of this section is a data structure in the stronger model that additionally has indirect-addressing as a unit-time operation. We start by presenting this data structure for the one-dimensional case. In Section 2.2, we show how to use this result to solve the d-dimensional case.

2.1 The one-dimensional case In this subsection, we 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 $\text{BINDEX}(x,y) = \lfloor \log |x-y| \rfloor$, which is an integer between zero and

$\lfloor k \log n \rfloor$. We assume w.l.o.g. that n is a power of two.

Let T_0 be the perfectly balanced binary search tree whose leaves store—from left to right—the intervals $[j, j+1)$, $0 \leqslant j < n^k$. With each 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. We 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 $O(k \log n)$, which is $O(\log n)$, because k is a constant, the height of T is also $O(\log n)$.

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 elements 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 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 $I(u) := [a, b)$ and whose left (resp. right) subtree is the output of the recursive call for the set V_1' and the interval $[a, (a+b)/2)$ (reps. 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 such that the interval $I(u)$ has length 2^i. The running time to build this tree is $O(nk \log n) = O(n \log n)$.

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 $\text{BINDEX}(x, y) = \lfloor \log |x - y| \rfloor$. We may assume w.l.o.g. that $x < y$. We observe that $\text{BINDEX}(x, y)$ 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 $\text{BINDEX}(x, y)$ 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 $\text{BINDEX}(x, y)$ 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

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

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

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

Given two elements x and y in V with $|x - y| \geqslant 1$ and $x < y$, we first test if $y - x = 1$ or $1 < y - x < 2$. In the first case, we have $\text{BI}\textsc{ndex}(x, y) = 0$, whereas $\text{BI}\textsc{ndex}(x, y) = 1$ in the second case. Assume that $y - x \geqslant 2$. We first 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 $j - 1 \leqslant \text{BI}\textsc{ndex}(x, y) \leqslant j + 1$. Since we know the values of j (it is stored with node v) and 2^j (it is the length of the interval $I(v)$), we can now easily compute $\text{BI}\textsc{ndex}(x, y)$ in $O(1)$ time.

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

To summarize, we have shown that after an $O(n \log n)$-time preprocessing, we can reduce the problem of computing $\text{BI}\textsc{ndex}(x, y)$ to answering two lowest common ancestor queries in a tree having size $O(n)$, plus an $O(1)$-time computation. Harel and Tarjan [17] have shown 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 [22], Gusfield [16] and Bender and Farach-Colton [5].) Hence, we have proved Theorem 2.1 for $d = 1$.

2.2 The d-dimensional case Now assume that V is a d-dimensional set of points. Let $p = (p_1, p_2, \dots, p_d)$ and $q = (q_1, q_2, \dots, q_d)$ be any two points of V with $|pq| \geqslant 1$, 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

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

This suggests the following solution. For each ℓ, $1 \leqslant \ell \leqslant d$, we build the data structure presented above 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 suggested algorithm to compute the integer $i = \lfloor \log |p_j - q_j| \rfloor$. Note that this algorithm also gives us the value 2^i. Given i and 2^i, we then compute $\text{BI}\textsc{ndex}(p, q)$ in $O(\log \log d)$ time, and Theorem 2.1 follows.

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

3 Pruning dense spanners

Let $G = (V, E)$ be a geometric graph that is a t-spanner for V, for some (possibly large) real constant $t > 1$. We denote the number of vertices and edges of G by n and m, respectively. In this section, we will show how to compute, for any given positive real constant ε, a $(1 + \varepsilon)$-spanner of G having only $O(n)$ edges. Our construction uses the fair-split tree and the well-separated pair decomposition (WSPD) of Callahan and Kosaraju; see [6].

LEMMA 3.1. *Let A and B be two finite sets of points that are well-separated w.r.t. a separation constant 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) $|px| \leqslant (2/s) \cdot |pq|$.*

3.1 A general algorithm Recall that $G = (V, E)$ is a t-spanner of V with n vertices and m edges. Our goal is to compute a sparse $(1+\varepsilon)$-spanner G' of G.

We start by computing a WSPD $\{A_i, B_i\}$, $1 \leqslant i \leqslant \ell$, for V, with separation constant $s = 4(1+(1+\varepsilon)t)/\varepsilon$ and $\ell = O(n)$. Let $G' = (V, E')$ be the graph that contains for each $1 \leqslant i \leqslant \ell$, exactly one (arbitrary) edge (x_i, y_i) of E with $x_i \in A_i$ and $y_i \in B_i$, provided such an edge exists (for this index i).

For each edge (p, q) of E, there is a path in G' between p and q having length at most $(1+\varepsilon)|pq|$. (This can be proved by induction on the length of the edges of E.) Hence, the graph G' is a $(1+\varepsilon)$-spanner of G.

We now show how the graph G' can be constructed efficiently. To compute the WSPD, we use the algorithm of Callahan and Kosaraju [6]. Given this WSPD, it is clearly sufficient to compute for each edge (p, q) of E, the index i such that $p \in A_i$ and $q \in B_i$. Arya *et al.* [3] have shown that, after an $O(n)$-time preprocessing of the split tree that gave rise to the WSPD, this index can be computed in $O(\log n)$ time. Hence, the entire graph G' can be computed in $O(n \log n + m \log n) = O(m \log n)$ time.

THEOREM 3.1. *Given a geometric graph G with n vertices and m edges, which is a t-spanner for V, for some real constant $t > 1$, we can compute, in $O(m \log n)$ time, a $(1+\varepsilon)$-spanner of G having $O(n)$ edges.*

3.2 A special case We assume that the distance between any two distinct points of V is in the interval $[1, n^k)$, where k is a positive integer constant. Consider the t-spanner $G = (V, E)$. We will show how a sparse $(1+\varepsilon)$-spanner of G can be computed in $O(n \log n + m)$ time.

The length-grouping step: Let $\varepsilon_0 := \frac{\varepsilon}{2(1+t(1+\varepsilon))}$. Consider the covering of $[1, n^k)$ by the intervals $I_i := [(1+\varepsilon_0)^i, (1+\varepsilon_0)^{i+1})$, $0 \leqslant i = O(\log n)$. For each vertex $p \in V$, we partition p's adjacency list using these intervals. That is, the edge set $E(p)$ is partitioned into subsets $E_i(p)$, $0 \leqslant i = O(\log n)$, where $E_i(p) := \{(p, q) \in E(p) : |pq| \in I_i\}$. By Theorem 2.1, this length-grouping step can be performed in $O(n \log n + m)$ time.

The angle-grouping step: Let $0 < \theta < \pi/4$, such that

$$\cos\theta - \sin\theta \geqslant 1 - \frac{\varepsilon}{2t(1+\varepsilon)}. \qquad (3.1)$$

Consider a collection $C_1, C_2, \ldots, C_h$ of simplicial cones of angular diameter θ, all having their apex at the origin, and that cover $\mathbb{R}^d$; see Yao [24]. In this second step, we further partition each subset $E_i(p)$, $p \in V$, $0 \leqslant i = O(\log n)$, into h subsets $E_{ij}(p)$, $1 \leqslant j \leqslant h$, where $E_{ij}(p) := \{(p, q) \in E_i(p) : q - p \in C_j\}$. It is clear that this step can be carried out in $O(n \log n + m)$ time.

At the end of this second step, we have partitioned each set $E(p)$ into $O(\log n)$ subsets. If q and r are any two points from the same subset, then $|pq| < (1+\varepsilon_0)|pr|$, $|pr| < (1+\varepsilon_0)|pq|$, and the angle between the line segments pq and pr is at most θ.

The preliminary pruning step: Let $G' = (V, E')$ be the subgraph of G that is obtained by picking exactly one (arbitrary) edge from each non-empty set $E_{ij}(p)$. It is clear that G' has $O(n \log n)$ edges and that it can be computed in $O(n \log n)$ time. To prove that G' is a $(1+\varepsilon)$-spanner of G, we use the following geometric lemma, which is similar to Lemma 2 in Arya and Smid [4]. Recall that $0 < \theta < \pi/4$.

LEMMA 3.2. *Let p, q and r be three points in $\mathbb{R}^d$ such that $p \neq q$, $p \neq r$, and the angle between pq and pr is at most θ. Let r' be the orthogonal projection of r onto the line through p and q, and assume that $|pr'| \leqslant |pq|$. Then $|rq| \leqslant |pq| - (\cos\theta - \sin\theta)|pr|$.*

LEMMA 3.3. *The graph G' is a $(1+\varepsilon)$-spanner of G.*

Proof. It is sufficient to prove that $\delta_{G'}(p, q) \leqslant (1+\varepsilon)|pq|$ for each edge (p, q) of the graph $G = (V, E)$. The complete proof, which is by induction on the lengths of the edges in E, will be given in the full paper. Here, we present one case of the proof.

Assume that (p, q) is not a shortest edge of E. Furthermore, assume that $\delta_{G'}(x, y) \leqslant (1+\varepsilon)|xy|$ for each edge (x, y) of E that is shorter than (p, q).

If (p, q) is an edge of E', then it is clear that $\delta_{G'}(p, q) \leqslant (1 + \varepsilon)|pq|$. So assume that (p, q) is not an edge of E'. Let r be a point of V such that (p, r) is an edge of E' that is in the same subset of the partition of p's adjacency list as (p, q). Recall that $|pq| < (1 + \varepsilon_0)|pr|$, $|pr| < (1 + \varepsilon_0)|pq|$, and the angle between pq and pr is at most θ. Let r' be the orthogonal projection of r onto the line through p and q. Similarly, let q' be the orthogonal projection of q onto the line through p and r.

Let us assume that $|pr'| \leqslant |pq|$. Applying the fact that G is a t-spanner for V, inequality (3.1) and Lemma 3.2, we obtain

$$\begin{aligned}
\delta_G(r, q) &\leqslant t|rq| \\
&\leqslant t|pq| - t(\cos\theta - \sin\theta)|pr| \\
&< t|pq| - \frac{t(\cos\theta - \sin\theta)}{1 + \varepsilon_0}|pq| \\
&\leqslant t|pq| - \frac{t\left(1 - \frac{\varepsilon}{2t(1+\varepsilon)}\right)}{1 + \varepsilon_0}|pq| \\
&\leqslant |pq|,
\end{aligned}$$

where the last inequality follows from our choice of ε_0. Hence, each edge on the shortest path in G between r and q has length less than $|pq|$. Therefore, by the induction hypothesis and the fact that G is a t-spanner for V, we have

$$\delta_{G'}(r, q) \leqslant (1 + \varepsilon)\delta_G(r, q) \leqslant t(1 + \varepsilon)|rq|.$$

Applying Lemma 3.2 and inequality (3.1), we get

$$\begin{aligned}
&\delta_{G'}(r, q) \\
&\quad\leqslant t(1 + \varepsilon)|pq| - t(1 + \varepsilon)(\cos\theta - \sin\theta)|pr| \\
&\quad\leqslant t(1 + \varepsilon)|pq| - t(1 + \varepsilon)\left(1 - \frac{\varepsilon}{2t(1 + \varepsilon)}\right)|pr|.
\end{aligned}$$

Hence,

$$\begin{aligned}
&\delta_{G'}(p, q) \\
&\quad\leqslant |pr| + \delta_{G'}(r, q) \\
&\quad\leqslant t(1 + \varepsilon)|pq| - \\
&\qquad\quad \left(t(1 + \varepsilon)\left(1 - \frac{\varepsilon}{2t(1 + \varepsilon)}\right) - 1\right)|pr| \\
&\quad= t(1 + \varepsilon)|pq| - (t(1 + \varepsilon) - \varepsilon/2 - 1)|pr| \\
&\quad< t(1 + \varepsilon)|pq| - \frac{t(1 + \varepsilon) - \varepsilon/2 - 1}{1 + \varepsilon_0}|pq| \\
&\quad\leqslant (1 + \varepsilon)|pq|,
\end{aligned}$$

where the last inequality follows from our choice for ε_0.

The final pruning step: The idea of this final step is to apply the general pruning algorithm of Section 3.1 to the graph $G' = (V, E')$. Since G' is a $((1 + \varepsilon)t)$-spanner for V, we have to take $s = 4(1 + (1 + \varepsilon)^2 t)/\varepsilon$ for the separation constant in the WSPD for V. The results of Section 3.1 imply that this yields a graph G'' having $O(n)$ edges and that is a $(1+\varepsilon)$-spanner of G'. Hence, G'' is a sparse $(1 + \varepsilon)^2$-spanner of our original t-spanner G. Since G' has $O(n \log n)$ edges, however, the running time is $O(n \log^2 n)$. We show below that we can use the special structure of the graph G' to improve the running time.

We start by computing the split tree T and the corresponding WSPD $\{A_f, B_f\}$, $1 \leqslant f \leqslant \ell$, for V, with separation constant $s = 4(1 + (1 + \varepsilon)^2 t)/\varepsilon$ and $\ell = O(n)$. The graph G'' that we want to compute contains for each f, $1 \leqslant f \leqslant \ell$, exactly one (arbitrary) edge (p, q) of E' with $p \in A_f$ and $q \in B_f$, provided such an edge exists. By Theorem 3.1, G'' is a $(1 + \varepsilon)$-spanner of G'. It remains to show how G'' can be computed efficiently.

Let f be an index with $1 \leqslant f \leqslant \ell$, and let p be a point of A_f. We first show how to decide if there is an edge (p, q) in E' with $q \in B_f$. We fix an arbitrary point b of B_f. By Lemma 3.1, we have $|pb|/(1 + 2/s) \leqslant |pq| \leqslant (1 + 2/s)|pb|$, for any $q \in B_f$. Hence, if i is the index such that $|pb| \in I_i = [(1 + \varepsilon_0)^i, (1 + \varepsilon_0)^{i+1})$, then there are a constant number of indices i' for which there is a point $q \in B_f$ with $|pq| \in I_{i'}$.

Let i' be such an index, and let $1 \leqslant j \leqslant h$. Each edge (p, q) of E' with $|pq| \in I_{i'}$ and $q - p \in C_j$ is contained in the subset $E_{i'j}(p)$. Since $E_{i'j}(p)$ contains at most one edge of E', it follows that the number of edges (p, q) in E' for which q may be in B_f that we have to consider is bounded by a constant. Moreover, if we know the value of the index i, then these candidate points q can be found in $O(1)$ time. Consider such a candidate point q. Let u be the node of T that represents the set B_f, and let v be the leaf of T that stores q. Then $q \in B_f$ if and only if v is in the subtree of u. Hence, we can decide in $O(1)$ time if $q \in B_f$.

This concludes the description of the four steps that compute a sparse $(1+\varepsilon)^2$-spanner G'' of the input t-spanner G. If we replace ε by $\varepsilon/3$ in the entire construction, then we obtain the following result.

THEOREM 3.2. *Let V be a set of n points in $\mathbb{R}^d$ such that all interpoint distances are in the interval $[1, n^k)$, where k is a positive integer constant. Given a geometric graph G with vertex set V and having m edges, which is a t-spanner for V, for some constant t, we can compute a $(1+\varepsilon)$-spanner of G having $O(n)$ edges, in $O(n \log n + m)$ time.*

4 Approximate shortest path queries

Let $G = (V, E)$ be a geometric graph that is a t-spanner for V, for some constant $t > 1$. We assume that the number of edges of G is $O(n)$. Let D be the length of the longest edge in G, let ε be a positive real constant, and let k be a positive integer constant. In this section, we will show how to preprocess G into a data structure such that for any two points p and q in V with $|pq| > D/n^k$, we can compute a $(1+\varepsilon)$-approximation to the shortest-path distance in G between p and q. This data structure will be based on the cluster-based implementation of the greedy spanner algorithm of Das and Narasimhan [12] and Gudmundsson *et al.* [15]. Therefore, we start by describing the main ideas of this algorithm. We may assume w.l.o.g. that $D = n^k$.

The greedy algorithm computes a $(1+\varepsilon)$-spanner G' of the t-spanner G. The basic version of this algorithm works as follows. The graph G' is initialized to $(V, \emptyset)$, and the edges of E are sorted according to their lengths. Then, the algorithm examines the edges of E one by one, in this sorted order. When the algorithm examines the edge (p, q), it decides 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 such a path does not exist. The final graph G' is a $(1+\varepsilon)$-spanner of G. To obtain a fast implementation, we choose real constants μ and λ with $1 \leqslant \lambda < \mu$, and partition E into $E_0 := \{(p,q) \in E : |pq| \leqslant \lambda\}$ and $E_i := \{(p,q) \in E : \mu^{i-1}\lambda < |pq| \leqslant \mu^i\lambda\}$, $1 \leqslant i = O(\log n)$.

After the graph G' has been initialized to (V, E_0), the remaining subsets E_i, $i \geqslant 1$, are processed one after another. At the beginning of the phase in which E_i is processed, the algorithm constructs a *cluster graph* H of the current graph G', which is dynamically maintained during this phase. Informally, this cluster graph H approximates the current graph G', in the sense that for any two points p and q for which $|pq|$ is approximately equal to $\mu^i\lambda$, $\delta_H(p,q)$ approximates $\delta_{G'}(p,q)$, which in turn 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. When the algorithm processes edge (p,q) of E_i, it uses the current cluster graph H in its decision 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'. The final graph G' is a $(1+\varepsilon)$-spanner of G, and the overall algorithm runs in $O(n \log n + \sum_{i \geqslant 1} |E_i|) = O(n \log n)$ time.

4.1 The cluster graph Let $L > 1$ be a real number, and consider the graph G' at the moment when the greedy algorithm has just processed all edges of E whose length is at most L. Das and Narasimhan prove that $\delta_{G'}(x,y) \leqslant (1+\varepsilon)|xy|$, for all edges $(x,y) \in E$ with $|xy| \leqslant L$. Let i be the integer such that $\mu^{i-1}\lambda < L \leqslant \mu^i\lambda$.

Das and Narasimhan, and Gudmundsson *et al.* choose an arbitrary real constant[1] α with $0 < \alpha < 1/4$, and use it to define the cluster graph H for G'. This cluster graph has the points of V as its vertices, and it has $O(n)$ edges. Moreover,

1. For all $p, q \in V$, $\delta_{G'}(p,q) \leqslant \delta_H(p,q)$.
2. For all $p, q \in V$ with $\delta_{G'}(p,q) > (1-2\alpha)\mu^{i-1}\lambda$,
$$\delta_H(p,q) \leqslant (1+\varepsilon)\frac{1+18\alpha}{1-2\alpha}\delta_{G'}(p,q).$$
3. For all $p, q \in V$ and any real number ℓ with $0 < \ell \leqslant (1+\varepsilon)\mu^i\lambda$, it takes $O(1)$ time to decide if $\delta_H(p,q) \leqslant \ell$.

[1] They use the notation δ instead of α.

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

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

- $\delta_G(p,q) \leqslant \delta_{G'}(p,q)$. This holds because G' is a subgraph of G.
- $\delta_{G'}(p,q) \leqslant \delta_H(p,q)$. This is one of the general properties of the cluster graph, see Section 4.1.
- $\delta_H(p,q) \leqslant (1+\varepsilon)((1+18\alpha)/(1-2\alpha))\delta_{G'}(p,q)$. Since $\delta_{G'}(p,q) \geqslant |pq| > \mu^{i-1}\lambda > (1-2\alpha)\mu^{i-1}\lambda$, this follows from the general properties of the cluster graph, see Section 4.1.
- $\delta_{G'}(p,q) \leqslant (1+\varepsilon)\delta_G(p,q)$. To prove this, first observe that $\delta_G(p,q) \leqslant t|pq| \leqslant \mu^i\lambda$. Hence, each edge on the shortest path in G between p and q has length at most $\mu^i\lambda$. The claim then follows from the general properties of the cluster graph, see Section 4.1.

Combining these inequalities, it follows that

$$\delta_G(p,q) \leqslant \delta_H(p,q) \leqslant (1+\varepsilon)^2\frac{1+18\alpha}{1-2\alpha}\delta_G(p,q),$$

provided that $\mu^{i-1}\lambda < |pq| \leqslant \mu^i\lambda/t$. Hence, by choosing the constant α such that $0 < \alpha < 1/4$ and $((1+18\alpha)/(1-2\alpha)) \leqslant 1+\varepsilon$, the value $\delta_H(p,q)$ (which can be computed in $O(1)$ time) is a $(1+\varepsilon)^3$-approximation to the length of the shortest path in G between p and q.

Hence, we have shown the following. For any constant λ with $1 \leqslant \lambda < \mu$, if p and q are two points with $|pq| > \lambda$, and if i is the positive integer such that $\mu^{i-1}\lambda < |pq| \leqslant \mu^i\lambda$, then we can compute a $(1+\varepsilon)$-approximation to $\delta_G(p,q)$ in $O(1)$ time, provided that (i) we know the index i and (ii) $|pq| \leqslant \mu^i\lambda/t$. We have seen in Section 2 how the index i can be computed. Condition (ii) says that the distance $|pq|$ must be in the "initial" part of the interval $(\mu^{i-1}\lambda, \mu^i\lambda]$. This condition can be met if we apply this construction using a constant number of values for λ. The details will be given in the following subsection.

4.3 The data structure We choose constants μ and α such that $\mu > t$, $0 < \alpha < 1/4$ and $((1+18\alpha)/(1-2\alpha)) \leqslant 1+\varepsilon$. For any j, $0 \leqslant j \leqslant \lfloor \log\mu/\log(\mu/t)\rfloor$, let $\lambda_j := (\mu/t)^j$. For each such j, we run the greedy spanner algorithm on the t-spanner G, using the subsets $E_0^j := \{(p,q) \in E : |pq| \leqslant \lambda_j\}$ and $E_i^j := \{(p,q) \in E : \mu^{i-1}\lambda_j < |pq| \leqslant \mu^i\lambda_j\}$, $1 \leqslant i \leqslant 1 + \lfloor k\log_\mu n - \log_\mu \lambda_j\rfloor$.

Our data structure consists of the collection of cluster graphs H_i^j, $0 \leqslant j \leqslant \lfloor \log\mu/\log(\mu/t)\rfloor$, $1 \leqslant i \leqslant 1 + \lfloor k\log_\mu n - \log_\mu \lambda_j\rfloor$, where H_i^j is the cluster graph at the moment when the greedy algorithm (using λ_j) has just processed all edges of the subset E_i^j.

Let us see how this data structure can be used to answer approximate shortest path queries. Let p and q be two points of V with $|pq| > 1$. Let i be the positive integer such that $\mu^{i-1} < |pq| \leqslant \mu^i$, and let j be the positive integer such that $\lambda_j < |pq|/\mu^{i-1} \leqslant (\mu/t)\lambda_j$. Then $\mu^{i-1}\lambda_j < |pq| \leqslant \mu^i\lambda_j/t$ and the results of Section 4.2 imply that $\Delta := \delta_{H_i^j}(p,q)$ can be computed in $O(1)$ time and satisfies $\delta_G(p,q) \leqslant \Delta \leqslant (1+\varepsilon)^2\delta_G(p,q)$. By replacing ε by $\varepsilon/3$ in the entire construction, we get a $(1+\varepsilon)$-approximation to $\delta_G(p,q)$.

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

In the full paper we show that, by some general properties of cluster graphs, a $(1+\varepsilon)$-approximate shortest path can be computed in time proportional to the number of its edges.

5 Further applications

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

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

Arikati *et al.* [2] have shown how to compute, in $O(n \log n)$ time, a $(1+\varepsilon)$-spanner G' of G, for any given constant $\varepsilon > 0$. Let p and q be two points of S, 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.

Clarkson [10] and Chen [7] 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. Hence, a $(1+\varepsilon)^3$-approximation of the shortest obstacle-avoiding path can be computed in $O(\log n)$ time.

5.2 Closest pair queries Given a geometric graph $G = (V, E)$ on n points and m 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 may assume that $m = O(n)$. 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.

Consider a WSPD $\{A_i, B_i\}$, $1 \leqslant i \leqslant \ell$, where $\ell = O(n)$, for the set S, with separation constant $s > 2t$. For each i for which both A_i and B_i are singleton sets, let Δ_i be a $(1+\varepsilon)$-approximation to $\delta_G(a_i, b_i)$, where a_i and b_i are the only points of A_i and B_i, respectively. Let i be an index for which Δ_i is minimum. Then the points $x := a_i$ and $y := b_i$ form a $(1+\varepsilon)$-approximate closest pair in S. (The proof follows from Lemma 3.1.) 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.

Next, we consider 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.

Let $\{A_i, B_i\}$, $1 \leqslant i \leqslant \ell$, be a WSPD for the set $X \cup Y$, where $\ell = O(n)$, with separation constant s, where $s > 2t$ and $s \geqslant 4t/\epsilon$. For each i 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 $\delta_G(a_i, b_i)$. Let i be an index for which Δ_i is minimum. Then the points $x := a_i$ and $y := b_i$ form a $(1+\varepsilon)$-approximate bichromatic closest pair in S. 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.

5.3 Computing stretch factors of a geometric graph

Given a geometric graph $G = (V, E)$ on n points and m edges, it was shown in [20] that the problem of computing a $(1+\varepsilon)$-approximation to the stretch factor t of G, can be reduced to the problem of computing $O(n)$ approximate shortest path queries, after computing a well-separated decomposition of V.

Assume that the stretch factor t is bounded from above by a known constant. Using our result, we can answer the approximate shortest path queries in $O(n)$ time after $O(m + n \log n)$ preprocessing, giving a total computation time of $O(m + n \log n)$.

6 Conclusions

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

References

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

[2] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *4th ESA*, LNCS, Vol. 1136, pp. 514–528, 1996.

[3] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *35th FOCS*, 703–712, 1994.

[4] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17:33–54, 1997.

[5] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *4th LATIN*, LNCS, Vol. 1776, pp. 88–94, 2000.

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

[7] D. Z. Chen. On the all-pairs Euclidean short path problem. In *6th SODA*, pp. 292–301, 1995.

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

[9] Y.-J. Chiang and J. S. B. Mitchell. Two-point euclidean shortest path queries in the plane. In *10th SODA*, 1999.

[10] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *19th STOC*, pp. 56–65, 1987.

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

[12] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *IJCGA*, 7:297–315, 1997.

[13] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *13th SoCG*, 1997.

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

[15] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. In *7th SWAT*, LNCS, Vol. 1851, pp. 314–327, 2000.

[16] D. Gusfield. Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge, UK, 1997.

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

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

[19] J. S. B. Mitchell. Shortest paths and networks. In *Handbook of Discrete and Computational Geometry*, pp. 445–466. CRC Press LLC, 1997.

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

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

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

[23] M. Thorup and U. Zwick. Approximate distance oracles. In *33rd STOC*, 2001.

[24] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11, 1982.