Dynamic algorithms for geometric spanners of small diameter: randomized solutions

Sunil Arya   David M. Mount   Michiel Smid

MPI–I–94–156   October 1994

Dynamic algorithms for geometric spanners of small diameter: randomized solutions

Sunil Arya   David M. Mount   Michiel Smid

MPI–I–94–156   October 1994

Dynamic algorithms for geometric spanners of small diameter: randomized solutions*

Sunil Arya[‡] David M. Mount[†] Michiel Smid[‡]

October 19, 1994

Abstract

Let S be a set of n points in $\mathbb{R}^d$ and let $t > 1$ be a real number. A t-spanner for S is a directed graph having the points of S as its vertices, such that for any pair p and q of points there is a path from p to q of length at most t times the Euclidean distance between p and q. Such a path is called a t-spanner path. The spanner diameter of such a spanner is defined as the smallest integer D such that for any pair p and q of points there is a t-spanner path from p to q containing at most D edges.

A randomized algorithm is given for constructing a t-spanner that, with high probability, contains $O(n)$ edges and has spanner diameter $O(\log n)$. A data structure of size $O(n \log^d n)$ is given that maintains this t-spanner in $O(\log^d n \log\log n)$ expected amortized time per insertion and deletion, in the model of random updates, as introduced by Mulmuley.

Previously, no results were known for spanners with low spanner diameter and for maintaining spanners under insertions and deletions.

1 Introduction

Given a set S of n points in $\mathbb{R}^d$, where d is a constant, and a real number $t > 1$, a *t-spanner* for S is a directed graph on S such that for each pair p and q of points of S there is a path from p to q having length at most t times the Euclidean distance between p and q. We call such a path a *t-spanner path.*

The problem of constructing t-spanners has received great attention. For the planar case, i.e., $d = 2$, Keil and Gutwin [7] introduce the so-called *Θ-graph.* They prove that

*Some of the results of this paper will appear in Proc. 35th Annu. IEEE Symp. on Foundations of Computer Science (FOCS), 1994, under the title *Randomized and deterministic algorithms for geometric spanners of small diameter.*

[†]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. E-mail: mount@cs.umd.edu. This author was partially supported by the National Science Foundation under grant CCR-9310705.

[‡]Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. E-mail: {arya,michiel}@mpi-sb.mpg.de. These authors were supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

for an appropriate choice of θ this graph is a t-spanner with $O(n)$ edges, and they give an $O(n \log n)$ time algorithm to construct it. Ruppert and Seidel [13] generalize this result to any fixed dimension d. They show that the d-dimensional Θ-graph is a t-spanner having $O(n)$ edges and that it can be constructed in $O(n \log^{d-1} n)$ time.

In Callahan and Kosaraju [3], Salowe [14] and Vaidya [15], optimal algorithms are given for constructing t-spanners: For any set S of n points in $\mathbb{R}^d$, $d \geq 2$, and for any $t > 1$, a t-spanner for S having $O(n)$ edges can be constructed in $O(n \log n)$ time.

There are several interesting quantities related to a t-spanner. First, it is clear that any spanner must have at least $n-1$ edges. All spanners referred to above have $O(n)$ edges, which is optimal. Second, the total length of all edges in a spanner is always at least equal to the length of a minimum spanning tree for S. We denote the latter by $wt(MST)$. Das and Narasimhan [5] give an $O(n \log^2 n)$ time algorithm for constructing a t-spanner with $O(n)$ edges. Combining their results with those in Das, Narasimhan and Salowe [6] shows that the total length of the edges of this spanner is bounded by $O(wt(MST))$. This result holds for any fixed dimension d.

For constructing bounded degree spanners, the best result is by Arya and Smid [2]: They give an $O(n \log^d n)$ time algorithm that builds a t-spanner such that each point has a degree that is bounded by a constant. In fact, a variant of their algorithm, combined with results of [5, 6], produces a bounded degree t-spanner such that the total length of all edges is bounded by $O(wt(MST))$. This variant also has running time $O(n \log^d n)$.

All spanners referred to above have a disadvantage in comparison with the complete Euclidean graph. Although the Euclidean lengths of t-spanner paths are within a constant factor of the Euclidean distance between points, the number of edges in these paths may generally be as large as $\Omega(n)$. The resulting inefficiency of computing spanner paths, storing them, and traversing them is a significant limitation in their usefulness.

In this paper, we consider the problem of constructing t-spanners with $O(n)$ edges and small *spanner diameter*. That is, for each pair p and q of points there must be a t-spanner path from p to q consisting of only a small number of edges. Moreover, it should be possible to compute such a t-spanner path efficiently. To our knowledge, this natural problem has not been considered before.

A second disadvantage of the known spanners is that they are static. That is, no efficient algorithms are known for maintaining a t-spanner when points are inserted and/or deleted in the set S. In the second part of this paper, we consider the problem of designing *dynamic* data structures for maintaining a t-spanner.

1.1 Summary of results

Intuitively, our results may be viewed as one way of generalizing skip lists to higher dimensions. Assume that the points of S are one-dimensional. Consider a *skip list* [12] for the points of S. By "flattening" the nodes of the skip list down to the lowest level, we can regard this data structure as a directed graph on S. This graph has an expected number of $O(n)$ edges. For each pair p and q of points, there is a path from p to q having length $|p-q|$ and containing an expected number of $O(\log n)$ edges. In fact, even the expected maximum number of edges on any such path is bounded by

$O(\log n)$. (See [10].) As a result, the skip list is a 1-spanner with expected spanner diameter $O(\log n)$. This spanner can be maintained in $O(\log n)$ expected time per insertion and deletion.

In this paper, we generalize this idea to the d-dimensional case, for any fixed d, by combining the Θ-graph of [7, 13] with skip lists. For any fixed $t > 1$, we get a t-spanner, which we call a *skip list spanner*.

We will show that the skip list spanner has an expected number of $O(n)$ edges, and its expected spanner diameter is bounded by $O(\log n)$. Also, we will show that the expected maximum time to construct a t-spanner path from any point of S to any other point of S is bounded by $O(\log n)$. These bounds even hold with high probability. Note that hence the *existence* of a t-spanner having $O(n)$ edges and $O(\log n)$ spanner diameter has been proven.

For a standard skip list it is relatively easy to show that the expected spanner diameter is bounded by $O(\log n)$. (See [10, 12].) For d-dimensional skip list spanners, however, the proof turns out to be more difficult.

Using *range trees* [8, 11], we can construct the skip list spanner in $O(n \log^{d-1} n)$ expected time and $O(n \log^{d-2} n)$ space.

We are not able to give efficient algorithms for maintaining the skip list spanner under arbitrary insertions and deletions. In the model of *random updates*, as introduced by Mulmuley [10], we do get an algorithm that is fast in the expected sense: Again using range trees, we design a data structure of size $O(n \log^d n)$ that maintains the skip list spanner in $O(\log^d n \log\log n)$ expected amortized time per random insertion and deletion.

The skip list spanner is a *randomized* data structure. In [1], *deterministic* algorithms are given for constructing t-spanners having $O(n)$ edges and $O(\log n)$ spanner diameter. At present, however, no efficient algorithms are known to update these deterministic spanners.

The rest of this paper is organized as follows. In Section 2, we give the basic definitions, introduce the Θ-graph, prove some basic results about it, and show how to construct this graph efficiently. Our construction uses a logarithmic factor less space than that of [13]. In Section 3, we define the skip list spanner, give the algorithm to construct a t-spanner path from any point to any other point, and prove that the expected running time and the expected spanner diameter are both bounded by $O(\log n)$. Section 4 considers the problem of maintaining the skip list spanner in the model of random insertions and deletions. Finally, in Section 5, we give some concluding remarks.

2 Spanners, simplicial cones and the Θ-graph

Let S be a set of n points in $\mathbb{R}^d$. We will consider graphs having the points of S as their vertices. For convenience, we assume that all graphs are directed. The *weight* of an edge (p, q) is defined as the Euclidean distance between p and q. The *weight of a path* in a graph is defined as the sum of the weights of all edges on the path. If (p, q) is an edge, then p is called its *source* and q is called its *sink*.

Let $t > 1$. A graph $G = (S, E)$ is called a *t-spanner* for S if for any pair p and q of points of S there is a path in G from p to q having weight at most t times the Euclidean distance between p and q. Any path satisfying this condition is called a *t-spanner path* from p to q. Given a t-spanner for S, we define a *path query* to be a pair (p, q) of points in S. The answer to a path query is a t-spanner path from p to q. An *augmented spanner* is a spanner together with an associated data structure for answering path queries and/or supporting updates.

It is not a restriction to consider only directed graphs. Any directed t-spanner can be converted into an undirected t-spanner by making the edges undirected. Similarly, given an undirected t-spanner, we get a directed t-spanner by replacing each undirected edge $\{p, q\}$ by a pair (p, q) and (q, p) of directed edges.

Since any t-spanner must be connected, it must have at least $n-1$ edges. Therefore, the goal is to construct t-spanners having $O(n)$ edges, where the constant factor only depends on t and d.

The *spanner diameter* of a t-spanner is defined as the smallest number D such that for any pair p and q of points there is a t-spanner path from p to q containing at most D edges. In this paper, we want to construct spanners with a low spanner diameter.

The Euclidean distance between the points p and q in $\mathbb{R}^d$ is denoted by $|pq|$. The following lemma is useful for showing that a graph is a t-spanner.

Lemma 1 *Let $t > 1$ be any real number and let S be a set of points in $\mathbb{R}^d$. Let $G = (S, E)$ be any directed graph such that the following holds. For any two distinct points p and q of S, there is a point r in S such that*

1. *(p, r) is an edge of E, and*
2. *$|rq| \leq |pq| - \frac{1}{t}|pr|$.*

Then the graph G is a t-spanner for S.

Proof: The proof is by induction on the rank of the interpoint distance. Let p, q be a closest pair of S. We claim that (p, q) is an edge of G. If this is true, then clearly there is a t-spanner path from p to q. Assume that (p, q) is not an edge. Let r be a point of S such that 1. and 2. hold. Note that $r \neq p$. Therefore, 2. implies that $|rq| < |pq|$, which is a contradiction.

Let p, q be a pair of points of S and assume that there are t-spanner paths between any pair of points that are at distance less than $|pq|$. If (p, q) is an edge of G, then there is a t-spanner path from p to q. Assume that (p, q) is not an edge. Let r be a point of S such that 1. and 2. hold. Then $r \neq p$, $r \neq q$ and, by 2., $|rq| < |pq|$. Hence, by the induction hypothesis, there is a t-spanner path from r to q. Consider the path that starts in p, takes the edge to r and then follows the t-spanner path from r to q. This path has weight at most $|pr| + t|rq|$. It follows immediately from 2. that this expression is at most equal to $t|pq|$. Hence, there is a t-spanner path from p to q. ∎

Let p and q be points in $\mathbb{R}^d$, both not equal to the origin 0, and let H be the two-dimensional plane that contains p, q and 0. (If $p = q$, then we take for H any plane that contains p and 0.) The vectors $\overrightarrow{0p}$ and $\overrightarrow{0q}$ are both contained in H. The *angle* between these vectors, which is a real number in the interval $[0 : \pi]$, is denoted by $angle(p, q)$.

We introduce the notion of cones. A *(simplicial) cone* is the intersection of d halfspaces in $\mathbb{R}^d$. The hyperplanes that bound these halfspaces are assumed to be in general position, in the sense that their intersection is a point, called the *apex* of the cone. In the plane, a cone having its apex at the point p is a wedge bounded by two rays emanating from p that make an angle at most equal to π.

Let C be any cone in $\mathbb{R}^d$ having its apex at the point p. The *angular diameter* of C is defined as the maximum value of $angle(q - p, r - p)$, where q and r range over all points of $C \cap \mathbb{R}^d$. For $d = 2$, this is exactly the angle between the two rays that form the boundary of C.

Let θ be a fixed real number such that $0 < \theta \leq \pi$. Let $\mathcal{C}$ be a collection of cones such that

1. each cone has its apex at the origin,
2. each cone has angular diameter at most θ,
3. all cones cover $\mathbb{R}^d$.

In Yao [16], it is shown how such a collection $\mathcal{C}$, consisting of $O((c/\theta)^{d-1})$ cones for a suitable constant c, can be obtained. In the plane and for $\theta = 2\pi/k$, we just rotate the positive x-axis over angles $i \cdot \theta$, $0 \leq i < k$. This gives k rays. The wedge between two successive rays defines a cone of $\mathcal{C}$.

For each cone $C \in \mathcal{C}$, let l_C be a fixed ray that emanates from the origin and that is contained in C.

Let C be any cone of $\mathcal{C}$ and let p be any point in $\mathbb{R}^d$. We define $C_p := C + p := \{x + p : x \in C\}$, i.e., C_p is the cone obtained by translating C such that its apex is at p. Similarly, we define $l_{C,p} := l_C + p$. Hence, $l_{C,p}$ is a ray that emanates from p and that is contained in the translated cone C_p.

Now we can introduce Θ-graphs. Keil and Gutwin [7] defined these for the case $d = 2$. Ruppert and Seidel [13] defined them for arbitrary dimensions $d \geq 2$.

Definition 1 (**[7, 13]**) Let $k \geq 2$ be an integer and let $\theta = 2\pi/k$. Let S be a set of points in $\mathbb{R}^d$. The directed graph $\Theta(S, k)$ is defined as follows.

1. The vertices of $\Theta(S, k)$ are the points of S.
2. For each point p of S and each cone C of $\mathcal{C}$ such that the translated cone C_p contains points of $S \setminus \{p\}$, there is an edge from p to the point r in $C_p \cap S \setminus \{p\}$ whose orthogonal projection onto $l_{C,p}$ is closest to p. (If there are several such points r then we take an arbitrary one.)

See Figure 1 for an illustration in the planar case.

Remark 1 If we take for r the point in $C_p \cap S \setminus \{p\}$ that is closest to p, then we get Yao's geographic neighbor graph [16]. Defining the edges as in Definition 1 has the advantage that the corresponding graph can be constructed in $O(n \log^{O(1)} n)$ time. For dimensions $d > 2$, it is not known if the geographic neighbor graph can be constructed within this time bound.

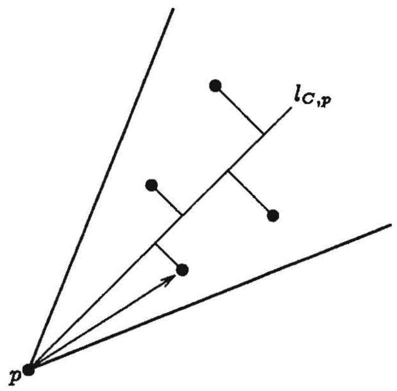

Figure 1: Illustration of the graph $\Theta(S, k)$ for $d = 2$.

We now prove that for $k > 8$, the graph $\Theta(S, k)$ is a spanner. We remark that this was proved already in [7] for the case where $d = 2$ and in [13] for the case where $d \geq 2$. Our proof is different from that of [7], but similar to the one in [13]. We give the proof here, in order to be self-contained. First, we prove a geometric lemma, which holds for $k \geq 8$.

Lemma 2 *Let $k \geq 8$ be an integer, let $\theta = 2\pi/k$, let p and q be any two distinct points in $\mathbb{R}^d$, and let C be the cone of $\mathcal{C}$ such that $q \in C_p$. Let r be any point in $\mathbb{R}^d \cap C_p$ such that the projection of r onto the ray $l_{C,p}$ is at least as close to p as the projection of q onto $l_{C,p}$. Then*

1. $|pr| \cos\theta \leq |pq|$, *and*
2. $|rq| \leq |pq| - (\cos\theta - \sin\theta)|pr|$.

Proof: If $r = q$, then 1. and 2. both hold. Therefore, assume in the rest of the proof that $r \neq q$. Let H be the two-dimensional plane that contains the ray $l_{C,p}$ and the point q. Let q' be the projection of q onto $l_{C,p}$ and let α be the angle between the vector $\overrightarrow{pq}$ and the ray $l_{C,p}$. Note that $0 \leq \alpha \leq \theta$. Then $|pq'| = |pq| \cos\alpha \leq |pq|$. Similarly, let H' be the two-dimensional plane that contains $l_{C,p}$ and r, let r' be the projection of r onto $l_{C,p}$, and let β be the angle between $\overrightarrow{pr}$ and $l_{C,p}$. Then $|pr'| = |pr| \cos\beta \geq |pr| \cos\theta$.

By our assumption, we have $|pr'| \leq |pq'|$. Therefore, $|pr| \cos\theta \leq |pr'| \leq |pq'| \leq |pq|$, which proves 1.

To prove 2., let H'' be the two-dimensional plane that contains the points p, q and r. Let l be the line through p and q, and let s be the projection of r onto l. Finally, let γ be the angle between the vectors $\overrightarrow{pq}$ and $\overrightarrow{pr}$. Note that $0 \leq \gamma \leq \theta$. We distinguish two cases depending on whether $|ps| \leq |pq|$ or $|ps| > |pq|$. (See Figure 2.)

Case 1: $|ps| \leq |pq|$.

We have $|rs| = |pr| \sin\gamma \leq |pr| \sin\theta$ and $|ps| = |pr| \cos\gamma \geq |pr| \cos\theta$. Applying the triangle inequality gives

$$|rq| \leq |rs| + |sq|$$

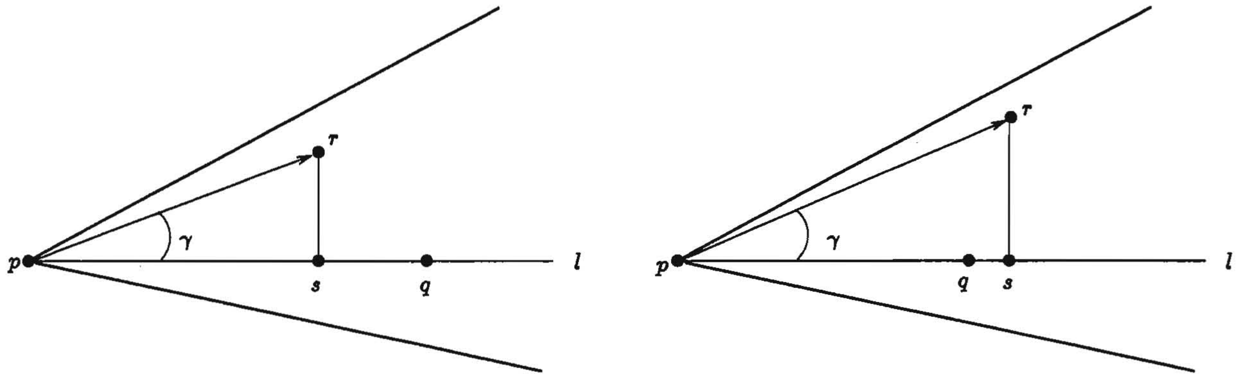

Figure 2: Cases 1 and 2.

$$
\begin{aligned}
&= |rs| + |pq| - |ps| \\
&\leq |pr| \sin\theta + |pq| - |pr| \cos\theta \\
&= |pq| - (\cos\theta - \sin\theta)|pr|.
\end{aligned}
$$

Case 2: $|ps| > |pq|$.

We have $|rs| = |pr| \sin\gamma$ and $|ps| = |pr| \cos\gamma$. Applying the triangle inequality and using the fact that the function $\sin x + \cos x$ is non-decreasing for $0 \leq x \leq \pi/4$ gives

$$
\begin{aligned}
|rq| &\leq |rs| + |sq| \\
&= |rs| + |ps| - |pq| \\
&= |pr|(\sin\gamma + \cos\gamma) - |pq| \\
&\leq |pr|(\sin\theta + \cos\theta) - |pq|.
\end{aligned}
$$

By 1., we have $|pr| \cos\theta \leq |pq|$, which is equivalent to the inequality

$$|pr|(\sin\theta + \cos\theta) - |pq| \leq |pq| - (\cos\theta - \sin\theta)|pr|.$$

This proves that $|rq| \leq |pq| - (\cos\theta - \sin\theta)|pr|$. ∎

Corollary 1 ([7, 13]) *Let $k > 8$ be an integer, let $\theta = 2\pi/k$ and let S be a set of points in $\mathbb{R}^d$. The graph $\Theta(S, k)$ is a t-spanner for $t = 1/(\cos\theta - \sin\theta)$. It contains $O((c/\theta)^{d-1} n)$ edges, for some constant c.*

Proof: Since $k > 8$, we have $0 < \cos\theta - \sin\theta < 1$. Therefore, $t > 1$. Lemma 2 implies that the conditions of Lemma 1 are satisfied. Hence, $\Theta(S, k)$ is a t-spanner. It follows from the definition of $\Theta(S, k)$ that each point of S has out-degree at most the number of cones in $\mathcal{C}$. This proves that the total number of edges is bounded by $O((c/\theta)^{d-1} n)$. ∎

Remark 2 For any $t > 1$, there is a $k > 8$ such that $1 < 1/(\cos\theta - \sin\theta) \leq t$ holds for $\theta = 2\pi/k$. Hence, for each $t > 1$, we get a t-spanner containing $O(n)$ edges.

Next, we consider the problem of constructing the graph $\Theta(S,k)$. In [7], it is shown how this problem can be solved in $O(n \log n)$ time using $O(n)$ space for the case where $d = 2$. In [13], an algorithm is given that contructs the graph $\Theta(S,k)$ in $O(n \log^{d-1} n)$ time using $O(n \log^{d-1} n)$ space, for any fixed dimension $d \geq 2$. We change the latter solution slightly, resulting in an algorithm having the same running time but using only $O(n \log^{d-2} n)$ space.

Before we can give the algorithm, we need to introduce some notation. Let C be any cone of $\mathcal{C}$. Recall that C is the intersection of d halfspaces. Let $h_1, h_2, \ldots, h_d$ be the hyperplanes that bound these halfspaces, and let $H_1, H_2, \ldots, H_d$ be lines through the origin such that H_i is orthogonal to h_i, $1 \leq i \leq d$. We give the line H_i a direction such that the cone C is "above" h_i. Let L be the line that contains the ray l_C. We give L the same direction as l_C. (See Figure 3 for an illustration in the planar case.)

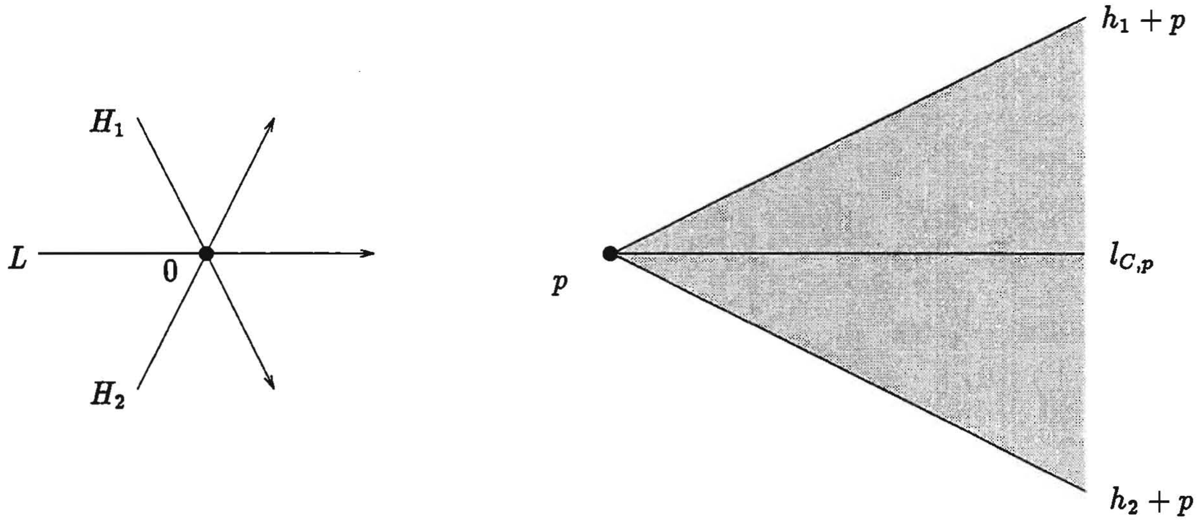

Figure 3: The directed lines H_1, H_2 and L, and the translated cone C_p.

Let p be any point in $\mathbb{R}^d$. We write the coordinates of p with respect to the standard coordinate axes as $p_1, p_2, \ldots, p_d$. For $1 \leq i \leq d$, we denote by p'_i the signed Euclidean distance between the origin and the orthogonal projection of p onto H_i, where the sign is positive or negative according to whether this projection is to the "right" or "left" of the origin. Similarly, p'_{d+1} denotes the signed Euclidean distance between the origin and the orthogonal projection of p onto L.

In this way, we can write the cone C as $C = \{x \in \mathbb{R}^d : x'_i \geq 0, 1 \leq i \leq d\}$. For $p \in \mathbb{R}^d$, we can write the translated cone C_p with apex p as

$$C_p = \{x \in \mathbb{R}^d : x'_i \geq p'_i, 1 \leq i \leq d\}.$$

We define $-C_p := -C + p := \{-x + p : x \in C\}$. Then we have

$$-C_p = \{x \in \mathbb{R}^d : x'_i \leq p'_i, 1 \leq i \leq d\}.$$

Let p be a point of S. Computing the edge of $\Theta(S,k)$ with source p and sink in the cone C_p is equivalent to finding among all points $q \in S \setminus \{p\}$ such that $q'_i \geq p'_i$ for all $1 \leq i \leq d$, a point with minimal q'_{d+1}-coordinate.

We define a d-layer data structure having the form of a range tree [8, 11] that will be used to construct the graph $\Theta(S, k)$. This data structure depends on the cone C.

There is a balanced binary search tree storing the points of S in its leaves, sorted by their q_1'-coordinates. (Points with equal q_1'-coordinates are stored in lexicographical order.) For each node v of this tree, let S_v be the subset of S that is stored in the subtree of v. Then v contains a pointer to the root of a balanced binary search tree storing the points of S_v in its leaves, sorted by their q_2'-coordinates. (Points with equal q_2'-coordinates are stored such that the points $(q_2', \dots, q_d')$ are in lexicographical order.) Each node w of this tree contains a pointer to the root of a balanced binary search tree storing the points of w's subtree in its leaves, sorted by their q_3'-coordinates, etc. At the d-th layer, there is a balanced binary search tree storing a subset of S in its leaves, sorted by their q_d'-coordinates. The binary tree that stores points sorted by their q_i'-coordinates is called a *layer-i tree.*

With each node u of any layer-d tree, we store the following additional information. Consider the subset of S that is stored in the subtree of u. We store with u the point of this subset whose q_{d+1}'-coordinate is minimal. (If there are several such points then we store only one of them.)

Given this data structure, we can compute the edges (p, q) of $\Theta(S, k)$ such that $q \in C_p$:

Consider any point $p \in \mathbb{R}^d$. We compute a set of $O(\log^d n)$ canonical nodes of layer-d trees, such that all subsets stored in the subtrees of these nodes partition the set of all points of $S \setminus \{p\}$ that are contained in the cone C_p. With each of these nodes u, we have stored a point q_u such that $q_{u,d+1}'$ is minimal in the subtree of u. Let q be a point such that $q_{u,d+1}'$ is minimal over all canonical nodes u. Then (p, q) is an edge in $\Theta(S, k)$.

The following lemma gives the complexity of the data structure. The proof is exactly the same as that for a standard range tree. For details, we refer the reader to Lueker [8]. We remark that the additional information—the minimal q_{d+1}'-coordinates stored in the nodes of the layer-d trees—can be computed in $O(n \log^{d-1} n)$ time by a bottom-up procedure.

Lemma 3 *Let S be a set of n points in $\mathbb{R}^d$, and let C be a cone of $\mathcal{C}$. The above d-layered data structure has size $O(n \log^{d-1} n)$ and can be built in $O(n \log^{d-1} n)$ time. We can maintain this data structure in $O(\log^d n)$ amortized time per insertion and deletion. Given any point $p \in \mathbb{R}^d$, we can compute in $O(\log^d n)$ time a point q in $C_p \cap S \setminus \{p\}$ for which q_{d+1}' is minimal, or determine that such a point does not exist.*

Hence, we can construct the graph $\Theta(S, k)$ in $O(n \log^d n)$ time by building the above data structure for each cone C separately and by querying it with each point of S. We can save a factor of $\log n$ by observing that all query points are known in advance; these are precisely the points of S. Again, we consider each cone C separately. We sort the points of S by their p_1'-coordinates. Then we sweep over them in decreasing order. All visited points are maintained in the data structure of Lemma 3, by taking only the final d coordinates $p_2', \dots, p_{d+1}'$ into account. (That is, we apply Lemma 3 for dimension $d - 1$.)

If the sweep line encounters a new point p, then we query the data structure and find a point q such that $q'_i \geq p'_i$ for all $2 \leq i \leq d$, and for which q'_{d+1} is minimal. Since at this moment, the data structure contains exactly all points r of S having a first coordinate r'_1 which is at least equal to p'_1, we know that q is in fact a point of S such that $q'_i \geq p'_i$ for all $1 \leq i \leq d$, and for which q'_{d+1} is minimal. Hence, (p, q) is an edge of $\Theta(S, k)$. We now insert the point $(p'_2, \ldots, p'_{d+1})$ into the data structure and the sweep line moves to the next point of S.

It is clear that this algorithm correctly constructs the graph $\Theta(S, k)$. We summarize our result:

Theorem 1 *Let $k > 8$ be an integer, let $\theta = 2\pi/k$, and let S be a set of n points in $\mathbb{R}^d$. The graph $\Theta(S, k)$ is a t-spanner for $t = 1/(\cos\theta - \sin\theta)$. It contains $O((c/\theta)^{d-1}n)$ edges, for some constant c. Using $O((c/\theta)^{d-1}n + n\log^{d-2} n)$ space, this graph can be constructed in time $O((c/\theta)^{d-1}n\log^{d-1} n)$.*

3 The skip list spanner

We have seen that the graph $\Theta(S, k)$ is a t-spanner for $t = 1/(\cos\theta - \sin\theta)$. Suppose that all points of S lie on a line. Then, $\Theta(S, k)$ can be seen as a list containing the points of S in the order in which they occur on this line. Clearly, this graph has spanner diameter $n - 1$.

In this section, we construct a t-spanner whose spanner diameter is bounded by $O(\log n)$ with high probability. The basic idea is to generalize skip lists [12].

Let S be a set of n points in $\mathbb{R}^d$. We construct a sequence of subsets, as follows: Let $S_1 = S$. Let $i \geq 1$ and assume that we already have constructed the subset S_i. For each point of S_i, we flip a fair coin. (All coin flips are independent.) The set S_{i+1} is defined as the set of all points of S_i whose coin flip produced heads. The construction stops if $S_{i+1} = \emptyset$. Let h denote the number of iterations of this construction. Then we have sets

$$\emptyset = S_{h+1} \subseteq S_h \subseteq S_{h-1} \subseteq S_{h-2} \subseteq \ldots \subseteq S_2 \subseteq S_1 = S.$$

The following lemma is well known. (For a proof, see e.g. [10].)

Lemma 4 *Consider the sets S_i, $1 \leq i \leq h$, produced by the given random process. We have*

1. *$h = O(\log n)$ with high probability. In particular, $E(h) = O(\log n)$.*
2. *$E(|S_i|) = n/2^{i-1}$, $1 \leq i \leq h$.*
3. *$\sum_{i=1}^{h} |S_i| = O(n)$ with high probability. In particular, $E(\sum_{i=1}^{h} |S_i|) = O(n)$.*

Definition 2 Let $k \geq 2$ be an integer and let $\theta = 2\pi/k$. Let S be a set of n points in $\mathbb{R}^d$. Consider the subsets S_i, $1 \leq i \leq h$, that are constructed by the given coin flipping process. The *skip list spanner*, $SLS(S, k)$, for S is defined as follows.

1. For each $1 \leq i \leq h$, there is a list L_i storing the points of S_i (in no particular order). We say that the points of S_i are at *level i* of the data structure.

2. For each $1 \leq i \leq h$, there is a graph $\Theta(S_i, k)$.
3. For each $1 \leq i \leq h$, there is a reversed graph $\Theta'(S_i, k)$, which is obtained from $\Theta(S_i, k)$ by reversing the direction of each edge.
4. For each $1 < i \leq h$ and each $p \in S_i$, the occurrence of p in L_i contains a pointer to its occurrence in L_{i-1}.
5. For each $1 \leq i < h$ and each $p \in S_{i+1}$, the occurrence of p in L_i contains a pointer to its occurrence in L_{i+1}.

Note that if all points of S lie on a line, we get a standard skip list. We will regard $SLS(S, k)$ as a directed graph with vertex set S and edge set the union of the edge sets of the graphs $\Theta(S_i, k)$ and $\Theta'(S_i, k)$.

Lemma 5 *Let $k > 8$ be an integer, let $\theta = 2\pi/k$ and let S be a set of n points in $\mathbb{R}^d$. The skip list spanner $SLS(S, k)$ is a t-spanner for $t = 1/(\cos\theta - \sin\theta)$. It contains $O((c/\theta)^{d-1}n)$ edges, for some constant c. Also with high probability, this graph can be constructed in time $O((c/\theta)^{d-1}n \log^{d-1} n)$ using $O((c/\theta)^{d-1}n + n\log^{d-2} n)$ space.*

Proof: The skip list spanner contains $\Theta(S, k)$, which, by Theorem 1, is a t-spanner. Therefore, $SLS(S, k)$ is also a t-spanner. Also, by Theorem 1, the number of edges of $SLS(S, k)$ is bounded by $O(\sum_{i=1}^{h}(c/\theta)^{d-1}|S_i|)$. Lemma 4 implies that, with high probability, this summation is bounded by $O((c/\theta)^{d-1}n)$. If the number of edges is larger, then we repeat the construction until $SLS(S, k)$ contains $O((c/\theta)^{d-1}n)$ edges. The bounds on the space requirement and the construction time follow in a similar way. ∎

Now we give the algorithm for solving path queries. That is, given points p and q of S, we show how to construct a t-spanner path from p to q. Of course, we can construct such a path by using only edges of $\Theta(S, k)$. In order to reduce the number of edges on the path, however, we do the following.

We start in the occurrence of p at level one of the skip list spanner and construct a path from p towards q. Suppose we have already constructed a path from p to x. If $x = q$, then we have reached our destination. Assume that $x \neq q$. We check if x occurs at level two. Assume this is not the case. Then we extend the path as follows. Let C be the cone of $\mathcal{C}$ such that $q \in C_x$. Let x' be the point of $C_x \cap S_1$ such that (x, x') is an edge of $\Theta(S_1, k$ [illegible] $'$ is the next p[illegible] the path from p towards q, i.e., we set $x := x'$. We keep on growing this path until $x = q$ or the point x occurs at level two of the skip list spanner. If x occurs at level two, we start growing a path from q towards x. Suppose we have already constructed a path from q to y. We stop growing this path if y is equal to one of the points on the path from p to x, or y occurs at level two. If y is equal to the point, say, p' on the path from p to x, then we report the path in $\Theta(S_1, k)$ from p to p', followed by the reverse of the path in $\Theta(S_1, k)$ from q to p'. (Note that the latter is a path in $\Theta'(S, k)$ and, hence, in $SLS(S, k)$.) Otherwise, if y occurs at level two, then we move with x and y to the second level of the skip list spanner and use the same procedure to extend the paths from x to y. The formal algorithm is given in Figure 4.

```
**Algorithm** *walk*$(p, q)$
(* $p$ and $q$ are points of $S$; the algorithm constructs a $t$-spanner path in the
skip list spanner $SLS(S, k)$ from $p$ to $q$ *)
**begin**
$p_0 := p$; $q_0 := q$; $a := 0$; $b := 0$; $r := 0$; $s := 0$; $i := 1$;
(* $p_0 = p, p_1, \ldots, p_r, \ldots, p_a$ and $q_0 = q, q_1, \ldots, q_s, \ldots, q_b$ are
paths in $SLS(S, k)$, $r = \min\{j : p_j \in S_i\}$, $s = \min\{j : q_j \in S_i\}$, and
$p_r, p_{r+1}, \ldots, p_a, q_s, q_{s+1}, \ldots, q_b \in S_i$ *)
$stop := false$;
**while** $stop = false$
**do while** $p_a \neq q_b$ and $p_a \notin S_{i+1}$
    **do** $C :=$ cone of $\mathcal{C}$ such that $q_b \in C_{p_a}$;
        $p_{a+1} :=$ point of $C_{p_a} \cap S_i$ such that $(p_a, p_{a+1})$ is an edge of $\Theta(S_i, k)$;
        $a := a + 1$
    **od**;
    (* $p_a = q_b$ or $p_a \in S_{i+1}$ *)
    **while** $q_b \notin \{p_r, p_{r+1}, \ldots, p_a\}$ and $q_b \notin S_{i+1}$
    **do** $C :=$ cone of $\mathcal{C}$ such that $p_a \in C_{q_b}$;
        $q_{b+1} :=$ point of $C_{q_b} \cap S_i$ such that $(q_b, q_{b+1})$ is an edge of $\Theta(S_i, k)$;
        $b := b + 1$
    **od**;
    (* $q_b \in \{p_r, p_{r+1}, \ldots, p_a\}$ or both $p_a$ and $q_b$ occur in $S_{i+1}$ *)
    **if** $q_b \in \{p_r, p_{r+1}, \ldots, p_a\}$
    **then** $l :=$ index such that $q_b = p_l$;
          output the path $p_0, p_1, \ldots, p_l, q_{b-1}, q_{b-2}, \ldots, q_0$;
          $stop := true$
    **else** $i := i + 1$; $r := a$; $s := b$
    **fi**
**od**
**end**
```

Figure 4: Constructing a t-spanner path from p to q in the skip list spanner.

Lemma 6 *Let $k > 8$ and $\theta = 2\pi/k$. For any pair p and q of points in S, algorithm walk(p, q) constructs a t-spanner path in $SLS(S, k)$ from p to q, for $t = 1/(\cos\theta - \sin\theta)$.*

Proof: In this proof, we use the notation of the algorithm in Figure 4. Consider the paths $p_0 = p, p_1, p_2, \ldots$ and $q_0 = q, q_1, q_2, \ldots$ that are constructed by the algorithm. First note that if $p_a \neq q_b$ and $p_a \notin S_{i+1}$, then p_{a+1} exists, i.e., as long as the two paths do not meet and the last point on the p-path does not occur at level $i + 1$, the p-path can always move to a next point. A similar observation shows that the q-path can always be extended.

The proof of the lemma is by induction on the number of levels of the skip list spanner. To prove the base case, assume that $SLS(S, k)$ consists of only one level. Consider what happens during the first iteration of the outer while-loop. In the first inner while-loop, a path $p_0 = p, p_1, p_2, \ldots$ is constructed. This inner while-loop terminates iff the last point on this path is equal to q.

Let $a \geq 0$ and consider the points p_a and p_{a+1}. Then $p_a \neq q$. Let C be the cone such that $q \in C_{p_a}$. It follows from the algorithm that (p_a, p_{a+1}) is an edge of the skip list spanner, $p_{a+1} \in C_{p_a}$, and the projection of p_{a+1} onto the ray l_{C,p_a} is at least as close to p_a as the projection of q onto l_{C,p_a}. Therefore, by Lemma 2, we have

$$|p_{a+1}q| \leq |p_a q| - (\cos\theta - \sin\theta)|p_a p_{a+1}| < |p_a q|. \tag{1}$$

This proves that during each iteration of the first inner while-loop, the distance between p_a and q becomes strictly smaller. As a result, this while-loop terminates. Let z be the number of iterations made. Then the algorithm has constructed a path $p_0 = p, p_1, p_2, \ldots, p_z = q$. The second inner while-loop does not make any iterations. Since the points $p_0, p_1, \ldots, p_z$ are pairwise distinct, the variable l in the else-case has value z. Hence, the algorithm reports the path $p_0 = p, p_1, p_2, \ldots, p_z = q$ and terminates. The weight of this path is bounded by

$$\begin{aligned} \sum_{a=0}^{z-1} |p_a p_{a+1}| &\leq t \sum_{a=0}^{z-1} (|p_a q| - |p_{a+1} q|) \\ &= t(|p_0 q| - |p_z q|) \\ &= t|pq|. \end{aligned}$$

Hence, the algorithm has constructed a t-spanner path from p to q. This proves the base case of the induction.

Let $h > 1$, and consider a skip list spanner consisting of h levels. Assume the lemma holds for all skip list spanners with less than h levels.

Consider again the first iteration of the outer while-loop. During the first inner while-loop, a path $p_0 = p, p_1, p_2, \ldots, p_z$ is constructed such that $p_z = q$ or $p_z \in S_2$. Also, (1) holds for all $0 \leq a \leq z - 1$.

If $p_z = q$, then the path $p_0 = p, p_1, p_2, \ldots, p_z$ is reported and the algorithm terminates. In exactly the same way as above, it follows that this path is a t-spanner path from p to q.

Assume that $p_z \neq q$. Then $p_z \in S_2$. Note that the points $p_0, p_1, \ldots, p_z$ are pairwise distinct. During the second inner while-loop, a path $q_0 = q, q_1, q_2, \ldots$ is constructed.

Using Lemma 2, it follows that

$$|q_{b+1}p_z| \leq |q_bp_z| - (\cos\theta - \sin\theta)|q_bq_{b+1}| < |q_bp_z|. \tag{2}$$

This second inner while-loop terminates iff the last point on the q-path is equal to one of the p_i's or occurs at level two of the skip list spanner. Since during each iteration, the distance between q_b and p_z becomes strictly smaller, this while-loop terminates. Let y be the number of iterations made. Then the algorithm has constructed a path $q_0 = q, q_1, q_2, \ldots, q_y$. It follows from (2) that the points on this path are pairwise distinct. Then, it follows from the termination condition that all points $p_0, p_1, \ldots, p_z, q_0, q_1, \ldots, q_{y-1}$ are pairwise distinct. There are two possible cases.

First assume that $q_y \in \{p_0, p_1, \ldots, p_z\}$. Let l be such that $q_y = p_l$. Then the algorithm reports the path $p_0 = p, p_1, \ldots, p_l = q_y, q_{y-1}, \ldots, q_0 = q$, having weight

$$\begin{aligned}
&\sum_{a=0}^{l-1}|p_ap_{a+1}| + \sum_{b=0}^{y-1}|q_bq_{b+1}| \\
&\leq \sum_{a=0}^{z-1}|p_ap_{a+1}| + \sum_{b=0}^{y-1}|q_bq_{b+1}| \\
&\leq t\sum_{a=0}^{z-1}(|p_aq| - |p_{a+1}q|) + t\sum_{b=0}^{y-1}(|q_bp_z| - |q_{b+1}p_z|) \\
&= t(|p_0q| - |p_zq| + |q_0p_z| - |q_yp_z|) \\
&= t(|pq| - |q_yp_z|) \\
&\leq t|pq|.
\end{aligned}$$

Hence, in this case the algorithm has constructed a t-spanner path from p to q.

Next assume that $q_y \notin \{p_0, p_1, \ldots, p_z\}$. Then $q_y \in S_2$ and the algorithm moves to level two of the skip list spanner. Note that the rest of the algorithm "takes place" at levels $2, \ldots, h$. These levels constitute a skip list spanner $SLS(S_2, k)$ consisting of $h - 1$ levels. Therefore, by the induction hypothesis, a t-spanner path from p_z to q_y is constructed during the rest of the algorithm. At termination, the algorithm reports the concatenation of the path $p_0, p_1, \ldots, p_z$, the t-spanner path from p_z to q_y, and the path $q_y, q_{y-1}, \ldots, q_0$. The weight of this path is bounded by

$$\begin{aligned}
&\sum_{a=0}^{z-1}|p_ap_{a+1}| + t|p_zq_y| + \sum_{b=0}^{y-1}|q_bq_{b+1}| \\
&\leq t\sum_{a=0}^{z-1}(|p_aq| - |p_{a+1}q|) + t|p_zq_y| + t\sum_{b=0}^{y-1}(|q_bp_z| - |q_{b+1}p_z|) \\
&= t(|p_0q| - |p_zq| + |p_zq_y| + |q_0p_z| - |q_yp_z|) \\
&= t|pq|.
\end{aligned}$$

Hence, also in this case the algorithm has constructed a t-spanner path from p to q. This completes the proof. ■

Remark 3 Consider the t-spanner path $p = p_0, p_1, \ldots, p_l = q_b, q_{b-1}, \ldots, q_0 = q$ that is computed by algorithm *walk*(p, q). It follows from the proof of Lemma 6 that for each fixed i, all p-points and all q-points that are added during the iteration of the outer while-loop that takes place at level i are pairwise distinct.

In the rest of this section, we analyze the expected behavior of algorithm *walk*. Let p and q be two fixed points of S. Let T and N denote the running time of algorithm *walk*(p, q) and the number of edges on the t-spanner path from p to q that is constructed by this algorithm, respectively. Note that T and N are random variables.

For each point p_{a+1} added to the p-path, we have to find the cone C such that $q_b \in C_{p_a}$. Similarly, for each point q_{b+1} added to the q-path, we have to find the cone C' such that $p_a \in C'_{q_b}$. In the planar case, such a cone can easily be computed from the angle made by the vector $\overrightarrow{p_a q_b}$ (resp. $\overrightarrow{q_b p_a}$) with the positive x-axis. For the d-dimensional case, we solve this problem as follows:

Recall that each cone of $\mathcal{C}$ is defined by d hyperplanes. We store the arrangement of all these $m := d|\mathcal{C}|$ hyperplanes in the data structure of [4]. With each face of the arrangement we store the name of one cone that contains this face. This structure has size $O(m^d)$ and allows to locate any point in $O(\log m)$ time. Note that $m = O((c/\theta)^{d-1})$.

To find a cone C such that $q_b \in C_{p_a}$, we locate the point $q_b - p_a$ in the arrangement. The cone that is reported is the one we are looking for.

It follows that $T = O(N \log(1/\theta) + h)$. Therefore, $E(T) = O(E(N)\log(1/\theta) + E(h)) = O(E(N)\log(1/\theta) + \log n)$. Hence, it suffices to estimate the expected value of N. Note that we use an extra amount of $O((c/\theta)^{d(d-1)})$ space.

Consider again the paths $p_0 = p, p_1, p_2, \ldots$ and $q_0 = q, q_1, q_2, \ldots$ that are constructed by the algorithm. Let i, $1 \leq i \leq h$, be fixed. We estimate the expected number of points that are added to the paths at level i of the skip list spanner.

Intuitively, the expected number of points added at level i is bounded by a constant. During the first inner while-loop, the p-path is extended until it meets the q-path or the last point on it occurs at level $i+1$. Since each point of S_i occurs at level $i+1$ with probability $1/2$, we expect that—at level i—at most a constant number of points are added to the p-path. During the second inner while-loop, the q-path is extended. By a similar argument, we expect that—at level i—at most a constant number of points are added to this path.

To make this rigorous, we have to show that each point added to one of these paths indeed occurs at level $i+1$ with probability $1/2$. In particular, we have to show that it is *not* the case that the coin flips that are used to build the skip list spanner cause the algorithm to visit points at level i for which it is more likely that they do not occur at level $i+1$.

Fix the sets $S_1, S_2, \ldots, S_i$. Let r and s be the minimal indices such that $p_r \in S_i$ and $q_s \in S_i$, respectively. Note that r and s are completely determined once p, q and $S_1, \ldots, S_i$ are fixed.

For the *sake of analysis*, assume that we have not yet flipped our coin for determining the set S_{i+1}. Consider the path $p'_r = p_r, p'_{r+1}, p'_{r+2}, \ldots, p'_m = q_s$ that the algorithm *would have* constructed if all points of S_i did not occur at level $i+1$. (It follows from the proof of Lemma 6 that the algorithm indeed would have constructed a path from p_r to q_s. Moreover, the points on this path are pairwise distinct.) Now let z be the number of points that are added—at level i—to the p-path by the *actual* algorithm. Note that z is a random variable.

Let $l \geq 0$ and assume that $z = l$. It is easy to see that $p'_r = p_r, p'_{r+1} = p_{r+1}, \dots, p'_{r+l} = p_{r+l}$. It follows from the actual algorithm that $p'_a \notin S_{i+1}$ for all a, $r \leq a \leq r + l - 1$. Therefore,

$$\Pr(z = l) \leq \Pr\left(\bigwedge_{a=r}^{r+l-1} (p'_a \notin S_{i+1})\right).$$

Since the path $p'_r, p'_{r+1}, \dots, p'_m$ is completely determined by the points p and q and the sets $S_1, \dots, S_i$, each of the points on this path is contained in S_{i+1} with probability $1/2$. Therefore, using the fact that all coin flips are independent, it follows that $\Pr(z = l) \leq \prod_{a=r}^{r+l-1} \Pr(p'_a \notin S_{i+1}) = (1/2)^l$. That is, the random variable z has a geometric distribution with parameter $1/2$.

Again, for the *sake of analysis*, consider the following experiment. We assume that we have not yet flipped our coin for determining the set S_{i+1}. Now we flip the coin for the points $p'_r, p'_{r+1}, p'_{r+2}, \dots$, in this order, stopping as soon as we obtain heads or after having obtained $m - r$ times tails. Clearly, the number of times we obtain tails has the same distribution as the random variable z above.

Let l, $0 \leq l \leq m - r$, be fixed and assume that $z = l$. If $l = m - r$, then the p-path constructed by the *actual* algorithm has reached point q_s and the algorithm terminates. So assume that $l < m - r$. Then, at this moment, we know that $p'_r, p'_{r+1}, \dots, p'_{r+l-1}$ do not occur at level $i + 1$, p'_{r+l} occurs at level $i + 1$, and for all points of $S'_i := S_i \setminus \{p'_r, p'_{r+1}, \dots, p'_{r+l}\}$ we have not yet flipped the coin. Let $q'_s = q_s, q'_{s+1}, q'_{s+2}, \dots$ be the path that *would have been* constructed during the second inner while-loop if all points of S'_i did not occur at level $i + 1$. Let y be the number of points of S_i that are added—at level i—to the q-path by the *actual* algorithm. Then, y is a random variable.

Let $t \geq 0$ and assume that $y = t$. Then, $q'_s = q_s, q'_{s+1} = q_{s+1}, \dots, q'_{s+t} = q_{s+t}$. By Remark 3, all points $p'_r, p'_{r+1}, \dots, p'_{r+l}, q'_s, q'_{s+1}, \dots, q'_{s+t-1}$ are pairwise distinct. In particular, $q'_b \in S'_i$ for all b, $s \leq b \leq s + t - 1$. As a result, we can say that in the actual skip list spanner, each q'_b occurs at level $i + 1$ independently with probability $1/2$. Since $q'_b \notin S_{i+1}$ for all b, $s \leq b \leq s + t - 1$, it follows that

$$\Pr(y = t) \leq \Pr\left(\bigwedge_{b=s}^{s+t-1} (q'_b \notin S_{i+1})\right) = \prod_{b=s}^{s+t-1} \Pr(q'_b \notin S_{i+1}) = (1/2)^t.$$

To summarize, conditional on fixed subsets $S_1, S_2, \dots, S_i$ and a fixed value of the random variable z, the random variable y has a geometric distribution with parameter $1/2$. Since this distribution does not depend on z, y also has a geometric distribution conditional on $S_1, \dots, S_i$ only.

Altogether, conditional on fixed subsets $S_1, S_2, \dots, S_i$, the random variables that count the number of points that are added—at level i—to the p- and q-paths both have a geometric distribution with parameter $1/2$. Since both distributions do not depend on $S_1, \dots, S_i$, this statement also holds unconditionally.

Now we can analyze the expected behavior of algorithm *walk*(p, q) in exactly the same way as for standard skip lists. (See e.g. Section 1.4 in Mulmuley [10].) Recall that T and N denote the running time of algorithm *walk*(p, q) and the number of edges on the t-spanner path computed by this algorithm, respectively. We saw already that $T = O(N \log(1/\theta) + h)$.

For $1 \leq i \leq h$, let M_i (resp. N_i) denote the number of edges that are added at level i to the p-path (resp. q-path). Then $N = \sum_{i=1}^{h}(M_i + N_i)$. Moreover, $M_1, N_1, M_2, N_2, \ldots, M_h, N_h$ are random variables, and each one is distributed according to a geometric distribution with parameter $1/2$. Each of these variables is independent of the ones that come later in the given enumeration. Using the Chernoff bound and the fact that $h = O(\log n)$ with high probability, it follows that $N = O(\log n)$ with high probability. (See e.g. [10].) This implies that the running time T is bounded by $O(\log(1/\theta) \log n)$ with high probability.

These bounds hold for fixed points p and q of S. Since there are only a quadratic number of such pairs, it follows that the maximum running time of algorithm *walk*, and the maximum number of edges on any t-spanner path computed by this algorithm are bounded by $O(\log(1/\theta) \log n)$ and $O(\log n)$, respectively, both with high probability. (See Observation 1.3.1 on page 10 of [10].) That is, with high probability, the skip list spanner has *spanner diameter* $O(\log n)$. In particular, this proves that there *exists* a t-spanner for S having $O(n)$ edges and $O(\log n)$ spanner diameter.

We summarize our result:

Theorem 2 *Let $k > 8$ be an integer, let $\theta = 2\pi/k$ and let S be a set of n points in $\mathbb{R}^d$.*

1. *The skip list spanner $SLS(S, k)$ is a t-spanner for $t = 1/(\cos\theta - \sin\theta)$. It contains an expected number of $O((c/\theta)^{d-1} n)$ edges, for some constant c.*
2. *Using $O((c/\theta)^{d-1} n + n \log^{d-2} n)$ expected space, this graph can be constructed in expected time $O((c/\theta)^{d-1} n \log^{d-1} n)$.*
3. *The expected maximum time to construct a t-spanner path from any point of S to any other point of S is bounded by $O(\log(1/\theta) \log n)$.*
4. *The expected spanner diameter of the skip list spanner is bounded by $O(\log n)$.*
5. *In all these bounds, the expectation is taken over all coin flips that are used to build the skip list spanner. Moreover, all bounds hold with high probability.*

4 Maintaining the skip list spanner

In this section, we consider the problem of maintaining the skip list spanner under insertions and deletions of points. Unfortunately, it is not possible—for our spanner—to achieve polylogarithmic update time for arbitrary insertions and deletions. Since there may be points in $\Theta(S, k)$ having $\Omega(n)$ in-degree, the worst-case update time is doomed to be $\Omega(n)$. We will see, however, that in the model of random insertions and deletions (see [10]), we can obtain polylogarithmic expected update time. We briefly recall the model of random updates. (For a detailed description, see [10].)

Consider any sequence u consisting of insert and delete operations. The *signature* σ of u is defined as the string consisting of the symbols $+$ and $-$ such that the i-th symbol is a $+$ (resp. $-$) iff the i-th update operation of u is an insertion (resp. a deletion). Let V be the set of points involved in u. Then we say that u is a *(V, σ)-sequence*.

Signatures must satisfy certain obvious conditions to be valid [10], e.g. a point cannot be deleted if it is not present.

Consider a set V of n points and a valid signature σ. An update sequence u is called a *random (V, σ)-sequence* if it is chosen from the uniform distribution on all (V, σ)-sequences. We can also regard a random (V, σ)-sequence as follows. Read the string σ from left to right. If the current symbol is $+$, then choose a random point of V that is not present yet and insert it. If the current symbol is $-$, then choose a random point that is present and delete it.

Let u be a random (V, σ)-sequence. For each i, let p_i denote the point of V that is involved in the i-th update of u, let V_i denote the set of points in V that are "present" at the start of the i-th update, and let n_i denote the size of V_i. (Note that n_i is completely determined by σ.) Then Mulmuley observes that

1. p_i is a random point of V, and
2. V_i is a random subset of V of size n_i.

Let S be a set of n points in $\mathbb{R}^d$. We first show how to maintain the graph $\Theta(S, k)$ under insertions and deletions.

If we insert a point q into S, then we have to compute all edges in $\Theta(S \cup \{q\}, k)$ having q as a source or a sink. Also, some edges have to be removed from the graph. The edges with source q can be found using the data structure of Lemma 3. The following observation indicates how the edges with sink q can be found.

Observation 1 *Let $q \in \mathbb{R}^d \setminus S$, and let C be a cone of $\mathcal{C}$. Let p be any point of S such that $q \in C_p$ or, equivalently, $p \in -C_q$.*

1. If

(a) there is no edge (p, r) in $\Theta(S, k)$ such that $r \in C_p$, or

(b) there is an edge (p, r) in $\Theta(S, k)$ such that $r \in C_p$ and the projection of q onto $l_{C,p}$ is closer to p than the projection of r onto $l_{C,p}$,

then the graph $\Theta(S \cup \{q\}, k)$ contains an edge from p to q.

2. If there is an edge (p, r) in $\Theta(S, k)$ such that $r \in C_p$ and the projection of r onto $l_{C,p}$ is closer to p than the projection of q onto $l_{C,p}$, then the graph $\Theta(S \cup \{q\}, k)$ does not contain an edge from p to q.

3. If there is an edge (p, r) in $\Theta(S, k)$ such that $r \in C_p$ and the projections of q and r onto $l_{C,p}$ are at the same distance from p, then there is no need to add an edge from p to q when q is inserted into S.

Similarly, if we delete a point q from S, then we have to delete all edges having q as a source or a sink. Deleting the edges with source q does not cause any problems. For any deleted edge (p,q)—having q as its sink—however, we have to find a new edge (p,r) such that $r \in C_p$, C being the cone such that $q \in C_p$.

This discussion suggests the following data structure for maintaining the graph $\Theta(S,k)$.

1. We store the graph $G = \Theta(S,k)$. With each point p of S, we store a dictionary containing all points q of S, such that (p,q) is an edge of G, and a dictionary containing all points r of S such that (r,p) is an edge of G. (The elements in these dictionaries are sorted by any ordering, e.g. by their names.)

2. For each cone C of $\mathcal{C}$, we store the data structure T_C of Lemma 3 for the points of S.

3. For each cone C of $\mathcal{C}$, we store a $(d+1)$-layer data structure T'_C for the points of S, which is defined as follows. (See Figure 5.)

   Recall the coordinates $p'_1, p'_2, \ldots, p'_{d+1}$ that we defined in Section 2. (These coordinates depend on C.)

   We store the points of S in a $(d+1)$-layer data structure T'_C, where each layer-j tree stores points sorted by their p'_j-coordinates, $1 \leq j \leq d$. With each node u of each layer-d tree, we store the following additional information. Let S^u be the subset of S that is stored in the subtree of u. We store with u two layer-$(d+1)$ trees:

   (a) a balanced binary search tree T_1^u storing all points p of S^u such that $\Theta(S,k)$ does not contain an edge with source p and sink in C_p. These points are stored in the leaves of the tree, sorted by their p'_{d+1}-coordinates. (In fact, any ordering can be taken, because we only use T_1^u as a dictionary.)

   (b) a balanced binary search tree T_2^u for the points in the set

   $$\{r^p \in C_p \cap S : p \in S^u \text{ and } \Theta(S,k) \text{ contains an edge from } p \text{ to } r^p\}.$$

   These points are stored in the leaves of the tree, sorted by their $(r^p)'_{d+1}$-coordinates. Moreover, the leaves are linked by pointers.

Having defined the data structure, we can give the update algorithms. Suppose we want to insert a point q. Assume w.l.o.g. that $q \notin S$.

1. For each cone C of $\mathcal{C}$, we do the following:

   (a) We use the data structure T_C to compute a point p in $C_q \cap S$ whose projection onto $l_{C,q}$ is closest to q. If p exists, then we insert the edge (q,p) into G.

   (b) We insert q into T_C.

   (c) We insert q into T'_C: We search for q in the first d layers. For each node u on the search path that belongs to a layer-d tree (there are $O(\log^d n)$ such nodes),

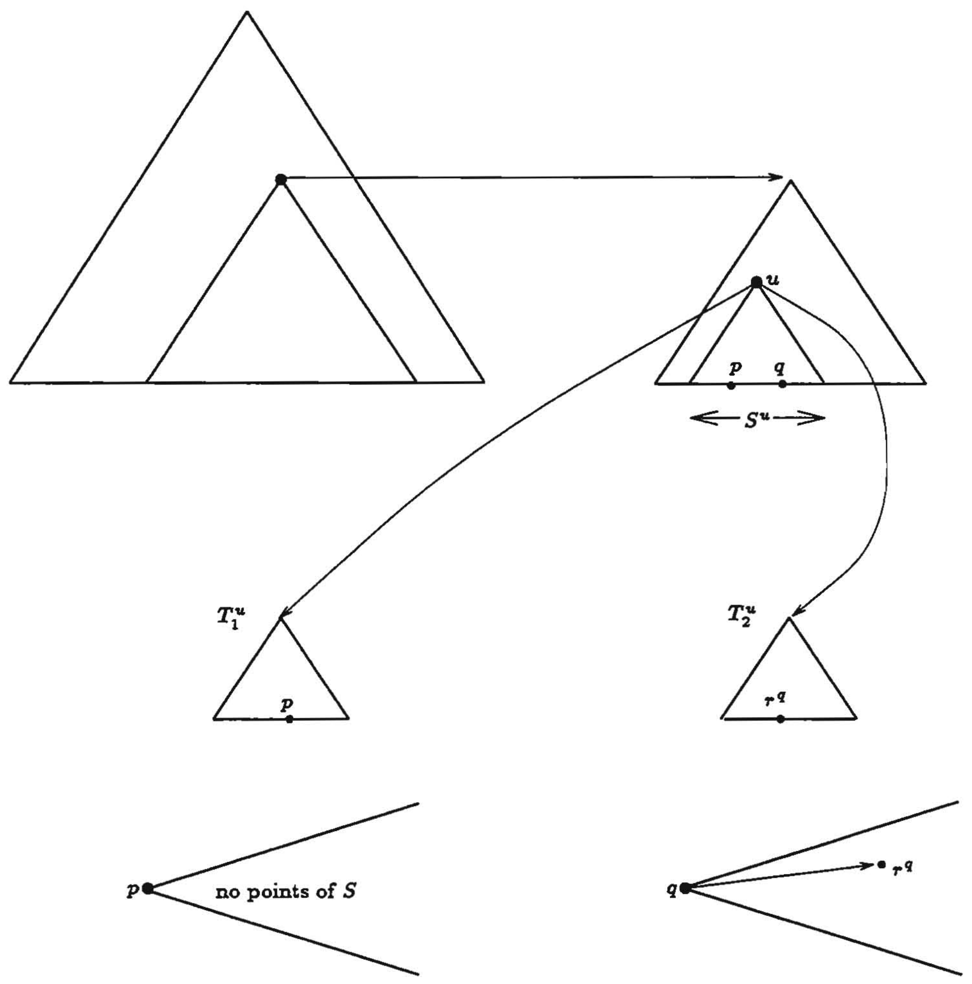

Figure 5: Illustration of the $(d+1)$-layered data structure T'_C for the case $d = 2$.

- if p does not exist, then we insert q into T_1^u,
- if p exists, then we insert p—being the point r^q—into T_2^u.

Finally, we insert q into all layer-j trees of T'_C, $1 \leq j \leq d$, into which it belongs and, if necessary, rebalance the data structure.

2. For each cone C of $\mathcal{C}$, we do the following:
   (a) We compute a set of $O(\log^d n)$ canonical nodes of layer-d trees in T'_C such that all sets stored in the subtrees of these nodes partition the set of all points of $S \setminus \{q\}$ that are contained in the cone $-C_q$.
   (b) For each of these nodes v,
      - we take all points stored in the layer-$(d+1)$ tree T_1^v. For each such point p, we insert the edge (p, q) into G.
      - we walk along the leaves of the layer-$(d+1)$ tree T_2^v, from right to left. For each point r^p encountered such that $(r^p)'_{d+1} > q'_{d+1}$, we replace the

edge (p, r^p) in G by (p, q).

   (c) For each edge (p, q) added to G in Step 2(b), we search for p in the first d layers of T'_C. For each node w on the search path that belongs to a layer-d tree,

      - if prior to the insertion of q there was no edge with source p and sink in C_p, then we delete p from the layer-$(d+1)$ tree T_1^w, and insert q—being the point r^p—into the corresponding tree T_2^w.
      - if prior to the insertion of q there was an edge with source p and sink in C_p, then we replace the occurrence of r^p in T_2^w by q—being the new point r^p.

The deletion algorithm is similar. Suppose we want to delete a point q from S.

1. We delete all edges from G having q as its source. (Note that each such edge is stored twice, see item 1 of the definition of the data structure.)
2. For each cone C of $\mathcal{C}$, we do the following:

   (a) We delete q from T_C.

   (b) We search for q in the first d layers of T'_C. For each node u on the search path that belongs to a layer-d tree,

      - if q is contained in the tree T_1^u, then we delete it,
      - otherwise, we delete r^q from the tree T_2^u.

   Finally, we delete q from all layer-j trees of T'_C, $1 \leq j \leq d$, in which it occurs and, if necessary, rebalance the data structure.

3. We delete all edges from G having q as its sink. For each such edge (p, q), let C be the cone such that $q \in C_p$. Then we do the following:

   (a) We use the data structure T_C and compute a point q' in $C_p \cap S$ whose projection onto $l_{C,p}$ is closest to p. If q' exists, then we insert the edge (p, q') into G.

   (b) We search for p in the first d layers of T'_C. For each node v on the search path that belongs to a layer-d tree,

      - if q' does not exist, then we insert p into the tree T_1^v and delete q—being the old point r^p—from T_2^v,
      - if q' exists, then we replace the occurrence of q in T_2^v—being the old point r^p—by q'—which is the new point r^p.

This concludes the description of the update algorithms. It is not difficult to see that these algorithms correctly maintain the graph $\Theta(S, k)$ and the corresponding data structures T_C and T'_C. The complete data structure has size $O((c/\theta)^{d-1} n \log^d n)$.

We analyze the update time. Consider again the insertion algorithm. By Lemma 3, Step 1(a) takes $O(\log^d n)$ time and Step 1(b) takes $O(\log^d n)$ amortized time, per cone C. Using dynamic fractional cascading [9], Step 1(c) takes $O(\log^d n \log\log n)$

amortized time per cone C. Hence, the total amortized time for Step 1 is bounded by $O((c/\theta)^{d-1} \log^d n \log\log n)$.

To estimate the time for Step 2, we fix a cone C. Step 2(a) takes $O(\log^d n)$ time. Let D_C denote the number of edges in the graph $\Theta(S \cup \{q\}, k)$ with sink q and source in $-C_q$. Then Step 2(b) takes $O(D_C \log n)$ time. Using dynamic fractional cascading, Step 2(c) takes $O(\log^d n \log\log n)$ amortized time for each of the D_C points p. It follows that the total amortized time for Step 2 is bounded by $O((c/\theta)^{d-1} \log^d n + \sum_C D_C \log^d n \log\log n)$.

Let D denote the in-degree of the new point q in the graph $\Theta(S \cup \{q\}, k)$. Then $D = \sum_C D_C$ and the total amortized time of the insertion algorithm is bounded by

$$O\left(\left((c/\theta)^{d-1} + D\right) \log^d n \log\log n\right).$$

By similar arguments, it follows that the amortized deletion time is bounded by the same quantity, now D being the in-degree of q in the graph $\Theta(S, k)$.

Note that these update times hold for any update. In the worst-case, the value of D can be $n-1$. The following lemma shows that for a random update, the expected value of D is small.

Lemma 7 *Let V be a set of points in $\mathbb{R}^d$ and let S be a random subset of V of size n. Let q be a random point of V. Then the expected in-degree of q in the graph $\Theta(S \cup \{q\}, k)$ is at most equal to the number of cones in $\mathcal{C}$.*

Proof: Let m denote the number of cones and let n' denote the size of $S \cup \{q\}$. Note that n' is equal to n or $n+1$. The graph $\Theta(S \cup \{q\}, k)$ contains at most mn' edges. Hence, the average in-degree in this graph is at most m. Since q is a random point in $S \cup \{q\}$, the claim follows. ∎

Consider a random (V, σ)-sequence. If S is the set of points at the start of the i-th update operation, then S is a random subset of V. Also, if q is the point involved in the i-th update, then q is a random point of V. Hence the expected value of D is at most equal to the number of cones in $\mathcal{C}$. This proves:

Lemma 8 *Using a data structure of size $O((c/\theta)^{d-1} n \log^d n)$, we can maintain the graph $\Theta(S, k)$ in $O((c/\theta)^{d-1} \log^d n \log\log n)$ expected amortized time per random update.*

Recall that the skip list spanner, $SLS(S, k)$, consists of Θ-graphs at levels, $1 \leq i \leq h$. For each level i of $SLS(S, k)$, we maintain the data structure described above for the points of S_i. (By a trivial extension, we do not only maintain the graph $\Theta(S_i, k)$, but also its reverse $\Theta'(S_i, k)$.)

To insert a point q, we flip our coin and determine the number of levels into which q has to be inserted. If this number is l, then we use the insertion algorithm given above to insert q into the augmented Θ-graphs corresponding to the levels $1, 2, \ldots, l$.

To delete a point q, we use the deletion algorithm given above to delete q from all augmented Θ-graphs in which it occurs.

To analyze the update time, suppose we update the augmented Θ-graph of level i. Since S_i is a random subset of S, which in turn is a random subset of V, Lemma 7 also holds for the graph that is stored at level i. Also, note that during an update, we update a constant expected number of levels. Therefore, Lemma 8 implies that the expected amortized update time of the augmented skip list spanner is bounded by $O((c/\theta)^{d-1} \log^d n \log\log n)$ per random update.

We have proved our final result:

Theorem 3 *Let $k > 8$ be an integer, let $\theta = 2\pi/k$ and let S be a set of n points in $\mathbb{R}^d$. Using a data structure of expected size $O((c/\theta)^{d-1} n \log^d n)$, we can maintain the skip list spanner $SLS(S,k)$ under random insertions and deletions, in an amount of $O((c/\theta)^{d-1} \log^d n \log\log n)$ expected amortized time per random update.*

Here, the expectation is taken over all coin flips that are used to build the skip list spanner and to determine the update sequence.

5 Concluding remarks

We have presented randomized algorithms for constructing a t-spanner with an expected number of $O(n)$ edges and $O(\log n)$ expected spanner diameter. This spanner can be constructed in $O(n \log^{d-1} n)$ expected time. For any pair p and q of points, a t-spanner path from p to q, containing an expected number of $O(\log n)$ edges, can be constructed in $O(\log n)$ expected time. All these bounds hold with high probability.

After augmenting this spanner with an additional data structure of size $O(n \log^d n)$, we can maintain it in the model of random updates, in $O(\log^d n \log\log n)$ expected amortized time per random insertion and deletion.

In [1], we give a deterministic construction for t-spanners of $O(\log n)$ spanner diameter that is based on the well-separated pair decompositions of [3]. It is not known, however, how to maintain these spanners under insertions and deletions.

We leave open the problem of providing dynamic spanners which can be efficiently updated for arbitrary updates.

Acknowledgements

The authors thank Andrea Eßer for producing the figures.

References

[1] S. Arya, D.M. Mount and M. Smid. *Randomized and deterministic algorithms for geometric spanners of small diameter.* Proc. 35th Annu. IEEE Symp. on Foundations of Computer Science, 1994, to appear.

[2] S. Arya and M. Smid. *Efficient construction of a bounded degree spanner with low weight.* Proc. 2nd Annual Symposium on Algorithms (ESA), Lecture Notes in Computer Science, Vol. 855, Springer-Verlag, Berlin, 1994, pp. 48–59.

[3] P.B. Callahan and S. Rao Kosaraju. *Faster algorithms for some geometric graph problems in higher dimensions.* Proc. 4th Annu. ACM-SIAM Symp. on Discrete Algorithms, 1993, pp. 291–300.

[4] B. Chazelle and J. Friedman. *Point location among hyperplanes and unidirectional ray-shooting.* Computational Geometry, Theory and Applications **4** (1994), pp. 53–62.

[5] G. Das and G. Narasimhan. *A fast algorithm for constructing sparse Euclidean spanners.* Proc. 10th Annu. ACM Sympos. Comput. Geom., 1994, pp. 132–139.

[6] G. Das, G. Narasimhan and J.S. Salowe. *A new way to weigh malnourished Euclidean graphs.* Proc. 6th Annu. ACM-SIAM Symp. on Discrete Algorithms, 1995, to appear.

[7] J.M. Keil and C.A. Gutwin. *Classes of graphs which approximate the complete Euclidean graph.* Discrete Comput. Geom. **7** (1992), pp. 13–28.

[8] G.S. Lueker. *A data structure for orthogonal range queries.* Proc. 19th Annu. IEEE Symp. on Foundations of Computer Science, 1978, pp. 28–34.

[9] K. Mehlhorn and S. Näher. *Dynamic fractional cascading.* Algorithmica **5** (1990), pp. 215-241.

[10] K. Mulmuley. *Computational Geometry, an Introduction through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, 1994.

[11] F.P. Preparata and M.I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, New York, 1985.

[12] W. Pugh. *Skip lists: a probabilistic alternative to balanced search trees.* Commun. ACM **33** (1990), pp. 668–676.

[13] J. Ruppert and R. Seidel. *Approximating the d-dimensional complete Euclidean graph.* Proc. 3rd Canadian Conf. on Computational Geometry, 1991, pp. 207–210.

[14] J.S. Salowe. *Constructing multidimensional spanner graphs.* Internat. J. Comput. Geom. Appl. **1** (1991), pp. 99–107.

[15] P.M. Vaidya. *A sparse graph almost as good as the complete graph on points in K dimensions.* Discrete Comput. Geom. **6** (1991), pp. 369–381.

[16] A.C. Yao. *On constructing minimum spanning trees in k-dimensional spaces and related problems.* SIAM J. Comput. **11** (1982), pp. 721-736.