Randomized and deterministic algorithms for geometric spanners of small diameter

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

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.

Randomized and deterministic algorithms are given for constructing t-spanners consisting of $O(n)$ edges and having $O(\log n)$ diameter. Also, it is shown how to maintain the randomized t-spanner under random insertions and deletions.

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 $\mathbb{R}^d$ 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. In [3, 5, 11, 12, 15], efficient algorithms are given for constructing a t-spanner with $O(n)$ edges. In [13], it is shown how a t-spanner can be computed having $O(n)$ edges such that each point has a degree that is bounded by a constant. This result was extended in [1, 4]. In these papers it is shown how to efficiently construct a t-spanner with $O(n)$ edges, such that each point has bounded degree and the total length of all edges is bounded by $O(\log n)$ (resp. $O(1)$) times the length of a minimum spanning tree for S.

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*. The latter 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. Moreover, it should be possible to compute such a t-spanner path efficiently. To our knowledge, this natural problem has not been considered before.

We also consider the problem of maintaining a t-spanner if points are inserted and deleted in S. All spanners referred to above are static.

1.1 Summary of results

Given a t-spanner for a set S, define a *path query* to be a pair of points $p, q \in S$. The answer to a path query is a t-spanner path from p to q, that is, a path whose length is at most t times the Euclidean distance between p and q.

Intuitively, our results may be viewed as one way of generalizing skip lists to higher dimensions (also see [8]). Assume that the points of S are one-dimensional. Consider a *skip list* [10] for the points of S. We can regard this data structure as a directed graph. This graph has

[*]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).

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 [8].) 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 the first part of this paper, we generalize this idea to the d-dimensional case, for any fixed d, by combining the Θ-graph of [5, 11] with skip lists and range trees [7, 9]. This gives a *randomized* spanner.

Theorem 1 *Let $t > 1$, and let S be a set of n points in $\mathbb{R}^d$.*

1. *There exists a t-spanner for S having an expected number of $O(n)$ edges and whose expected spanner diameter is bounded by $O(\log n)$.*
2. *Using an associated data structure of size $O(n)$, the expected maximum time to answer any path query is bounded by $O(\log n)$. Such a path query computes a t-spanner path containing an expected number of $O(\log n)$ edges.*
3. *Using $O(n \log^{d-2} n)$ expected space, we can build this t-spanner in $O(n \log^{d-1} n)$ expected time.*
4. *Using $O(n \log^d n)$ expected space, we can maintain this t-spanner under random insertions and deletions [8], in $O(\log^d n \log\log n)$ expected amortized time per random update.*
5. *In all these bounds, the expectation is taken over all coin flips that are used to build the t-spanner. Moreover, all bounds hold with high probability.*

The constant factors are of the form $(c/(t-1))^{d-1}$ (for a suitably chosen constant c), except in the bound on the number of edges on the path from p to q (where the constant factor is independent of t and d) and in the time for constructing a t-spanner path (where the constant is proportional to $\log(c/(t-1))$).

In the second part of the paper, we consider *deterministic* spanners. This construction is based on a well-separated pair decomposition of the point set [2]. Construction of spanners is asymptotically more efficient, but path query processing is slower, and updates are not considered.

Theorem 2 *Let $t > 1$, and let S be a set of n points in $\mathbb{R}^d$.*

1. *There exists a t-spanner for S having $O(n)$ edges and whose spanner diameter is bounded by $O(\log n)$.*
2. *Using an associated data structure of size $O(n)$, we can answer any path query in $O(\log^2 n)$ time. Such a path query computes a t-spanner path containing $O(\log n)$ edges.*
3. *Alternatively, using an associated data structure of size $O(n \log n)$, we can answer any path query in $O(\log n)$ time. Such a path query computes a t-spanner path containing $O(\log n)$ edges.*
4. *Using $O(n)$ space, we can build this t-spanner in $O(n \log n)$ time.*

Regarding the constant factors in these results, the size of the spanner is $O((c/(t-1))^d n)$, the construction time is $O(n \log n + (c/(t-1))^d n)$, and query time is $O(d \log^2 n + d^2 \log(c/(t-1)) \log n)$. As before, constant factors for path length are independent of t and d.

2 Spanners, simplicial cones and the Θ-graph

In this section we review some results that we will later use to construct randomized spanners. 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*. The Euclidean distance between the points p and q in $\mathbb{R}^d$ is denoted by $|pq|$. Let $t > 1$. A graph $G = (S, E)$ is called a *t-spanner* for S if for any pair p, 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.

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 angle between any two vectors $\overrightarrow{pq}$ and $\overrightarrow{pr}$, 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 (i) each cone has its apex at the origin, (ii) each cone has angular diameter at most θ, (iii) 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 of $i \cdot \theta$, $0 \leq i < k$. 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 [5] defined these for the case $d = 2$. Ruppert and Seidel [11] defined them for arbitrary dimensions $d \geq 2$. The following technical result is needed to prove the spanner bounds. A proof can be found in [11].

Definition 1 ([5, 11]) 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: The vertices of $\Theta(S, k)$ are the points of S. 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.

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

We 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.

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 $q'_i \geq p'_i$, $1 \leq i \leq d$, a point with minimal q'_{d+1}-coordinate. This problem can be solved using a d-layer range tree T_C ([9]). Note that this data structure depends on the cone C.

Lemma 2 ([6]) *Let S be a set of n points in $\mathbb{R}^d$, and let C be a cone of $\mathcal{C}$. The d-layered range tree 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 using a sweep algorithm. The result is as follows. (We remark that this result was proved already in [5] for the planar case and in [11] for the case where $d \geq 2$. Our algorithm uses a factor of $O(\log n)$ less space than the algorithm of [11].)

Theorem 3 ([5, 11]) *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 [10].

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 $\emptyset = S_{h+1} \subseteq S_h \subseteq S_{h-1} \subseteq S_{h-2} \subseteq \ldots \subseteq S_2 \subseteq S_1 = S$. It is well known that (i) $h = O(\log n)$ with high probability; in particular, $E(h) = O(\log n)$, (ii) $E(|S_i|) = n/2^{i-1}$, $1 \leq i \leq h$, and (iii) $\sum_{i=1}^{h} |S_i| = O(n)$ 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. For each $1 \leq i \leq h$, there is a graph $\Theta(S_i,k)$, and a reversed graph $\Theta'(S_i,k)$, which is obtained from $\Theta(S_i,k)$ by reversing the direction of each edge. We say that the points of S_i are at *level i* of the data structure. 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 $\Theta'(S_i,k)$.

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 $\Theta(S_1,k)$. Then x' is the next point on 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. In the latter case, 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'. 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. The formal algorithm is given in Figure 1.

```
Algorithm walk(p, q)
(* p and q are points of S. This algorithm
   constructs a t-spanner path in the skip list
   spanner SLS(S, k) from p to q.
*)
begin
$p_0 := p$; $q_0 := q$;
$a := b := r := s := 0$;
$i := 1$;
(* $p_0 = p, p_1, \ldots, p_r, \ldots, p_a$ 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 $p_0, p_1, \ldots$
                  $\ldots, p_l, q_{b-1}, q_{b-2}, \ldots, q_0$;
          stop := true
    else $i := i + 1$; $r := a$; $s := b$
    fi
od
end
```

Figure 1: Constructing a t-spanner path from p to q in the skip list spanner.

Lemma 3 *Let $k > 8$ and $\theta = 2\pi/k$. For any pair p and q of points in S, this algorithm constructs a t-spanner path in $SLS(S,k)$ from p to q, for $t = 1/(\cos\theta - \sin\theta)$.*

Proof: 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. Then, by Lemma 1, we have $|p_{a+1}q_b| \leq |p_a q_b| - (\cos\theta - \sin\theta)|p_a p_{a+1}| < |p_a q_b|$ and $|q_{b+1}p_a| \leq |q_b p_a| - (\cos\theta - \sin\theta)|q_b q_{b+1}| < |q_b p_a|$. This proves that the algorithm terminates. Using the two given inequalities, it can be shown that it constructs a t-spanner path from p to q. ∎

Remark 1 Consider the t-spanner path $p_0 = p, p_1, \ldots, p_l = q_b, q_{b-1}, \ldots, q_0 = q$ that is computed by algorithm *walk*(p,q). In the full paper it is shown 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. 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 $i+1$. (By Remark 1, all points on this path are 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 $p'_r = p_r, p'_{r+1} = p_{r+1}, \ldots, p'_{r+l} = p_{r+l}$. 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 $p'_r, p'_{r+1}, \ldots, p'_m$ is completely determined by the points p and q and the sets $S_1, \ldots, S_i$, 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 $p'_r, p'_{r+1}, p'_{r+2}, \ldots$, 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 $p'_r, p'_{r+1}, \ldots, p'_{r+l-1}$ do not occur at level $i+1$, p'_{r+l} occurs at level $i+1$, and for all points of $S'_i := S_i \setminus \{p'_r, p'_{r+1}, \ldots, p'_{r+l}\}$ we have not yet flipped the coin. Let $q'_s = q_s, q'_{s+1}, q'_{s+2}, \ldots$ 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, $q'_s = q_s, q'_{s+1} = q_{s+1}, \ldots, q'_{s+t} = q_{s+t}$. By Remark 1, all points $p'_r, p'_{r+1}, \ldots, p'_{r+l}, q'_s, q'_{s+1}, \ldots, q'_{s+t-1}$ 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 $i+1$ with probability $1/2$. Since $q'_b \notin S_{i+1}$ for all b, $s \leq b \leq s+t-1$, it follows that

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

i.e.,

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

We have shown that, conditional on fixed subsets $S_1, S_2, \ldots, S_i$ 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 $S_1, \ldots, S_i$ only.

To summarize, we have shown that, conditional on fixed subsets $S_1, S_2, \ldots, S_i$, 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 $S_1, \ldots, S_i$, 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 [8].) Let N denote the number of edges on the t-spanner path from p to q that is constructed by the algorithm.

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. [8].) It is clear that the time for constructing this t-spanner path is proportional to $N + h$. Therefore, the running time of algorithm *walk*(p, q) is also bounded by $O(\log n)$ 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 both bounded by $O(\log n)$ with high probability. (See Observation 1.3.1 on page 10 of [8].) 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.

Let $k > 8$ be an integer, let $\theta = 2\pi/k$ and let $t = 1/(\cos\theta - \sin\theta)$. Then the skip list spanner $SLS(S, k)$ is the t-spanner for which parts 1., 2. and 3. of Theorem 1 hold.

4 Maintaining the skip list spanner

In this section, we consider the problem of maintaining the skip list spanner when points are inserted and deleted in S. 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, we can obtain polylogarithmic expected update time. For a detailed description of this model, see [8].

Consider a set V of points and a sequence of updates involving these points. For each i, let p_i denote the point of V that is involved in the i-th update, 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. If the update sequence is random in this model, then for each i (i) p_i is a random point of V, and (ii) V_i is a random subset of V of size n_i.

We first show how to maintain the graph $\Theta(S, k)$ under insertions and deletions. Among other things, we need a data structure solving the following query problem. Given any point $q \in \mathbb{R}^d \setminus S$, find all points p of S such that the graph $\Theta(S \cup \{q\}, k)$ contains an edge from p to q.

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 $q \in C_p$. Then the graph $\Theta(S \cup \{q\}, k)$ contains an edge from p to q iff (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 $l_{C,p}$.

This suggests the following data structure. Fix any cone C of $\mathcal{C}$. 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 any 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:

(i) 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;

(ii) A balanced binary search tree T_2^u for the points in the set $\{r_p \in C_p \cap S : p \in S^u$ and $\Theta(S, k)$ contains an edge from p to $r_p\}$. These points are stored in the leaves of the tree, sorted by their $(r_p)'_{d+1}$-coordinates.

Given this data structure, we can query it with any point $q \in \mathbb{R}^d \setminus S$, as follows. 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 $S \setminus \{q\}$ that are contained in the cone $-C_q$. For each of these nodes u, we report all points stored in its layer-$(d+1)$ tree T_1^u. Also, we walk along the leaves of its layer-$(d+1)$ tree T_2^u, from right to left, and report all points $p \in S^u$ for which $(r_p)'_{d+1} > q'_{d+1}$.

It follows from the above discussion that we report exactly the set of all points p such that $q \in C_p$ and the graph $\Theta(S \cup \{q\}, k)$ contains an edge from p to q.

The following data structure is used for maintaining the graph $\Theta(S, k)$:

(i) We store the graph $G = \Theta(S, k)$. With each point of S, we store a list of all points q of S such that (p, q) is an edge, and a list of all points r of S such that (r, p) is an edge.

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

(iii) For each cone C of $\mathcal{C}$, we store the above $(d+1)$-layer data structure T'_C for the points of S.

In the full paper, the complete insertion and deletion algorithms are given. Let D be the in-degree of the new point q in the graph $\Theta(S \cup \{q\}, k)$. Then, the insertion algorithm takes $O(((c/\theta)^{d-1} + D) \log^d n \log\log n)$ amortized time. (Here, dynamic fractional cascading [7] is used.) In a symmetric way, the amortized deletion time can be bounded by the same quantity, but now D is the in-degree of the point to be deleted before the operation. 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 4 *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. ■

Lemma 5 *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 structure of Lemma 5 for the points of S_i.

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 insert q into the Θ-graphs corresponding to the levels $1, 2, \ldots, l$. To delete a point q, we delete q from all Θ-graphs in which it occurs.

To analyze the update time, suppose we update the Θ-graph of level i. Since S_i is a random subset of S, which in turn is a random subset of V, Lemma 4 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 5 implies that the expected amortized update time of the skip list spanner is bounded by $O((c/\theta)^{d-1} \log^d n \log\log n)$ per random update.

Let $k > 8$ be an integer, let $\theta = 2\pi/k$ and let $t = 1/(\cos\theta - \sin\theta)$. Then the dynamic skip list spanner $SLS(S, k)$ is the t-spanner for which part 4. of Theorem 1 holds.

5 Deterministic Construction

In this section we consider deterministic approaches to constructing t-spanners of small diameter. As mentioned earlier, the construction is not based on a derandomization of the construction presented in the previous section, but rather on a well-separated pair decomposition of the point set. Callahan and Kosaraju introduced this structure as a mechanism for solving a number of problems on point sets in d-dimensional space [2, 3], and in [3] they show that such a decomposition can be used to compute t-spanners. (Vaidya gave a similar construction, with a slightly slower running time [15].) They did not consider the issues of the diameter of the resulting spanners, or the time needed to answer path queries (and indeed, the t-spanner paths that result may have as many as $n-1$ edges). In this section we present a proof of Theorem 2, by showing that low diameter spanners can be constructed with the same time and space bounds.

We begin with a short review of relevant definitions from [2]. Let S be a set of n points in $\mathbb{R}^d$. Let $s > 0$ be a real value called the *separation*. We say that two point-sets A and B are *well-separated* if and only if they can each be enclosed in d-spheres of radius r, whose distance of closest approach is at least sr. A *well-separated pair decomposition* is a set of pairs of nonempty subsets of S, $\{\{A_1, B_1\}, \{A_2, B_2\}, \dots \{A_m, B_m\}\}$, such that

(1) $A_i \cap B_i = \emptyset$, for all $i = 1, 2, \dots, m$.

(2) For each unordered pairs of distinct elements $\{a, b\}$ of S, there exists a unique pair $\{A_i, B_i\}$ in the decomposition such that $a \in A_i$ and $b \in B_i$.

(3) A_i and B_i are well-separated, for all $i = 1, 2, \dots, m$.

In [2], Callahan and Kosaraju show that a well-separated pair decomposition of size $O(s^d n)$ can be computed for S in time $O(n \log n + s^d n)$. In [3], they show that a t-spanner can be computed from the decomposition. We present a simplified explanation of the construction, which suffices for our purposes. Given S and $t > 1$, construct a well-separated pair decomposition of S with separation $s = 4/(t-1)$. For each set A_i (or B_i) in the decomposition, select a representative point $\mathit{rep}(A_i) \in A_i$. For each pair $\{A_i, B_i\}$ in the decomposition, include in the spanner the undirected edge $\{\mathit{rep}(A_i), \mathit{rep}(B_i)\}$ between their respective representatives.

Given a path query (a, b), we construct a path from a to b in the spanner as follows.

(1) If $a = b$, then return the empty path.

(2) Otherwise, there is a unique pair of sets $\{A_i, B_i\}$ with $a \in A_i$ and $b \in B_i$ in the decomposition. Construct a path π_a recursively from a to the representative $\mathit{rep}(A_i)$. Construct a path π_b recursively from b to the representative $\mathit{rep}(B_i)$.

(3) Return the concatenation of π_a, the edge $(\mathit{rep}(A_i), \mathit{rep}(B_i))$, and the reversal of π_b.

Recall that the *weight* of a path is the sum of the Euclidean lengths of its edges. It is an easy induction proof, that the resulting path is of weight at most $t|ab|$. (See, for example, Lemma 4.1 in [3] with $\epsilon = 4/s$. Any point in A_i (and B_i) may be used as a representative).

As it stands, this construction does not provide upper bounds on the number of edges in each path, but we can establish such bounds by a careful choice of the representative points. Before presenting details of the construction of the low diameter spanner, we need to consider the structure of the well-separated pair decomposition in greater detail.

The well-separated pair decomposition for the point set S is derived from a binary tree T, called the *fair-split tree*. The tree T is of size $O(n)$, but it need not be balanced, and may have depth as large as $\Omega(n)$. The leaves of the fair split tree correspond 1–1 with the points of S, and each internal node of the tree is associated with d-dimensional rectangle that contains all the points contained within the leaves of the subtree rooted at this node. We refer to nodes by the associated subset of points of S. Each pair $\{A_i, B_i\}$ in the well-separated pair decomposition is represented as an unordered pair of nodes in T. Since each node of T is associated with a subset of S, this node can also be associated with the representative point for this subset.

The structure of T will determine the choice of representative points. For each internal node u in T, consider the two subtrees rooted at the children of u. The edge going to the subtree having the larger number of leaves is labeled as *heavy* and the other is labeled as *light* (ties may be broken either way). Since every internal node has exactly one heavy edge from one of its children, there is a unique maximal chain of heavy edges leading up from each leaf in T. These chains partition the nodes of T into n subsets, one associated with each leaf. For each node u in T, let $l(u)$ denote the leaf whose chain contains u. The representative point associated with u is the point associated with $l(u)$. The resulting assignment of representative points is called the *heavy-subtree assignment*. Observe that this provides a valid assignment of representative points, because each node is associated with a leaf contained within its subtree, and hence each subset is associated with a member point. We can now present the main result of this choice of representatives.

Lemma 6 *If we apply the above path finding algorithm to the t-spanner that results from the heavy-subtree assignment of representative points, then the path generated has at most $2 \log n$ edges.*

Proof: Define the *length* of a path to be the number of edges in the path. Consider any two points $a, b \in S$, and let $\{A_1, B_1\}$ denote the pair in the well-separated pair decomposition containing this pair. Since these sets are disjoint, $|A_1| + |B_1| \leq n$. It suffices to show that the path from a to the representative of its ancestor, $\mathit{rep}(A_1)$, has length at most $\log |A_1|$ (and a similar result will follow for b).

Let $a_1 = \mathit{rep}(A_1)$. If $a = a_1$, then the length of the path is zero, and the conclusion follows trivially. Otherwise, consider the pair $\{A_2, B_2\}$ in the decomposition such that $a \in A_2$ and $a_1 \in B_2$. By the structure of the well-separated pair decomposition, A_2 and B_2 are both disjoint subsets of A_1, and hence both of the corresponding nodes in T are descendents of A_1, and neither is a descendent of the other. We claim that $a_1 = \mathit{rep}(B_2)$. This is because a_1 is a descendent of B_2, and hence the maximal chain leading from the leaf a_1 to B_2's ancestor, A_1, must pass through B_2. Thus, the path from *rep*(A_1) to *rep*(B_2) has length zero.

Next, we claim that $|A_2| \leq |A_1|/2$. The reason is that otherwise, all the edges between A_2 and A_1 in the fair split tree would have to be heavy edges, implying that the representative point for A_1 would be a member of A_2, contradicting the disjointness of A_2 and B_2. By induction, the length of the path from a to the representative of A_2 is at most $\log|A_2|$, which is at most $\log|A_1| - 1$. Adding to this the single edge from A_2's representative to a_1 (B_2's representative) gives a total path length of at most $\log|A_1|$, as desired. ■

We claim that we can implement the path finding algorithm in $O(\log^2 n)$ time. We have argued that the algorithm outputs $O(\log n)$ edges, and hence it suffices to show that each recursive invocation can be implemented in $O(\log n)$ time. The only nontrivial step is that of determining for a pair of distinct points, $\{a, b\} \in S$, the pair $\{A, B\}$ in the well-separated pair decomposition containing this pair. Call this a *pair query.* To see how to answer these queries, we need to take a closer look at the construction of the well-separated pair decomposition first.

Following Callahan and Kosaraju's development in [2], each internal node u of the fair-split tree T, has exactly two children, called A and B. The recursive procedure for constructing a well-separated pair decomposition of the set of pairs in the cross-product $A \times B$, can be viewed as a binary computation tree, $C(A, B)$, associated with u. Each node of this computation tree is associated with a pair $\{A', B'\}$, where $A' \subseteq A$ and $B' \subseteq B$. If this pair is well-separated, then this is a leaf in the computation tree, and otherwise it is an internal node (see [2] for details). The size of all the computation trees is asymptotically equal to the size of the final well-separated pair decomposition.

Although computation trees are not necessarily balanced, each computation tree can be represented in balanced form by computing a *centroid decomposition* of this tree. The centroid decomposition tree comes about by recursively finding a *centroid edge*, that is, an edge whose removal splits the tree into two connected subtrees of size at most a factor of roughly 2/3 the size of the original tree. Then each of these subtrees is recursively decomposed. This results in a binary decomposition of depth $O(\log m)$, where m is the size of the computation tree. Each internal node of the resulting binary tree is associated with an edge of the computation tree. We assume that each subset in the computation tree is associated with its enclosing rectangle. The tree will be searched in the usual top-down manner. From the disjointness of these rectangles (see [2]), it follows that for a given pair, $\{A, B\}$, we can determine in $O(d)$ time whether $a \in A$ and $b \in B$ (or vice versa). This is done by testing membership in the corresponding rectangles. As a consequence, given any edge of the centroid decomposition, in $O(d)$ time we can determine in which subtree of the computation tree to continue the search for the pair (a, b). Details will be given in the full paper.

We also augment the fair-split tree with any data structure for answering lowest common ancestor queries in $O(\log n)$ time (e.g. [14] is more than sufficient). These augmentations can all be computed within the same asymptotic time bound as the construction of the well-separated pair decomposition.

To answer a pair query $\{a, b\}$ in $O(\log n)$ time, we begin by finding the lowest common ancestor of a and b in the fair-split tree. We know that the desired pair will lie within the computation tree for this node. Next we search the computation tree (using its centroid decomposition) for the desired pair. Since each such query to the centroid decomposition tree can be answered in $O(d)$ time, and the size of the computation tree is at most $(c/(t-1))^d n$, the time for this search is $O(d \log n + d^2 \log(c/(t-1))) = O(\log n)$. Repeating this for each of the $O(\log n)$ edges of the path, yields the desired complexity.

In the current spanner, it takes $O(\log^2 n)$ time to answer a path query. We can reduce this to $O(\log n)$, by using an associated data structure of size $O(n \log n)$.

We know that there are at most $O(\log n)$ distinct representative points on the path from a leaf to the root of the box decomposition tree. For each point x, we maintain the list of these $O(\log n)$ representative points, say $x_1, x_2, \ldots, x_r$. Here, x_1 equals the representative point of the root of the tree and x_r equals the representative point of the leaf, namely x. Each point x_i in this list keeps one pointer to another point x_j. The point x_j is selected according to the following rule. Let $\{X, Y\}$ denote the well-separated pair for the pair of points x and x_i, such that x belongs to X and x_i belongs to Y. Then point x_j is the representative point of X. (Since node X must be an ancestor of leaf x, the representative point of X is in the above list.)

To compute the t-spanner path between a and b, we proceed as follows. First we find the well separated pair $\{A_1, B_1\}$ separating a and b. This can be done in $O(\log n)$ time using the previous method. Let a_1 (resp. b_1) be the representative point of A_1 (resp. B_1). Then we have to construct paths between a and a_1, and between b and b_1. To construct the path between a and a_1, we look at the list stored with a. We can find a_1 in this list in $O(\log n)$ time. To compute the path between a and a_1, we simply follow the pointers in this list, starting from a_1 and ending when he have reached a. This computes the same path as above.

6 Conclusion

We have presented randomized and deterministic algorithms for constructing t-spanners of $O(n)$ edges and $O(\log n)$ diameter. The randomized spanner takes $O(n \log^{d-1} n)$ time to construct while the deterministic spanner can be constructed faster in $O(n \log n)$ time. After augmenting these spanners with an additional $O(n)$ size data structure we can efficiently determine the spanner path between any two given points. These path queries can be answered in $O(\log n)$ time for the randomized spanner and $O(\log^2 n)$ time for the deterministic spanner. We have also shown how to maintain the randomized spanner in expected polylogarithmic time per random update.

Our work suggests many interesting open problems. Perhaps the most important question is whether there exist t-spanners of $O(n)$ edges having $o(\log n)$ diameter in high dimensions (in one dimension, there do) and how efficiently these can be constructed. Finally, it would be interesting to provide dynamic spanners that can be efficiently updated for arbitrary updates.

References

[1] S. Arya and M. Smid. *Efficient construction of a bounded degree spanner with low weight.* Proc. 2nd ESA, LNCS, Springer-Verlag, Berlin, 1994.

[2] P.B. Callahan and S. Rao Kosaraju. *A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields.* Proc. 24th Annu. ACM Sympos. Theory Comput. 1992, pp. 546–556.

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

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

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

[6] G.S. Lueker. *A data structure for orthogonal range queries.* Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci., 1978, pp. 28–34.

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

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

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

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

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

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

[13] J.S. Salowe. *On Euclidean spanner graphs with small degree.* Proc. 8th Annu. ACM Sympos. Comput. Geom., 1992, pp. 186–191.

[14] B. Schieber and U. Vishkin. *On finding lowest common ancestors: Simplifications and parallelization.* SIAM J. Comput. **17** (1988), pp. 1253–1262.

[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.