Efficient Construction of a Bounded Degree Spanner with Low Weight

Sunil Arya* Michiel Smid*

May 2, 1996

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 graph having the points of S as its vertices such that for any pair p, q of points there is a path between them of length at most t times the Euclidean distance between p and q.

An efficient implementation of a greedy algorithm is given that constructs a t-spanner having bounded degree such that the total length of all its edges is bounded by $O(\log n)$ times the length of a minimum spanning tree for S. The algorithm has running time $O(n \log^d n)$.

Applying recent results of Das, Narasimhan and Salowe to this t-spanner gives an $O(n \log^d n)$ time algorithm for constructing a t-spanner having bounded degree and whose total edge length is proportional to the length of a minimum spanning tree for S. Previously, no $o(n^2)$ time algorithms were known for constructing a t-spanner of bounded degree.

In the final part of the paper, an application to the problem of distance enumeration is given.

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 graph having the points of S as its vertices such that for any pair p, q of points there is a path between them having total length at most t times the Euclidean distance between p and q.

Much research has been recently done on the problem of efficiently constructing spanners that satisfy additional constraints. Quantities that are of interest are the number of edges in the spanner, the maximum degree, and the weight, which is defined as the total length of all edges. It is clear that each t-spanner must have at least $n-1$ edges. Also, the weight must be at least equal to the weight of a minimum spanning tree for S. We denote the latter by $wt(MST)$.

We give a brief overview of known results on spanner constructions. See also Table 1.

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

degree	weight	time	reference
$O(1)$	$\star$	$O(n^2 \log n)$	[14, 17]
$O(1)$	$O(wt(MST))$	$O(n^3 \log n)$	[2, 4, 6]
$\star$	$O(wt(MST))$	$O(n \log^2 n)$	[5]
$O(1)$	$O(wt(MST))$	$O(n \log^d n)$	this paper

Table 1: Results for constructing a t-spanner for a set of n points in $\mathbb{R}^d$. All constant factors depend on t and d. A $\star$ indicates that the quantity can be very large.

Feder and Nisan gave a simple $O(n^2 \log n)$ time algorithm for constructing spanners with bounded degree. (See [14, 17].) However, these spanners can have a very large weight.

Chandra et al.[2] present a path greedy algorithm for constructing a spanner with bounded degree. Recent results of Das et al. [4, 6] prove that this spanner has weight $O(wt(MST))$. The algorithm of [2] has running time $O(n^3 \log n)$.

Das and Narasimhan [5] present a fast implementation of a variant of the path greedy algorithm using graph clustering techniques that runs in $O(n \log^2 n)$ time. Again applying the results of [4, 6] shows that the resulting spanner has weight $O(wt(MST))$. Its degree, however, can be very large.

In [14], it is shown that there exists a t such that a t-spanner of degree four can be constructed. In [3], the analogous result is proved for degree-3 spanners. Hence, there has been much interest in spanners of small degree.

In this paper, we present an $O(n \log^d n)$ time algorithm for constructing a bounded degree spanner having weight $O(wt(MST))$. The importance of this result lies in the fact that this is the first algorithm that constructs such a spanner in $o(n^2)$ time. In fact, it is even the first $o(n^2)$ time algorithm for constructing a spanner of bounded degree.

A set of directed edges is said to possess the *gap property* if the sources and sinks of any two edges in the set are separated by a distance at least proportional to the length of the shorter of the two edges. Chandra et al.[2] have shown that if the edges of a graph can be partitioned into a constant number of subsets such that within each subset the gap property holds, then the weight of the graph is bounded by $O(wt(MST) \log n)$ and it has bounded degree.

The idea of the path greedy algorithm is to consider pairs of points in order of increasing distance, adding an edge (p, q) if and only if the partial spanner built until then does not already contain a path between p and q of length at most t times the distance between p and q. It is obvious that the resulting graph is a t-spanner. Additionally, Chandra et al. prove that the edges in this spanner can be partitioned into a constant number of subsets such that each subset satisfies the gap property. Hence, it has bounded degree and weight $O(wt(MST) \log n)$.

In this paper we show that we can in some sense reverse the emphasis of this greedy strategy. We consider pairs of points in order of increasing distance, adding an edge (p, q) if and only if it does not violate the gap property. More precisely, the edges of the partial spanner built until then can be partitioned into a constant number of subsets such that within each subset the gap property holds. (We call this the *gap* *greedy* strategy). It is obvious that the resulting graph has weight $O(wt(MST)\log n)$ and bounded degree. We are able to show that this graph is also a t-spanner.

The major advantage of the gap greedy approach is that we can give an efficient implementation for a minor variant of it that runs in $O(n\log^d n)$ time. One of the main ideas is that we do not have to consider the pairs in increasing order of their exact distance. It suffices to consider them in increasing order of their approximate distance. If an edge (p,q) is added to the spanner, then several points become "forbidden" as source or destination end points for later edges. Using range trees, we can implicitly maintain the non-forbidden points and their approximate distances. In each iteration, we then take a pair p,q of non-forbidden points having "minimal approximate" distance, add this pair as an edge to the graph, determine the points that become forbidden and remove the approximate distances they induce from the data structure.

Hence, in $O(n\log^d n)$ time, we construct a spanner of bounded degree having weight $O(wt(MST)\log n)$. By applying the results of [5] to this spanner, we get an $O(n\log^d n)$ time algorithm for constructing a spanner of bounded degree with weight $O(wt(MST))$.

In the final part of this paper we show how spanners can be used to enumerate distances efficiently. More precisely, given the spanner that results from our algorithm, we can enumerate the k smallest distances in the set S in sorted order, in time $O(n + k\log k)$. The value of k need not be known at the start of the enumeration. We show similar results for enumerating approximate distances.

For the problem of enumerating the k smallest distances, the following was known. Salowe [13] and Lenhof and Smid [9] achieve $O(n\log n + k)$ time for any dimension, but in both algorithms, the value of k must be known in advance and the distances are not enumerated in sorted order. In the plane, Dickerson et al.[8] show that given the Delaunay triangulation, the k smallest distances can be enumerated in $O(n + k\log k)$ time. In this algorithm, the value of k need not be known in advance and the distances are enumerated in sorted order.

Hence our spanner can be regarded as an efficient data structure that can be used for distance enumeration.

The rest of this paper is organized as follows. In Section 2, we define the basic geometric notions that are used in the paper and prove the main lemmas that we will use in order to show that a graph is a spanner. In Section 3, we give the simple gap greedy algorithm. In Section 4, we introduce cones and define approximate distance functions based on them. Using these, we give a variant of the algorithm of Section 3. In Section 5, we give the efficient implementation of this variant. Section 6 gives the application of bounded degree spanners to the problem of distance enumeration. In Section 7, we conclude with some remarks and open problems.

2 Preliminaries

In this section, we introduce the basic terminology and recall and prove some facts that will be used in the rest of the paper.

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 only consider directed graphs. 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 $|pq|$. We denote by $|pq|_\infty$ the L_∞-distance between p and q, i.e., $|pq|_\infty = \max_{1\leq i\leq d} |p_i - q_i|$.

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.

Remark 1 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.

Given a t-spanner $G = (S,E)$ and a point p of S, we define the *degree* of p as the sum of its *in-degree* and its *out-degree* in G. Define the *weight* of a set of edges as the sum of the weights of all its elements. The *weight of a t-spanner* is the weight of its edge set.

In order to estimate the weight of a t-spanner, Chandra et al.[2] introduced the *gap property*: Let $w \geq 0$. A set E of directed edges satisfies the w-gap property if for any two edges (p,q) and (r,s) in E, we have

$$\min(|pr|,|qs|) > w \cdot \min(|pq|,|rs|),$$

i.e., the sources and sinks of any two edges are separated by at least w times the weight of the shorter edge. Clearly, this implies that no two edges of E share a source, and no two edges share a sink.

Lemma 1 (Chandra et al.[2]) *Let E be a set of directed edges that satisfies the w-gap property. If $w \geq 0$, then no two edges share a source, and no two edges share a sink. Further, if $w > 0$, then the weight of E is $O((1/w)\log n)$ times the weight of a minimum spanning tree for S.*

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.) Then 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 $angle(p,q)$.

The following lemma enables us to prove that a graph is a t-spanner. Its proof is closely related to the proof of Lemma 4.1 in Chandra et al.[2]. Intuitively the lemma says that a graph is a spanner if for any edge e missing from the graph there is a similarly-directed edge e' close by (relative to the length of e') with length not much greater than e.

Lemma 2 *Let t, θ and w be real numbers such that $0 < \theta < \pi/4$, $0 \leq w < (\cos\theta - \sin\theta)/2$ and $t \geq 1/(\cos\theta - \sin\theta - 2w)$. Let S be a set of points in $\mathbb{R}^d$ and let $G = (S,E)$ be a directed graph such that the following holds. For any two points p and q of S there is an edge $(r,s) \in E$, such that*

1. *angle*$(q-p, s-r) \le \theta$, $|rs| \le |pq|/\cos\theta$ *and* $|pr| \le w|rs|$,

2. *or angle*$(p-q, r-s) \le \theta$, $|rs| \le |pq|/\cos\theta$ *and* $|qs| \le w|rs|$.

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

Proof: We use induction on the rank of the interpoint distance. Let p, q be any pair of points in S. If $p = q$, then there is nothing to show. So assume $p \neq q$. Let (r, s) be the edge guaranteed by the lemma. We will prove that (i) $|pr| < |pq|$, (ii) $|sq| < |pq|$, and (iii) there is a t-spanner path from p to q.

Assume that edge (r, s) satisfies condition 1. (The case that condition 2. holds can be treated by a symmetric argument.) Since $|rs| \le |pq|/\cos\theta$ and $0 < \theta < \pi/4$, we have $|rs| < |pq| \cdot \sqrt{2}$. Also, since $w < 1/2$ and $|pr| \le w|rs|$, we have $|pr| < |rs|/2$. Combining this gives $|pr| < |pq| \cdot \sqrt{2}/2 < |pq|$, which proves (i).

To prove (ii) and (iii) we need to consider two cases. Let l be the ray that emanates from r and that has the same direction as the vector $\overrightarrow{pq}$. Let v be the point on l such that $|rv| = |pq|$. Note that $|pr| = |vq|$. Let u be the orthogonal projection of s onto l. Let H be the two-dimensional plane that contains the ray l and the point s. Then the points r, s, u and v are all contained in H. Let α be the angle between $\overrightarrow{rs}$ and l. Then $\alpha = angle(q-p, s-r) \le \theta$, $\sin\alpha = |su|/|rs|$ and $\cos\alpha = |ru|/|rs|$. The two cases depend on whether $|ru| \le |rv|$ or $|ru| > |rv|$. (See Figure 1.)

Case 1: $|ru| \le |rv|$.

To show that $|sq| < |pq|$, we apply the triangle inequality and simplify:

$$\begin{aligned} |sq| &\le |su| + |uv| + |vq| \\ &= |su| + |rv| - |ru| + |vq| \\ &= |su| + |pq| - |ru| + |pr| \\ &\le |rs|\sin\alpha + |pq| - |rs|\cos\alpha + w|rs| \\ &\le |rs|\sin\theta + |pq| - |rs|\cos\theta + w|rs| \\ &= |pq| - |rs|(\cos\theta - \sin\theta - w). \end{aligned} \tag{1}$$

Since $w < (\cos\theta - \sin\theta)/2$, we conclude that $|sq| < |pq|$, which proves (ii).

It remains to prove (iii). By the induction hypothesis, there are t-spanner paths from p to r and from s to q. Consider the path that starts in p, takes the t-spanner path to r, then takes the edge to s, and finally takes the t-spanner path from s to q. The weight W of this path is at most equal to $t|pr| + |rs| + t|sq|$. Using (1), the assumptions of condition 1. and simplifying we get

$$\begin{aligned} W &\le tw|rs| + |rs| + t|pq| - t|rs|(\cos\theta - \sin\theta - w) \\ &= t|pq| - |rs|(t(\cos\theta - \sin\theta - 2w) - 1) \\ &\le t|pq|. \end{aligned}$$

Hence the graph G contains a t-spanner path from p to q.

Case 2: $|ru| > |rv|$.

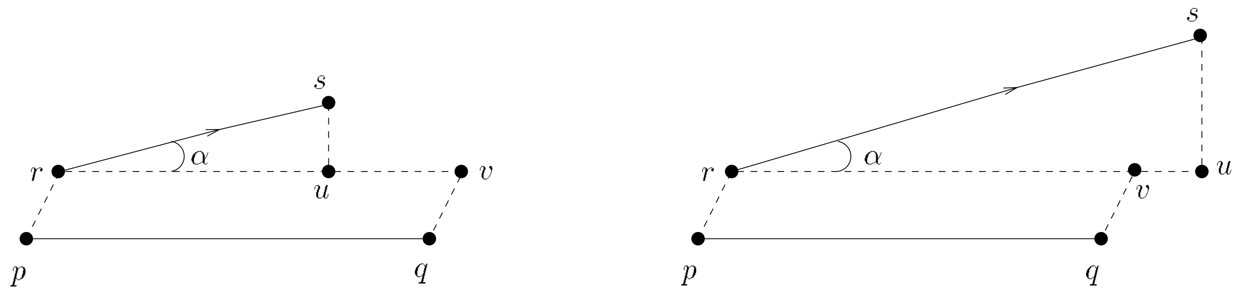

Figure 1: Cases 1 and 2 in Lemma 2.

As in Case 1, we apply the triangle inequality and simplify:

$$\begin{aligned}
|sq| &\leq |su| + |uv| + |vq| \\
&= |su| + |ru| - |rv| + |vq| \\
&= |rs|\sin\alpha + |rs|\cos\alpha - |pq| + |pr| \\
&\leq |rs|(\sin\theta + \cos\theta + w) - |pq| \\
&\leq |rs|(\sin\theta + w) \qquad (2) \\
&\leq \frac{|pq|}{\cos\theta}\left(\sin\theta + \frac{\cos\theta - \sin\theta}{2}\right) \\
&= \frac{1}{2}|pq|(1 + \tan\theta).
\end{aligned}$$

Since $0 < \theta < \pi/4$, we have $\tan\theta < 1$. Therefore, $|sq| < |pq|$, which proves (ii).

As in Case 1, we prove that the path formed by combining the t-spanner path from p to r, followed by the edge (r, s), followed by the t-spanner path from s to q, is a t-spanner path from p to q. This will prove (iii) and complete the proof of the lemma. Let W denote the weight of this path. Then $W \leq t|pr| + |rs| + t|sq|$. Using (2), the assumptions of condition 1. and simplifying we get

$$\begin{aligned}
W &\leq tw|rs| + |rs| + t|rs|(\sin\theta + w) \\
&= t|pq| - t|pq| + |rs|(t(\sin\theta + 2w) + 1) \\
&\leq t|pq| - t|rs|\cos\theta + |rs|(t(\sin\theta + 2w) + 1) \\
&= t|pq| - |rs|(t(\cos\theta - \sin\theta - 2w) - 1) \\
&\leq t|pq|,
\end{aligned}$$

i.e., there is a t-spanner path in G from p to q. ■

Remark 2 Given $t > 1$, let w and θ be assigned any values consistent with the expressions $0 < \theta < \pi/4$, $0 \leq w < (\cos\theta - \sin\theta)/2$ and $t \geq 1/(\cos\theta - \sin\theta - 2w)$. The undirected spanner built by the path greedy algorithm (see [2]) may be regarded as a directed spanner as indicated in Remark 1. It has the following property: Given any two edges (p,q) and (r,s) in the spanner, if the angle between them is at most θ, then they satisfy the w-gap property.

To show that this is true, assume w.l.o.g. that edge (r,s) was added first to the spanner. Then $|rs| \leq |pq|$. For the sake of contradiction, assume that the edges (p,q) and (r,s) do not satisfy the w-gap property. Then $|pr| \leq w|rs|$ or $|qs| \leq w|rs|$. Assume first that $|pr| \leq w|rs|$. From the proof of Lemma 2, we know that $|pr| < |pq|$ and $|sq| < |pq|$. Consider the moment when (p,q) is added to the spanner. Then the pairs (p,r) and (s,q) have been tested already, so there are t-spanner paths from p to r and from s to q. It follows from the proof of Lemma 2 that there must already be a t-spanner path from p to q and, therefore, edge (p,q) would not be added. The case $|qs| \leq w|rs|$ can be treated in a similar way.

Thus the path greedy spanner possesses the w-gap property for any pair of edges with angle at most θ, such that w and θ are consistent with the above expressions.

3 A greedy algorithm

In this section, we give a simple greedy algorithm for computing a spanner with bounded degree and low weight. In later sections, we modify this algorithm such that it can be implemented efficiently.

Let S be a set of n points in $\mathbb{R}^d$. The following algorithm *gap_greedy*(S,θ,w) constructs a spanner for S. If $w > 0$, then the edges of this spanner can be partitioned into a constant number of subsets, such that within each subset the w-gap property holds. This will guarantee that the spanner has bounded degree and low weight.

The algorithm considers all ordered pairs (p,q) of points in increasing order of their distances. The edge (p,q) is added to the graph iff there is no edge (r,s) in the current graph such that (p,q) and (r,s) have roughly the same direction and the sources p and r are close to each other, or (q,p) and (s,r) have roughly the same direction and the sources q and s are close to each other.

A formal description of our algorithm is given in Figure 2. We remark that for $w = 0$, this is exactly Feder and Nisan's algorithm. (See [14, 17].)

Lemma 3 *Algorithm gap_greedy*(S,θ,w) *computes a t-spanner for* $t = 1/(\cos\theta - \sin\theta - 2w)$.

Proof: Consider the edge set E that is constructed by the algorithm. We prove that this set satisfies the conditions of Lemma 2. This will prove that the graph (S,E) is a t-spanner.

Let (p,q) be any ordered pair of points of S. If (p,q) is an edge of E, then the conditions of Lemma 2 hold with $r = p$ and $s = q$. Assume that (p,q) is not contained in E. Consider the iteration where the pair (p,q) is inspected. We did not add (p,q) to E because this set contained an edge (r,s) such that (i) $angle(q-p, s-r) \leq \theta$ and $|pr| \leq w|rs|$, or (ii) $angle(p-q, r-s) \leq \theta$ and $|qs| \leq w|rs|$. Since (r,s) is contained in E at the moment when we inspect the pair (p,q), we must have $|rs| \leq |pq|$. This proves that $|rs| \leq |pq|/\cos\theta$. Hence condition 1. or 2. of Lemma 2 is satisfied. ■

Algorithm *gap_greedy*(S, θ, w)
(* S is a set of n points in $\mathbb{R}^d$, $0 < \theta < \pi/4$, $0 \leq w < (\cos\theta - \sin\theta)/2$ *)
begin
sort the $2\binom{n}{2}$ ordered pairs of points according to their distances (ties are broken arbitrarily) and store them in a list L;
$E := \emptyset$;
for all ordered pairs $(p, q) \in L$ (* visit pairs in sorted order *)
do $add := true$;
    for each edge $(r, s) \in E$
    do if $angle(q - p, s - r) \leq \theta$
        then $add := add \wedge (|pr| > w|rs|)$
        fi;
        if $angle(p - q, r - s) \leq \theta$
        then $add := add \wedge (|qs| > w|rs|)$
        fi
    od;
    if $add = true$ **then** $E := E \cup \{(p, q)\}$ **fi**
od;
output the set E
end

Figure 2: The greedy algorithm.

Lemma 4 *If $w \geq 0$, then algorithm gap_greedy(S, θ, w) computes a spanner of degree at most $O((c/\theta)^{d-1})$, for a suitable constant c. Further, if $w > 0$, then the weight of this spanner is bounded by $O((c/\theta)^{d-1}(1/w)\log n)$ times the weight of a minimum spanning tree for S.*

Proof: Consider any two edges (p, q) and (r, s) of the spanner (S, E) that is constructed by the algorithm. Assume that $angle(q-p, s-r) \leq \theta$. Then also $angle(p-q, r-s) \leq \theta$. If (r, s) was added to E before (p, q) then it follows from our algorithm that $|rs| \leq |pq|$, $|pr| > w|rs|$ and $|qs| > w|rs|$. If (p, q) was added before (r, s), then we have $|pq| \leq |rs|$, $|rp| > w|pq|$ and $|sq| > w|pq|$. Therefore, we must have $|pr| > w \cdot \min(|pq|, |rs|)$ and $|qs| > w \cdot \min(|pq|, |rs|)$, i.e., the w-gap property holds for the edges (p, q) and (r, s).

Consider a collection of $O((c/\theta)^{d-1})$ cones having their apex at the origin, one having angular diameter at most θ, such that the entire collection covers $\mathbb{R}^d$, for a suitable constant c. (In the next section, these notions are defined precisely.) Number these cones $C_1, C_2, \ldots, C_m$. Define $E_i := \{(p, q) \in E : q - p \in C_i\}$, $1 \leq i \leq m$. Then for each fixed i, the edges of E_i satisfy the w-gap property.

Lemma 1 implies that, if $w \geq 0$, no two edges of E_i share a source, and no two edges share a sink. Since the sets E_i, $1 \leq i \leq m$, partition E, it follows that each point of S has degree at most $2m = O((c/\theta)^{d-1})$. Also, if $w > 0$, then Lemma 1 implies that the total weight of E_i is bounded by $((1/w)\log n)$ times the weight of a minimum spanning tree for S. This proves that the total weight of the spanner is bounded by $((c/\theta)^{d-1}(1/w)\log n)$ times the weight of a minimum spanning tree for S. ∎

We briefly examine the question of what sorts of tradeoffs are possible between the three quantities of interest for spanners, namely, the spanner constant t, the degree, and the weight bound. For algorithm *gap_greedy*, we can assign any values to θ and w such that $0 < \theta < \pi/4$ and $0 \leq w < (\cos\theta - \sin\theta)/2$. Assume that $t > 1$ is given. If we want the best bound on the degree, then we must choose the largest possible cone angle. Thus we must choose θ such that $t = 1/(\cos\theta - \sin\theta)$. In this case, since $w = 0$, the weight bound can grow arbitrarily bad.

More interesting is the case of how to choose θ and w to achieve the best weight bound. Assume that we want a $(1+\epsilon)$-spanner where ϵ is a small constant. We saw in Lemma 4 that for $w > 0$, the spanner produced by algorithm *gap_greedy*(S, θ, w) has weight $O((c/\theta)^{d-1}(1/w)\log n)$ times the weight of a minimum spanning tree for S. Hence, in order to minimize the weight, we have to maximize $\theta^{d-1}w$. Since $t = 1 + \epsilon = 1/(\cos\theta - \sin\theta - 2w)$, we get

$$w = \frac{1}{2}\left(\cos\theta - \sin\theta - \frac{1}{1+\epsilon}\right).$$

If ϵ is small, then θ will also be small, and we can approximate the expression for w by

$$\begin{aligned} w &\sim \frac{1}{2}(1 - \theta - (1 - \epsilon)) \\ &\sim \frac{1}{2}(\epsilon - \theta). \end{aligned}$$

Therefore, we have to maximize $\theta^{d-1}(\epsilon - \theta)$. Differentiating and equating to zero we find that this expression is maximum for $\theta = (1 - 1/d)\epsilon$. This gives $w = \epsilon/(2d)$. The corresponding $(1+\epsilon)$-spanner has a weight that is bounded by $C \cdot \log n$ times the weight of a minimum spanning tree for S, where

$$C = O\left((c/\theta)^{d-1}(1/w)\right) = O\left(c^{d-1}\frac{2d}{\epsilon}(1-1/d)^{1-d}\epsilon^{1-d}\right) = O(dc^{d-1}\epsilon^{-d}).$$

Since algorithm *gap_greedy* inspects all pairs (p, q) of points explicitly, its running time is $\Omega(n^2)$. In the next section, we modify the algorithm. As we will see, the modified version can be implemented such that its running time is bounded by $O(n \log^d n)$.

4 Towards an efficient implementation

We start by introducing the notion of cones. A *(simplicial) cone* is the intersection of d halfspaces in $\mathbb{R}^d$. The intersection of the hyperplanes that bound these halfspaces is called the *apex* of the cone. We always assume that a cone is closed and that its apex is a point. 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 $0 < \theta < \pi/4$. 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 [18], 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 $\theta = \pi/k$, we just rotate the positive x-axis over angles of $i \cdot \theta$, $0 \leq i < 2k$. This gives $2k$ rays. Each wedge between two successive rays defines one 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.

After having introduced the terminology, we can modify algorithm *gap_greedy*. There are three major modifications. Consider again the formal description of the algorithm. First, we replace the condition "$angle(q-p, s-r) \leq \theta$" by "$q-p$ and $s-r$ are contained in the same cone of $\mathcal{C}$". Clearly, the latter condition implies the first one.

Second, we replace the condition "$|pr| > w|rs|$" by "$|pr|_\infty > (w/\sqrt{d})|rs|$", i.e., for the pair p, r, we switch from the Euclidean metric to the L_∞-metric. Note that all points r for which $|pr|_\infty \leq \delta$ are contained in the d-dimensional axes-parallel cube centered at p having sides of length 2δ. Using range trees, we can find such points r efficiently. (Finding all points r such that $|pr| \leq \delta'$ takes much more time.)

Third, instead of inspecting all pairs in increasing order of their distances, we inspect them in order of their *approximate* distances, to be defined below. As we will see, in this way we do not have to inspect all pairs explicitly.

Let C be any cone of $\mathcal{C}$ and let p and q be two points in $\mathbb{R}^d$. Let $C_p := C + p := \{x + p : x \in C\}$, i.e., C_p is the cone obtained by translating C such that its apex is at p. Similarly, let $l_{C,p} := l_C + p$. Then we define

$$\delta_C(p,q) := \begin{cases} \text{Euclidean distance between } p \text{ and} & \\ \text{the orthogonal projection of } q \text{ onto } l_{C,p} & \text{if } q \in C_p \\ \infty & \text{if } q \notin C_p. \end{cases}$$

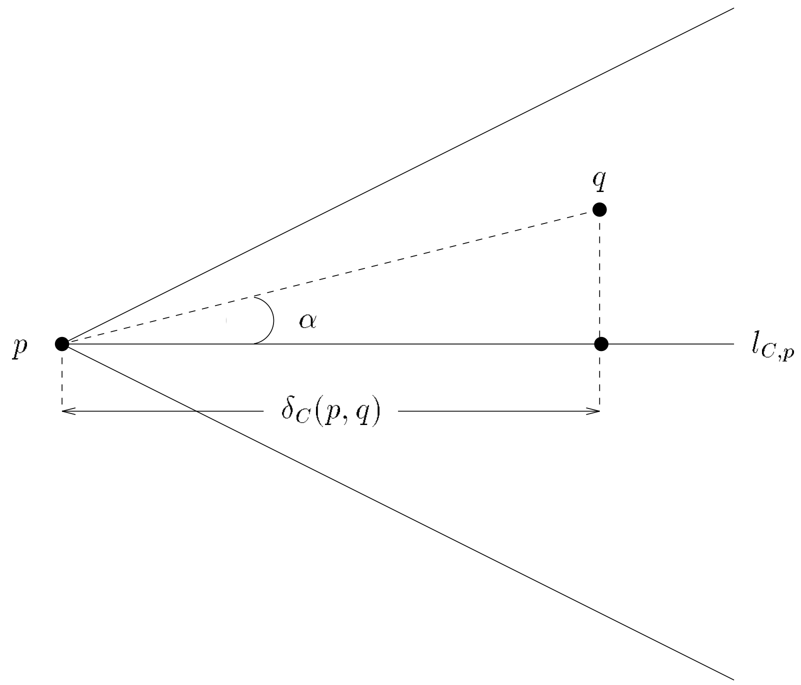

Figure 3: The approximate distance $\delta_C(p,q)$.

See Figure 3. Note that δ_C is not a metric. The following lemma says that $\delta_C(p,q)$ is a good approximation for the Euclidean distance between p and q, if $q \in C_p$.

Lemma 5 *Let p and q be points in $\mathbb{R}^d$. If $q \in C_p$, then $|pq| \cos\theta \leq \delta_C(p,q) \leq |pq|$.*

Proof: Assume that $q \in C_p$. Let H be the two-dimensional plane that contains the point q and the ray $l_{C,p}$. Note that H contains the vector $\overrightarrow{pq}$. Let α be the angle between $l_{C,p}$ and $\overrightarrow{pq}$. (See Figure 3.) Then, $0 \leq \alpha \leq \theta$ and $cos \alpha = \delta_C(p,q)/|pq|$. Hence, $\delta_C(p,q) = |pq| \cos\alpha \geq |pq| \cos\theta$ and $\delta_C(p,q) = |pq| \cos\alpha \leq |pq|$. ∎

```
Algorithm gap_greedy'(S, θ, w)
(* S is a set of n points in IR^d, 0 < θ < π/4, 0 ≤ w < (cos θ − sin θ)/2 *)
begin
for each cone C
do for each r ∈ S and s ∈ S do dist(r, s) := δ_C(r, s) od;
    E_C := ∅;
    while there are r ≠ s such that dist(r, s) < ∞
    do choose r ≠ s such that dist(r, s) is minimal;
        E_C := E_C ∪ {(r, s)};
        for each p ∈ S such that |pr|_∞ ≤ (w/√d)|rs|
        do for each q ∈ S do dist(p, q) := ∞ od
        od;
        for each q ∈ S such that |qs|_∞ ≤ (w/√d)|rs|
        do for each p ∈ S do dist(p, q) := ∞ od
        od
    od
od;
output the set E := ∪_C E_C
end
```

Figure 4: Towards an efficient implementation of the greedy algorithm.

Now we can give the modified algorithm. For each fixed cone C, we compute a set E_C of edges (p, q) such that $q - p \in C$. The union of all these sets will form the edge set of our final spanner.

Consider a cone C. We find the pair (r, s) of distinct points for which $\delta_C(r, s)$ is minimal and add the edge (r, s) to E_C. Having added the edge (r, s), we do not want to add edges (p, q) such that $q - p \in C$ and the distance between p and r is small. That is, after having added (r, s), all points p that are "close" to r should not occur as sources of edges that are added later. Similarly, after having added the edge (r, s), all points q that are "close" to s should not occur as sinks of edges that are added later.

That is, the addition of the edge (r, s) causes certain points to become "forbidden" as a source or a sink.

In the next iteration, we find the pair (r', s') of non-forbidden points for which $\delta_C(r', s')$ is minimal and proceed in the same way.

The formal algorithm is given in Figure 4. Consider the while-loop of this algorithm. If the edge (r, s) is added to E_C, then the value of $dist(r, s)$ is set to ∞ during the same iteration of this loop. That is, during each iteration, the number of pairs p, q for which $dist(p, q) < \infty$ strictly decreases. This proves that the while-loop terminates.

Lemma 6 *Algorithm gap_greedy'(S, θ, w) computes a t-spanner for $t = 1/(\cos\theta - \sin\theta - 2w)$.*

Proof: The proof is similar to that of Lemma 3. Consider the set E of edges that is computed by the algorithm. Let (p, q) be any ordered pair of points of S. If

$(p,q) \in E$, then the conditions of Lemma 2 hold. So, assume that (p,q) is not contained in E. Let C be a cone such that $q \in C_p$. Consider the iteration during which the edge set E_C is constructed. At the start of this iteration, $dist(p,q)$ has a finite value. Since the edge (p,q) is not added to E_C, the value of $dist(p,q)$ changes to ∞ during some iteration of the while-loop. Let (r,s) be the edge that is added to E_C during that iteration. At the start of it, we have $dist(r,s) \leq dist(p,q) < \infty$, $dist(r,s) = \delta_C(r,s)$ and $dist(p,q) = \delta_C(p,q)$. Moreover, we have $|pr|_\infty \leq (w/\sqrt{d})|rs|$ or $|qs|_\infty \leq (w/\sqrt{d})|rs|$. We consider these two cases separately.

Case 1: $|pr|_\infty \leq (w/\sqrt{d})|rs|$.

Then, $|pr| \leq \sqrt{d} \cdot |pr|_\infty \leq w|rs|$. Since $s - r$ and $q - p$ are both contained in C, we have $angle(q-p, s-r) \leq \theta$. By Lemma 5, we have $|rs| \leq \delta_C(r,s)/\cos\theta$ and $\delta_C(p,q) \leq |pq|$. Since $\delta_C(r,s) \leq \delta_C(p,q)$, we conclude that $|rs| \leq |pq|/\cos\theta$. Hence, condition 1. of Lemma 2 holds for the pair (p,q).

Case 2: $|qs|_\infty \leq (w/\sqrt{d})|rs|$.

It follows in the same way as in Case 1 that $|qs| \leq w|rs|$, $angle(p-q, r-s) \leq \theta$ and $|rs| \leq |pq|/\cos\theta$. Hence, condition 2. of Lemma 2 holds for the pair (p,q).

To summarize, we have shown that for each pair (p,q) of points one of the conditions of Lemma 2 is satisfied. This proves that the graph (S,E) is a t-spanner. ∎

Lemma 7 *If $w \geq 0$, then algorithm $gap\_greedy'(S,\theta,w)$ computes a spanner of degree at most $O((c/\theta)^{d-1})$, for a suitable constant c. Further, if $w > 0$, then the weight of this spanner is bounded by $O((c/\theta)^{d-1}(1/w)\log n)$ times the weight of a minimum spanning tree for S.*

Proof: Consider any cone C. We will prove that the edges of E_C satisfy the $(w/\sqrt{d})$-gap property. Then, the claim follows from Lemma 1.

Consider any two edges (p,q) and (r,s) of E_C. Assume w.l.o.g. that (r,s) was added to E_C before (p,q). Then we must have $|pr|_\infty > (w/\sqrt{d})|rs|$ and $|qs|_\infty > (w/\sqrt{d})|rs|$. (Otherwise, the algorithm would have set $dist(p,q) := \infty$. Therefore, the pair (p,q) would never have been chosen as a pair with minimal and finite $dist(\cdot,\cdot)$-value and, hence, the edge (p,q) would never have been added to E_C.) But this implies that

$$|pr| \geq |pr|_\infty > (w/\sqrt{d})|rs| \geq (w/\sqrt{d}) \cdot \min(|pq|,|rs|),$$

and

$$|qs| \geq |qs|_\infty > (w/\sqrt{d})|rs| \geq (w/\sqrt{d}) \cdot \min(|pq|,|rs|),$$

i.e., the $(w/\sqrt{d})$-gap property holds. ∎

5 An efficient implementation

In this section, we show how to implement algorithm gap_greedy' such that its running time is bounded by $O(n \log^d n)$. The main idea is to use range trees (see [11]) for maintaining the minimal value $dist(r,s)$ for all "non-forbidden" points r and s. The technique is related to the ones in [7, 16] for maintaining the closest pair or k-point cluster in a dynamically changing set of points.

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 5.)

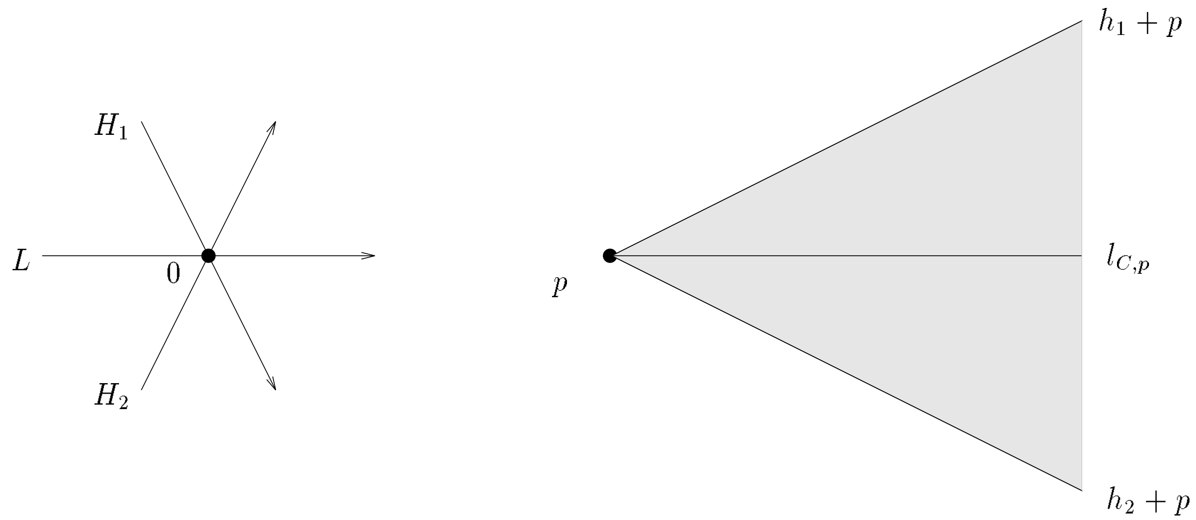

Figure 5: The directed lines H_1, H_2 and L, and the translated cone C_p.

Let p be any point in $\mathbb{R}^d$. We write the coordinates of p w.r.t. 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

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

If $q \in C_p$, then we have $\delta_C(p, q) = q'_{d+1} - p'_{d+1}$.

Let S be a set of n points in $\mathbb{R}^d$. During our algorithm we will maintain a data structure having the form of a $(d+1)$-layered range tree. This data structure depends on the cone C. We describe it in detail.

There is a balanced binary search tree storing the points of S in its leaves, sorted by their p'_1-coordinates. (Points with equal p'_1-coordinates are stored in lexicographical order.) Let v be any node of this tree and 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 p'_2-coordinates. (Points with equal p'_2-coordinates are stored such that the points $(p'_2, \ldots, p'_d)$ are in lexicographical order.) Any 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 p'_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 p'_d-coordinates. The binary tree that stores points sorted by their p'_i-coordinates is called a *layer-i tree*.

Before we can define the last layer of the data structure, we need to introduce some notation. Let u be any node of a layer-d tree. We inductively define a sequence $u_d, u_{d-1}, \ldots, u_1$ of nodes such that u_i belongs to a layer-i tree: Define $u_d = u$. Given u_i, walk to the root r of its layer-i tree. Then u_{i-1} is the node of the layer-$(i-1)$ tree that contains a pointer ro r. (See Figure 6.)

For $1 \leq i \leq d$, let x'_{ui} be the maximal p'_i-coordinate that is stored in the left subtree of node u_i. Let x_u be the point with coordinates $x'_{u1}, x'_{u2}, \ldots, x'_{ud}$. (Note that these coordinates are w.r.t. the "axes" $H_1, H_2, \ldots, H_d$. In general, x_u is not a point of S.)

Now we can define the $(d+1)$-st layer of the data structure. Consider again any node u of a layer-d tree. Let S_{ud} be the subset of S that is stored in the subtree of u. Consider the point x_u. Let $S^+_{u,d+1}$ be a subset of $\{p \in S_{ud} : p'_i \geq x'_{ui}, 1 \leq i \leq d\}$ and let $S^-_{u,d+1}$ be a subset of $\{p \in S_{ud} : p'_i \leq x'_{ui}, 1 \leq i \leq d\}$. (The algorithm determines the sets $S^+_{u,d+1}$ and $S^-_{u,d+1}$. For the description of the data structure, we assume that they are any subsets.) Note that all points of $S^+_{u,d+1}$ and $S^-_{u,d+1}$ are contained in the cones C_{x_u} and $-C_{x_u}$, respectively.

Node u of the layer-d tree contains pointers to

1. a list $L^+_{u,d+1}$ storing the points of $S^+_{u,d+1}$, sorted by their p'_{d+1}-coordinates,
2. a list $L^-_{u,d+1}$ storing the points of $S^-_{u,d+1}$, sorted by their p'_{d+1}-coordinates,
3. a variable $\eta_{d+1}(u)$ having value
$$\eta_{d+1}(u) = \min\{\delta_C(p,q) : p \in S^-_{u,d+1}, q \in S^+_{u,d+1}\},$$
4. and, in case, $\eta_{d+1}(u) < \infty$, a pair of points that realizes $\eta_{d+1}(u)$.

These two lists are called layer-$(d+1)$ lists. If $S^-_{u,d+1}$ or $S^+_{u,d+1}$ is empty, then $\eta_{d+1}(u) = \infty$. (In particular, this is true if u is a leaf.) Otherwise, we have $\eta_{d+1}(u) = \delta_C(p,q) = q'_{d+1} - p'_{d+1}$, where p and q are the maximal and minimal elements that are stored in the lists $L^-_{u,d+1}$ and $L^+_{u,d+1}$, respectively.

During our algorithm, the layer-i trees for $1 \leq i \leq d$ do not change, except for certain η-variables that are defined below. For each node u of a layer-d tree, the corresponding layer-$(d+1)$ lists initially store the sets $\{p \in S_{ud} : p'_i \geq x'_{ui}, 1 \leq i \leq d\}$ and $\{p \in S_{ud} : p'_i \leq x'_{ui}, 1 \leq i \leq d\}$. During the algorithm, elements will be deleted from these lists.

In order to speed up searching during the algorithm, we store all points of S in a dictionary. With each point p, we store

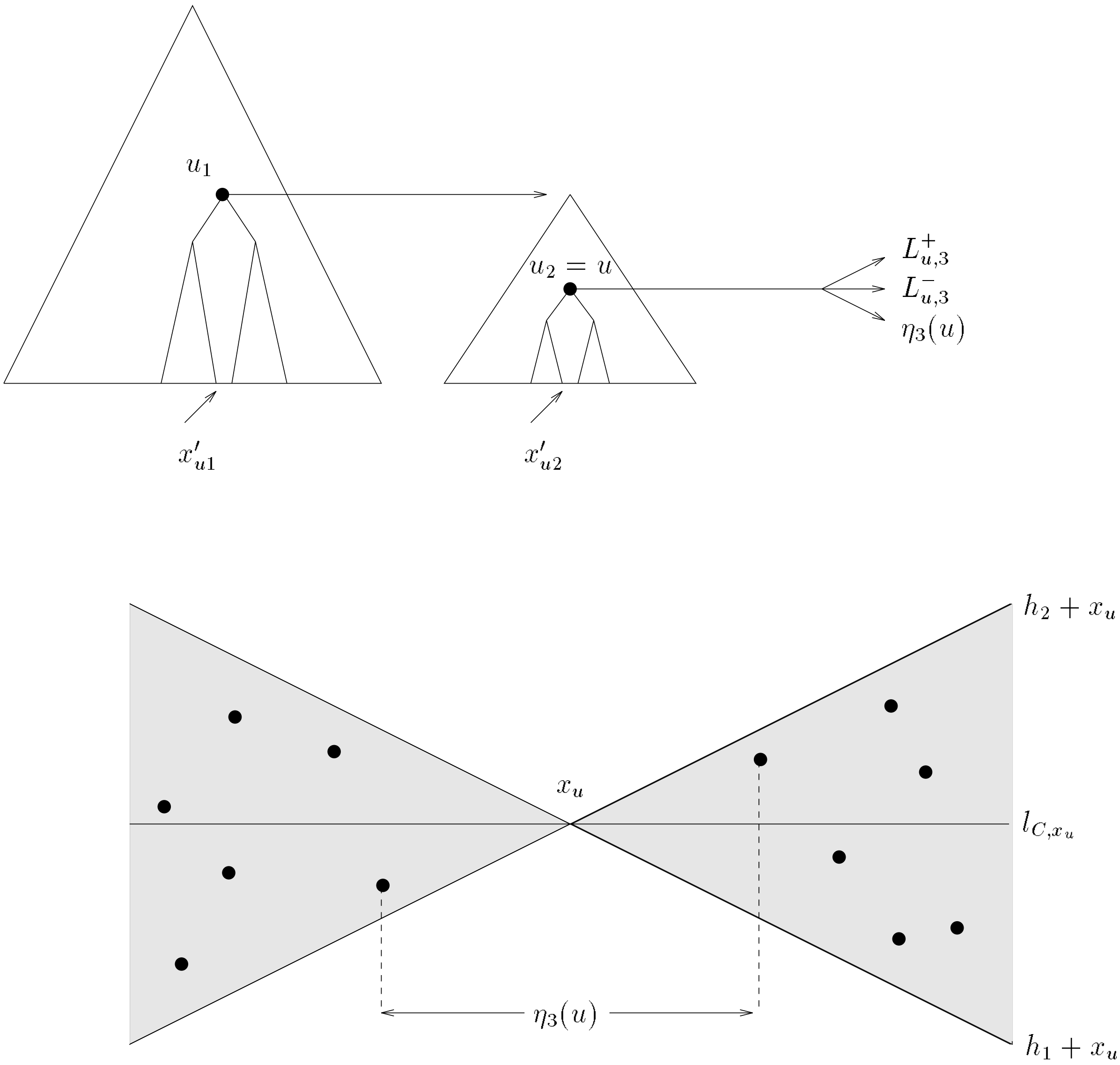

Figure 6: Illustration of the $(d+1)$-layered data structure for $d = 2$. The points in the cone C_{x_u} belong to the set $S^+_{u,3}$; those in the cone $-C_{x_u}$ belong to $S^-_{u,3}$.

---

1. a list of pointers to the positions of the occurrences of p in all lists $L^+_{u,d+1}$, and
2. a list of pointers to the positions of the occurrences of p in all lists $L^-_{u,d+1}$.

We are almost done with the description of the data structure. We saw that for each layer-$(d+1)$ structure there is a corresponding η_{d+1}-value. Let $1 \leq i \leq d$ and let v be any node of a layer-i tree. If v is a leaf then v stores a variable $\eta_i(v)$ having value ∞. If v is not a leaf, then let v_l and v_r be the left and right sons of v, respectively. Also, let $\eta_{i+1}(v)$ be the variable that is stored with the layer-$(i+1)$ structure that corresponds to v. Then node v stores a variable $\eta_i(v)$ having value

$$\eta_i(v) = \min(\eta_i(v_l), \eta_i(v_r), \eta_{i+1}(v)), \qquad (3)$$

and, in case $\eta_i(v) < \infty$, a pair of points that realizes $\eta_i(v)$.

This concludes the description of our $(d+1)$-layered data structure. Recall that the entire structure depends on the cone C.

Let q be any point of S. We can delete q from all lists $L^+_{u,d+1}$ in which it occurs and update the entire data structure, as follows: Search for q in the dictionary, and follow the pointers to the positions of all occurrences of q in the lists $L^+_{u,d+1}$. For each such u, do the following:

1. Delete q from $L^+_{u,d+1}$. If the list $L^-_{u,d+1}$ is empty, then we are done. Otherwise, let p be the maximal element of $L^-_{u,d+1}$. Go to 2.

2. If q was not the minimal element of $L^+_{u,d+1}$, then we are done. If q was the only element in its list, then we set $\eta_{d+1}(u) := \infty$. Otherwise, if q was not the only element in its list, then let r be the new minimal element of $L^+_{u,d+1}$. Then, set $\eta_{d+1}(u) := \delta_C(p, r) = r'_{d+1} - p'_{d+1}$, and store the pair (p, r).

Now, all layer-$(d+1)$ structures are updated correctly. To update the rest of the data structure, we do the following: We search for q in the layer-1 tree. For each node on the path, we search for q in the corresponding layer-2 tree, etc., until we have located q in all layer-d trees that contain this point. Then we walk back along all these paths. During the walk, we update the values $\eta_i(\cdot)$ according to (3).

It is easy to see that the entire operation can be performed in time $O(\log^d n)$. In a completely symmetric way, we can delete a point p from all lists $L^-_{u,d+1}$ and update the entire data structure.

Now we can give the efficient implementation of algorithm *gap_greedy*$'$. As before, we consider all cones separately. If C is the current cone, then we maintain besides the above $(d+1)$-layered data structure two d-layered range trees storing subsets of S according to their standard coordinates $p_1, p_2, \ldots, p_d$. Recall that such a range tree can be used to find all points that are contained in a d-dimensional rectangle having sides that are parallel to the standard axes. A complete description of the algorithm is given in Figure 7.

Lemma 8 *Consider the iteration for the cone C. During the execution of this iteration, if $\eta < \infty$, then*

$$\eta = \min\{\delta_C(p, q) : p \in RT_{source}, q \in RT_{sink}, p \neq q\}.$$

Proof: Since all η_i-variables, $1 \leq i \leq d+1$, either have value ∞ or $\delta_C(p, q)$ for some $p \in RT_{source}$ and $q \in RT_{sink}$, it is clear that

$$\eta \geq \min\{\delta_C(p, q) : p \in RT_{source}, q \in RT_{sink}, p \neq q\}. \tag{4}$$

If RT_{source} or RT_{sink} is empty, then $\eta = \infty$, which is a contradiction to our assumption that $\eta < \infty$. Hence, both these structures are non-empty. Let $r \in RT_{source}$ and $s \in RT_{sink}$ such that

$$\delta_C(r, s) = \min\{\delta_C(p, q) : p \in RT_{source}, q \in RT_{sink}, p \neq q\}.$$

Algorithm *gap_greedy''*(S, θ, w)
(* S is a set of n points in $\mathbb{R}^d$, $0 < \theta < \pi/4$, $0 \leq w < (\cos\theta - \sin\theta)/2$ *)
begin
for each cone C
do store the points of S in the $(d+1)$-layered data structure T defined above;
  the two layer-$(d+1)$ lists of each node u of each layer-d tree of T store
  the sets $S^-_{u,d+1} = \{p \in S_{ud} : p'_i \leq x'_{ui}, 1 \leq i \leq d\}$ and
  $S^+_{u,d+1} = \{p \in S_{ud} : p'_i \geq x'_{ui}, 1 \leq i \leq d\}$;
  store the points of S in two d-layered range trees RT_{source} and RT_{sink}
  according to their standard coordinates;
  $E_C := \emptyset$;
  $\eta :=$ value stored with the root of the layer-1 tree of T;
  while $\eta < \infty$
  do let (r, s) be a pair such that $\eta = \delta_C(r, s)$;
    $E_C := E_C \cup \{(r, s)\}$;
    for each $p \in RT_{source}$ such that $|pr|_\infty \leq (w/\sqrt{d})|rs|$
    do delete p from RT_{source};
      delete p from all lists $L^-_{u,d+1}$, and update T and
      η as described in the text
    od;
    for each $q \in RT_{sink}$ such that $|qs|_\infty \leq (w/\sqrt{d})|rs|$
    do delete q from RT_{sink};
      delete q from all lists $L^+_{u,d+1}$, and update T and
      η as described in the text
    od
  od
od;
output the set $E := \bigcup_C E_C$
end

Figure 7: The efficient implementation of the greedy algorithm.

If we can show that there is a node u in some layer-d tree of T such that $\eta_{d+1}(u) = \delta_C(r,s)$, then we must have

$$\eta \leq \min\{\delta_C(p,q) : p \in RT_{source}, q \in RT_{sink}, p \neq q\}.$$

This will prove the lemma.

Consider the layer-1 tree of T. Let u_1 be the highest node in this binary tree such that r and s are contained in different subtrees of u_1. Let $1 < i \leq d$ and assume that $u_1, u_2, \ldots, u_{i-1}$ have been defined already, and that u_{i-1} is a node of a layer-$(i-1)$ tree. Then, let u_i be the highest node in the layer-i tree that corresponds to u_{i-1} such that r and s are contained in different subtrees of u_i. In this way, we get a sequence of nodes $u_1, u_2, \ldots, u_d$ such that

- u_1 is a node of the layer-1 tree of T,
- u_i is a node of the layer-i tree that corresponds to u_{i-1}, $1 < i \leq d$,
- r and s are contained in different subtrees of u_i, $1 \leq i \leq d$.

We claim that $\eta_{d+1}(u_d) = \delta_C(r,s)$, which will complete the proof.

Let $u = u_d$ and consider the point x_u as defined in the description of T. (The nodes $u_d, u_{d-1}, \ldots, u_1$ defined in the description of T are exactly the nodes that we just defined. The difference is that they are defined in the reversed order.) Since $\eta < \infty$, (4) implies that $\delta_C(r,s) < \infty$. Hence $s \in C_r$. This shows that $s'_i \geq r'_i$ for $1 \leq i \leq d$. Since r and s are in different subtrees of u_i, we know that x'_{ui} separates the coordinates r'_i and s'_i. Therefore, we must have $r'_i \leq x'_{ui} \leq s'_i$ for $1 \leq i \leq d$. Since $r \in RT_{source}$ and $s \in RT_{sink}$, it follows that r and s are contained in the lists $L^-_{u,d+1}$ and $L^+_{u,d+1}$, respectively. But then, since $\delta_C(r,s)$ is minimal, we must have $\eta_{d+1}(u) = \delta_C(r,s)$. ∎

We now prove that algorithms *gap_greedy*$'$ and *gap_greedy*$''$ compute the same graph (S,E). Assume for the sake of analysis, that we run both algorithms in parallel. Consider a cone C. After the initialization of the iteration for C, we have

$$\begin{aligned}\{dist(r,s) : r \in S, s \in S, r \neq s, dist(r,s) < \infty\} = \\ \{\delta_C(r,s) : r \in RT_{source}, s \in RT_{sink}, r \neq s, \delta_C(r,s) < \infty\}.\end{aligned} \tag{5}$$

Consider one iteration of the while-loop of both algorithms and assume that (5) holds at the beginning of these iterations. Algorithm *gap_greedy*$'$ takes a pair (r',s') for which $dist(r',s')$ is a minimal element in the set on the left-hand side. By Lemma 8, algorithm *gap_greedy*$''$ takes a pair (r'',s'') for which $\delta_C(r'',s'')$ is a minimal element in the set on the right-hand side. Hence we have $dist(r',s') = \delta_C(r'',s'')$. Note that the sets in (5) may have several minimal elements. In that case, we force algorithm *gap_greedy*$'$ to choose the same pair as *gap_greedy*$''$. We denote the chosen pair by (r,s). Both algorithms add the edge (r,s) to their edge sets E_C. Then *gap_greedy*$'$ updates certain *dist*-values and *gap_greedy*$''$ updates the structures RT_{source}, RT_{sink} and T. By comparing the algorithms, it follows immediately that (5) still holds after the iteration.

This proves that algorithms *gap_greedy*$'$ and *gap_greedy*$''$ compute the same edge set E. We proved in Lemmas 6 and 7 that *gap_greedy*$'$ always produces a t-spanner of bounded degree and, if $w > 0$, its weight is at most $O(\log n)$ times the weight of a minimum spanning tree for S. Hence, the same is true for algorithm *gap_greedy*$''$.

We analyze the complexity of our algorithm. Consider one cone C. The $(d+1)$-layered structure T has size $O(n \log^d n)$ and can be built in time $O(n \log^d n)$. The structures RT_{source} and RT_{sink} have size $O(n \log^{d-1} n)$ and can be built in time $O(n \log^{d-1} n)$. By applying dynamic fractional cascading ([10]) and observing that we only delete points, their amortized deletion time is bounded by $O(\log^{d-1} n)$, and their query time is bounded by $O(\log^{d-1} n)$ plus the number of reported points. Since each point of S is reported in at most one query for each RT-structure, the total query time is bounded by $O(n \log^{d-1} n)$.

Consider one point p of S. It is deleted at most once from RT_{source}, taking $O(\log^{d-1} n)$ amortized time. If it is deleted from RT_{source}, then we delete p from all lists $L^-_{u,d+1}$ and update T and η. We saw already that this takes $O(\log^d n)$ time.

Hence for each point p of S, we spend $O(\log^d n)$ time for updating RT_{source} and T. The same bound holds for updating RT_{sink} and T. It follows that the entire algorithm has running time $O(n \log^d n)$. This proves:

Theorem 1 *Let t, θ and w be real numbers such that $0 < \theta < \pi/4$, $0 \leq w < (\cos\theta - \sin\theta)/2$ and $t \geq 1/(\cos\theta - \sin\theta - 2w)$. Let S be a set of n points in $\mathbb{R}^d$. In $O((c/\theta)^{d-1} n \log^d n)$ time and using $O((c/\theta)^{d-1} n + n \log^d n)$ space, algorithm gap_greedy$''(S, \theta, w)$ computes a t-spanner for S such that each point of S has degree at most $O((c/\theta)^{d-1})$, for some suitable constant c. If $w > 0$, then the weight of this t-spanner is at most $O((c/\theta)^{d-1}(1/w) \log n)$ times the weight of a minimum spanning tree for S.*

Corollary 1 *Let t and θ be real numbers such that $0 < \theta < \pi/4$ and $t \geq 1/(\cos\theta - \sin\theta)$. Let S be a set of n points in $\mathbb{R}^d$. In $O((c/\theta)^{d-1} n \log^d n)$ time and using $O((c/\theta)^{d-1} n + n \log^d n)$ space, we can compute a t-spanner for S such that each point of S has degree at most $O((c/\theta)^{d-1})$ and the weight of this t-spanner is at most a constant times the weight of a minimum spanning tree for S.*

Proof: Let θ' be such that $0 < \theta' < \pi/4$ and $\sqrt{t} \geq 1/(\cos\theta' - \sin\theta')$. Let G be the $\sqrt{t}$-spanner that is constructed by algorithm *gap_greedy*$''(S, \theta', 0)$. Das and Narasimhan [5] show how to compute in $O(n \log^2 n)$ time a $\sqrt{t}$-spanner G' of G. Clearly, G' is a t-spanner for S. Also, since G' is a subgraph of G, it has bounded degree. Das and Narasimhan partition the edges of G' into two sets E_0 and E_1. The total weight of the edges in E_0 is bounded by the weight of a minimum spanning tree for S. The edges in E_1 satisfy the so-called leap-frog property. Recent results of [4, 6] show that the leap-frog property implies that the total weight of the edges in E_1 is proportional to the weight of a minimum spanning tree for S. ■

6 Application to distance enumeration

Salowe ([12, 15]) has suggested the use of Dijkstra's algorithm with bounded degree spanners for *interdistance enumeration*. Let S be a set of n points in $\mathbb{R}^d$ and let k be an integer between 1 and $\binom{n}{2}$. Then we want to enumerate the k smallest distances, sorted in non-decreasing order. The value of k may or may not be known in advance.

In Section 6.1, we show that we can use *any* bounded degree spanner to enumerate the k smallest interpoint distances *approximately* in $O(n + k \log k)$ time, not including the time to construct the spanner. In Section 6.2, we show that we can also do exact enumerations using any bounded degree spanner in $O((n + k) \log n)$ time. Finally, in Section 6.3, we show how to improve the time bound for exact enumeration to $O(n + k \log k)$ by exploiting special properties of the bounded degree spanner constructed in this paper.

6.1 Approximate interdistance enumeration

Let $G = (S, E)$ be any t-spanner for S having bounded degree. Although we describe our algorithm for an undirected spanner, the enumeration technique can also be used on a directed spanner of bounded out-degree. Let p and q be two points of S. The *weight* of this pair is defined as the Euclidean distance between p and q, and its *pseudo-weight* is defined as the Euclidean length of a shortest path in G between p and q.

The algorithm for approximate distance enumeration is similar to that of Dickerson et al.[8]. We initialize a priority queue with all pairs of points corresponding to the edges of G, with priority given by the pseudo-weight of the pair. In each iteration, we extract the pair p, q with smallest priority and report it together with its weight. For each edge (q, r) of G, we compute the priority of the pair p, r as the sum of the priority of the pair p, q and the weight of the edge (q, r). We insert the pair p, r into the priority queue if it has not already been reported and if it is not already in the queue with a smaller priority. We do the symmetrical thing with all edges (p, s) of G.

It is easy to see that this algorithm is running Dijkstra's shortest path algorithm simultaneously from all the points of S and that the pairs are reported in order of non-decreasing pseudo-weight. Our claim is that this implies that the pairs are reported approximately in order of non-decreasing weight. We make this precise in the following lemma.

Lemma 9 *Consider the t-spanner $G = (S, E)$. Arrange all pairs of points in order of non-decreasing weight and assign an index to each pair based on its rank in this sequence. Let w_i and w'_i denote the weight and pseudo-weight of the pair with index i, respectively. Let π be a permutation of the pairs that orders them on the basis of non-decreasing pseudo-weight, i.e., $w'_{\pi(1)} \leq w'_{\pi(2)} \leq w'_{\pi(3)} \leq \ldots$ Then for any i, $1 \leq i \leq \binom{n}{2}$,*

$$\frac{w_i}{t} \leq w_{\pi(i)} \leq t w_i \tag{6}$$

and

$$w_i \leq w'_{\pi(i)} \leq t w_i. \tag{7}$$

Proof: It follows from the definition of a t-spanner that for any i

$$w_i \leq w'_i \leq t w_i. \tag{8}$$

First we show that (7) and (8) together imply (6). Applying (8) with $\pi(i)$, we see that $w_{\pi(i)} \leq w'_{\pi(i)}$. By (7), $w'_{\pi(i)} \leq t w_i$. Hence, $w_{\pi(i)} \leq t w_i$, which proves the right inequality of (6). Again applying (8) with $\pi(i)$, we get $w_{\pi(i)} \geq w'_{\pi(i)}/t$, which by (7) is at least equal to w_i/t. This proves the left inequality of (6).

Thus it remains to prove (7). We first show that $w_i \leq w'_{\pi(i)}$. There are two cases to consider. First assume that $\pi(i) \geq i$. Then $w_{\pi(i)} \geq w_i$. Using (8) with $\pi(i)$, this implies the desired result. Next assume that $\pi(i) < i$. Since π is a one-to-one function, there is a j, $1 \leq j < i$, such that $\pi(j) \geq i$. (Otherwise, all values $\pi(1), \pi(2), \ldots, \pi(i)$ would belong to the set $\{1, 2, \ldots, i-1\}$.) Since $j < i$, we have $w'_{\pi(j)} \leq w'_{\pi(i)}$. Also, since $\pi(j) \geq i$, we have $w_{\pi(j)} \geq w_i$. Applying (8) with $\pi(j)$, we see that $w'_{\pi(j)} \geq w_{\pi(j)}$. Combining these inequalities, we get $w_i \leq w_{\pi(j)} \leq w'_{\pi(j)} \leq w'_{\pi(i)}$, which is the desired result.

To show that $w'_{\pi(i)} \leq t w_i$, we again consider two cases. First assume that $\pi(i) \leq i$. Then $w_{\pi(i)} \leq w_i$. Applying (8) with $\pi(i)$ gives $w'_{\pi(i)} \leq t w_{\pi(i)}$. Hence, $w'_{\pi(i)} \leq t w_i$. Next assume that $\pi(i) > i$. Since π is a one-to-one function, there is a j, $j > i$, such that $\pi(j) \leq i$. (Otherwise, all values $\pi(l)$, $i \leq l \leq \binom{n}{2}$, would belong to the set $\{i+1, i+2, \ldots, \binom{n}{2}\}$.) Since $j > i$, we have $w'_{\pi(j)} \geq w'_{\pi(i)}$. Also, since $\pi(j) \leq i$, we have $w_{\pi(j)} \leq w_i$. Applying (8) with $\pi(j)$, we see that $w'_{\pi(j)} \leq t w_{\pi(j)}$. It follows that $w'_{\pi(i)} \leq w'_{\pi(j)} \leq t w_{\pi(j)} \leq t w_i$. This completes the proof. ∎

The algorithm described above reports the sequence

$$w_{\pi(1)}, w_{\pi(2)}, \ldots, w_{\pi(k)}.$$

The right inequality in (6) implies that this sequence approximates the true k smallest distances.

We estimate the running time of the algorithm. Assume that k is known in advance. To improve the efficiency of the priority queue, we maintain only k pairs in it. The time to initialize the priority queue is $O(n)$. Since the spanner G has bounded degree, the queue is updated $O(k)$ times. Each operation on the priority queue takes $O(\log k)$ time. Therefore, the total running time is bounded by $O(n + k \log k)$.

If k is not known, then we proceed as follows: First, we initialize the priority queue with the $O(n)$ pairs that correspond to the edges of G. Then we take an initial constant value k_0 and run the above algorithm. If we have reported k_0 pairs, then we undo all operations we performed so far, i.e., until we have our initial priority queue again, and repeat the same procedure with value $2k_0$. We keep on doing this until we have reported k pairs. The running time of this algorithm is bounded by $O(n + \sum_{i \geq 0} k/2^i \log k) = O(n + k \log k)$.

6.2 Exact interdistance enumeration

Consider again an arbitrary undirected t-spanner $G = (S, E)$ of bounded degree. (Again, the enumeration technique can also be used on a directed spanner of bounded out-degree.) We can enumerate the k exact smallest distances, using basically the same algorithm as in Section 6.1. There are two differences. First, the priority queue is maintained at full size, i.e., we do not prune it to keep only k pairs. Second, we do not immediately report the pairs as they are extracted from the queue; instead we keep track of the k closest pairs seen so far. We continue to run the algorithm until the pseudo-weight of the pair extracted from the queue is larger than t times the weight of the k-th closest pair seen so far. At termination the k closest pairs seen by the algorithm are reported.

We prove the correctness of this algorithm. Let x be the weight of the k-th closest pair reported by the algorithm. We claim that any pair not seen by the algorithm has weight at least equal to x. This will prove that the algorithm correctly reports the k closest pairs of S.

Since pairs are enumerated in order of non-decreasing pseudo-weight, any pair not seen by the algorithm must have pseudo-weight at least equal to tx. Using the notation of Lemma 9, let i be the index of such a pair. Then $w'_i \geq tx$. Then (8) implies that $w_i \geq w'_i/t \geq x$, which establishes the correctness of the algorithm.

Before we analyze the running time of the algorithm, we prove the following claim: The algorithm terminates as soon as it extracts a pair from the queue with index i such that $w_i > tw_k$. (Note that during its execution, the algorithm does not know w_k.)

To prove this, consider such a pair with index i. Note that $w'_i > w_i$, which implies that $w'_i \geq tw_k$. Since the algorithm extracts pairs in order of non-decreasing pseudo-weight, it must already have extracted all pairs with pseudo-weight at most equal to tw_k. It follows from (8) that if a pair has weight at most w_k, then it has pseudo-weight at most tw_k. Thus, all pairs with weight at most w_k have been extracted already. Therefore, at the moment when the pair with index i is extracted, w_k is the weight of the k-th closest pair seen so far. Hence, the algorithm terminates at this moment, proving the claim.

Now we estimate the running time. The number of pairs extracted from the queue is at most equal to the number of pairs having weight at most tw_k. In [9, 13], it is shown that the latter is bounded by $O(n+k)$. Hence, after initializing the queue, which takes $O(n)$ time, the algorithm performs $O(n+k)$ queue operations. (This follows from the fact that the spanner G has bounded degree.) Since each queue operation takes $O(\log n)$ time, the entire running time is bounded by $O((n + k)\log n)$.

6.3 Improved solution for exact interdistance enumeration

We can improve the time bound of Section 6.2 by using the bounded degree spanner that is constructed by algorithm *gap_greedy*$''(S, \theta, w)$ for $0 < \theta < \pi/4$ and $w = 0$. To enumerate the k exact closest pairs, we run the same algorithm as in Section 6.1, with one change: The priority of a pair of points is given by its weight.

The running time of this algorithm is clearly the same as that of Section 6.1: it is bounded by $O(n + k \log k)$. We give an inductive proof that the algorithm outputs the k closest pairs in order of non-decreasing weight.

Consider the closest pair p, q in S. Since p and q are connected by an edge in the spanner, this pair is put into the priority queue in the initialization step. Hence, it is the first pair to be reported.

Let $1 < m \leq k$, and assume that the $m - 1$ closest pairs have been reported by the algorithm. Let p, q be the m-th closest pair in S. We show that this pair is the next one to be reported. If p and q are connected by an edge in the spanner, then we are done, because then this pair was put into the queue in the initialization step. Hence, now this pair has smallest priority in the queue, and it will be reported.

Assume that p and q are not connected by an edge. Then it follows from the proof of Lemma 2 that (i) there is a point $s \in S$ such that (p, s) is an edge and $|sq| < |pq|$, or (ii) there is a point $r \in S$ such that (q, r) is an edge and $|pr| < |pq|$. Assume first that (i) holds. Then s, q must be one of the $m - 1$ closest pairs. At the moment when this pair was reported, the algorithm inserted the pair p, q into the queue. Hence, after $m - 1$ pairs have been reported, the pair p, q has minimal priority in the queue. Hence, it is the next pair to be reported. Case (ii) can be treated similarly.

7 Concluding remarks

We have given an $O(n \log^d n)$ time algorithm that constructs a t-spanner of bounded degree having a weight that is proportional to the weight of a minimum spanning tree for the n points.

After the first version of this paper was written, the authors, together with Das, Mount, and Salowe, gave an $O(n \log n)$ time algorithm—that is based on completely different techniques—to construct a bounded degree spanner having weight proportional to the weight of a minimum spanning tree. See [1].

Acknowledgements

The authors thank Andrea Eßer for producing the figures.

References

[1] S. Arya, G. Das, D.M. Mount, J.S. Salowe, M. Smid. *Euclidean spanners: short, thin, and lanky.* Proc. 27th Annu. ACM Sympos. on the Theory of Computing, 1995, to appear.

[2] B. Chandra, G. Das, G. Narasimhan and J. Soares. *New sparseness results on graph spanners.* Proc. 8th Annu. ACM Sympos. Comput. Geom., 1992, pp. 192–201.

[3] G. Das and P.J. Heffernan. *Constructing degree-3 spanners with other sparseness properties.* Proc. 4th Annual Intern. Symp. on Algorithms, Lecture Notes in Computer Science, Vol. 762, Springer-Verlag, Berlin, 1993, pp. 11–20.

[4] G. Das, P. Heffernan and G. Narasimhan. *Optimally sparse spanners in 3-dimensional Euclidean space.* Proc. 9th Annu. ACM Sympos. Comput. Geom., 1993, 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. Salowe. *A new way to weigh malnourished Euclidean graphs.* Proc. 6th Annu. ACM-SIAM Sympos. on Discrete Algorithms, 1995, pp. 215–222.

[7] A. Datta, H.P. Lenhof, C. Schwarz and M. Smid. *Static and dynamic algorithms for k-point clustering problems.* Proc. 3rd WADS, Lecture Notes in Computer Science, Vol. 709, Springer-Verlag, Berlin, 1993, pp. 265-276.

[8] M.T. Dickerson, R.L. Drysdale and J.R. Sack. *Simple algorithms for enumerating interpoint distances and finding k nearest neighbors.* Internat. J. Comput. Geom. Appl. **2** (1992), pp. 221–239.

[9] H.-P. Lenhof and M. Smid. *Enumerating the k closest pairs optimally.* Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci., 1992, pp. 380–386.

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

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

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

[13] J.S. Salowe. *Enumerating interdistances in space.* Internat. J. Comput. Geom. Appl. **2** (1992), pp. 49–59.

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

[15] J.S. Salowe. Personal communication, 1994.

[16] M. Smid. *Maintaining the minimal distance of a point set in polylogarithmic time.* Discrete Comput. Geom. **7** (1992), pp. 415-431.

[17] 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.

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