Fast pruning of geometric spanners *

Joachim Gudmundsson[1], Giri Narasimhan[2], and Michiel Smid[3]

[1] Department of Mathematics and Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands h.j.gudmundsson@tue.nl
[2] School of Computer Science, Florida International University, Miami, FL 33199, USA. giri@cs.fiu.edu
[3] School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. michiel@scs.carleton.ca

Abstract. Let S be a set of points in $\mathbb{R}^d$. Given a geometric spanner graph, $G = (S, E)$, with constant dilation t, and a positive constant ε, we show how to construct a $(1 + \varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges in time $\mathcal{O}(|E| + |S| \log |S|)$. Using the results in [11], this implies an $\mathcal{O}(|E| + |S| \log |S|)$-time algorithm that constructs a $(1 + \varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges and total weight $\mathcal{O}(wt(MST(S)))$.

1 Introduction

Complete graphs represent ideal communication networks, but they are expensive to build; sparse spanners represent low cost alternatives. The number of edges of the spanner network is a measure of its sparseness; other sparseness measures include the weight, the maximum degree and the number of Steiner points. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and many other areas, and have been the subject of considerable research [1, 2, 6, 8, 16].

Consider a set S of n points in $\mathbb{R}^d$. Throughout this paper, we will assume that d is constant. A network on S can be modelled as an undirected graph G with vertex set S and with edges $e = (u, v)$ of weight $wt(e)$. In this paper we consider geometric networks, where the weight of the edge $e = (u, v)$ is equal to the Euclidean distance $|uv|$ between its two endpoints u and v. Let $\delta_G(p, q)$ denote the length of a shortest path in G between p and q. Hence, G is a *t-spanner* for S, if $\delta_G(p, q) \leqslant t \cdot |pq|$ for any two points p and q of S. The minimum value t such that G is a t-spanner for S is called the *dilation* of G. A subgraph G' of G is a t'-spanner of G, if $\delta_{G'}(p, q) \leqslant t' \cdot \delta_G(p, q)$ for any two points p and q of S.

Many algorithms are known that compute t-spanners with useful properties such as linear size ($\mathcal{O}(n)$ edges), bounded degree, small spanner diameter (i.e., any two points are connected by a t-spanner path consisting of only a small number of edges), low weight (i.e., the total length of all edges is proportional to the weight of a minimum spanning tree of S), and fault-tolerance; for example, see [1–4, 6–9, 11, 15, 16, 18, 20], and the surveys [10, 19]. All these algorithms compute t-spanners for any given constant $t > 1$. However, all these algorithms either start with a point set, or start with a spanner that has a linear number of edges.

In this paper we consider the problem of efficiently *pruning* a given t-spanner, even if it has a superlinear number of edges. That is, given a geometric graph $G = (S, E)$ in $\mathbb{R}^d$ with n points and constant dilation t, and a positive constant ε, we consider the problem of constructing a $(1 + \varepsilon)$-spanner of G with $\mathcal{O}(n)$ edges. Thus the resulting subgraph of G is guaranteed to be a $(t(1 + \varepsilon))$-spanner of S.

* J.G. was supported by the Netherlands Organisation for Scientific Research (NWO) and M.S. was supported by NSERC.

The greedy algorithm of [8, 11] can be used to compute a $(1+\varepsilon)$-spanner G' of G. However, the greedy algorithm starts by sorting the edges of G and, thus, has running time $\Omega(|E| \log n)$. In [13], an algorithm was presented with running time $\mathcal{O}(|E| \log n)$, that produces a $(1+\varepsilon)$-spanner G' of G with $\mathcal{O}(n)$ edges.

In this paper, we show how the running time can be improved to $\mathcal{O}(|E| + n \log n)$ time. Furthermore, using the results in [11], we show that with the same time complexity, we can compute a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(n)$ edges and with total weight $\mathcal{O}(wt(MST(S)))$.

In a series of papers by Gudmundsson et al. [12–14], it was shown that approximate shortest path queries can be answered in constant time using $\mathcal{O}(|E| \log n)$ preprocessing, provided that the given graph is a t-spanner. The time complexity of the preprocessing depends on the time to prune the graph, which was shown to be $\mathcal{O}(|E| \log n)$. Using the pruning algorithm presented here, we improve the preprocessing time of the data structure in [12–14] to $\mathcal{O}(|E| + n \log n)$. We also improve the time complexity in [17] for computing a $(1+\varepsilon)$-approximation to the dilation of a geometric graph to $\mathcal{O}(|E| + n \log n)$, provided we know in advance that the dilation is bounded from above by a constant. In Section 6 we consider several other applications for which our pruning tool improves the running time of existing algorithms.

Our model of computation is the traditional algebraic computation tree model with the added power of indirect addressing.

2 Preliminaries

In the next sections, we will show how to prune a graph. Our construction uses the well-separated pair decomposition of Callahan and Kosaraju [5]. We briefly review this decomposition below.

If X is a bounded subset of $\mathbb{R}^d$, then we denote by $R(X)$ the smallest axes-parallel d-dimensional rectangle that contains X. We call $R(X)$ the *bounding box* of X. Let $l(R(X))$, or $l(X)$, be the length of the longest side of $R(X)$.

Definition 1. *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well-separated *with respect to s, if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that (i) C_A contains the bounding box $R(A)$ of A, (ii) C_B contains the bounding box $R(B)$ of B, and (iii) the minimum distance between C_A and C_B is at least s times the radius of C_A.*

The parameter s will be referred to as the *separation constant*. The next lemma follows easily from Definition 1.

Lemma 1. *Let A and B be two finite sets of points that are well-separated w.r.t. s, let x and p be points of A, and let y and q be points of B. Then (i) $|xy| \leqslant (1 + 4/s) \cdot |pq|$, and (ii) $|px| \leqslant (2/s) \cdot |pq|$.*

Definition 2 ([5]). *Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A* well-separated pair decomposition *(WSPD) for S with respect to s is a sequence of pairs of non-empty subsets of S, $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$, such that*

1. *$A_i \cap B_i = \emptyset$, for all $i = 1, \ldots, m$,*
2. *for any two distinct points p and q of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $q \in A_i$ and $p \in B_i$,*

3. A_i and B_i are well-separated w.r.t. s, for all $i = 1, \ldots, m$.

The integer m is called the size *of the WSPD.*

Callahan and Kosaraju showed that a WSPD of size $m = \mathcal{O}(n)$ can be computed in $\mathcal{O}(n \log n)$ time. Their algorithm uses a binary tree T, called the *split tree*. We briefly describe the main ideas behind their work because they are useful when we describe our results. They start by computing the bounding box of S, which is successively split by $(d-1)$-dimensional hyperplanes, each of which is orthogonal to one of the axes. If a box is split, then each of the two resulting boxes contains at least one point of S. If a box contains exactly one point, the box is not split any further. The split tree T stores the points of S at its leaves; one leaf per point. Each node stores the bounding box of all points in its subtree, and is associated with a subset of S, denoted by S_u.

Callahan and Kosaraju showed that the split tree T can be computed in $\mathcal{O}(n \log n)$ time, and that, given T, a WSPD of size $m = \mathcal{O}(n)$ can be computed in $\mathcal{O}(n)$ time. Each pair $\{A_i, B_i\}$ in this WSPD is represented by two nodes u_i and v_i of T, i.e., we have $A_i = S_{u_i}$ and $B_i = S_{v_i}$. We end this section with three lemmas that will be used later on.

Lemma 2. *Let u and u' be two nodes in the split tree T such that u' is in the subtree of u and the path between them contains at least d edges. Then the length of a longest side of the bounding box of u' is at most $1/2$ times the length of a longest side of the bounding box of u.*

Proof. Follows from the construction of the split tree; see [5] for details. □

Lemma 3. *Let A and B be two sets of points in $\mathbb{R}^d$ that are well-separated with respect to s, and let p and q be points in A and B, respectively. The length of each side of the bounding boxes of A and B is less than or equal to $(2/s)|pq|$.*

Proof. Let C_A and C_B be two balls of equal radius ρ such that $R(A) \subseteq C_A$, $R(B) \subseteq C_B$, and the distance between C_A and C_B is greater than or equal to $s\rho$. Moreover, let L be the length of a longest side of the bounding boxes $R(A)$ and $R(B)$. We have to show that $L \leqslant (2/s)|pq|$.

We may assume without loss of generality that L is the length of a longest side of $R(A)$. Since $R(A)$ is contained in a ball of radius ρ, we have $L \leqslant 2\rho$. Since the distance between C_A and C_B is at least $s\rho$, and since $p \in C_A$ and $q \in C_B$, we have $|pq| \geqslant s\rho$. Combining these two inequalities implies that $L \leqslant (2/s)|pq|$. □

Lemma 4. *Let A and B be two sets of points in $\mathbb{R}^d$, and let $p \in A$ and $q \in B$. Assume that the length of each side of the bounding boxes of A and B is less than or equal to $\alpha|pq|$, where $\alpha = 2/((s+4)\sqrt{d})$. Then the sets A and B are well-separated with respect to s.*

Proof. Let L be the length of a longest side of the bounding boxes $R(A)$ and $R(B)$. Then there are two balls C_A and C_B of radius $\rho = \frac{1}{2}\sqrt{d}L$ such that $R(A) \subseteq C_A$ and $R(B) \subseteq C_B$. By the assumption in the lemma, we have $\rho \leqslant \frac{1}{2}\sqrt{d}\alpha|pq| = |pq|/(s+4)$.

Let a and b be the points on the boundaries of C_A and C_B, respectively, for which $|ab|$ is minimum. Then,

$$|ab| \geq |pq| - |pa| - |bq| \geq (s+4)\rho - 2\rho - 2\rho = s\rho.$$

Hence the result. □

3 A general pruning approach

Recall that we are given a set S of n points in $\mathbb{R}^d$, a t-spanner $G = (S, E)$ for some real constants $t > 1$ and $\varepsilon > 0$. Our goal is to compute a sparse $(1+\varepsilon)$-spanner G' of G.

Now suppose that there exists a set of m pairs, $P = \{\{a_1, b_1\}, \ldots, \{a_m, b_m\}\}$, with the property that for each edge (p, q) in E, there is an index i such that for some s,

1. $|pa_i| \leqslant (2/s)|a_i b_i|$ and $|qb_i| \leqslant (2/s)|a_i b_i|$, or
2. $|pb_i| \leqslant (2/s)|a_i b_i|$ and $|qa_i| \leqslant (2/s)|a_i b_i|$.

In other words, for each edge (p, q), the set P contains a "close approximation". Then, we show below that, if $s = \frac{1}{\varepsilon}((1+\varepsilon)(8t+4)+4)$, then there exists a $(1+\varepsilon)$-spanner of G with at most m edges. As the keen reader may have guessed, we will show later that the set P can be easily constructed from a WSPD of S.

To prove the existence of the subgraph G' as a $(1+\varepsilon)$-spanner of G, we prune G *with respect to* a set of pairs P as follows. Let C_i, $1 \leqslant i \leqslant m$, be m lists that are initially empty. For each edge (p, q) in E, pick any index i for which condition (i) or (ii) from above is satisfied, and add the edge (p, q) to the list C_i. We define G' to be the graph (S, E'), where the edge set E' contains exactly one edge from each non-empty list C_i, $1 \leqslant i \leqslant m$.

Lemma 5. *The graph $G' = (S, E')$ is a $(1+\varepsilon)$-spanner of G.*

Proof. It suffices to prove the following claim: For each edge (p, q) of E, there is a path between p and q in G' having length at most $(1+\varepsilon)|pq|$. We will prove this claim by induction on the length of the edge (p, q).

To start the induction, let (p, q) be a shortest edge in E. Let i be the index such that $(p, q) \in C_i$. Then (i) $|pa_i| \leqslant (2/s)|a_i b_i|$ and $|qb_i| \leqslant (2/s)|a_i b_i|$, or (ii) $|pb_i| \leqslant (2/s)|a_i b_i|$ and $|qa_i| \leqslant (2/s)|a_i b_i|$. We may assume without loss of generality that (i) holds, as illustrated in Fig. 1a. Let (x, y) be the edge of C_i that is contained in E'. Observe that (i) $|xa_i| \leqslant (2/s)|a_i b_i|$ and $|yb_i| \leqslant (2/s)|a_i b_i|$, or (ii) $|xb_i| \leqslant (2/s)|a_i b_i|$ and $|ya_i| \leqslant (2/s)|a_i b_i|$. Again, we may assume without loss of generality that (i) holds. If not, we switch the roles of x and y. Since G is a t-spanner for S, we have

$$\delta_G(p, x) \leqslant t|px| \leqslant t\left(|pa_i| + |a_i x|\right) \leqslant (4t/s)|a_i b_i|.$$

Moreover, we have

$$|a_i b_i| \leqslant |a_i p| + |pq| + |qb_i| \leqslant (4/s)|a_i b_i| + |pq|.$$

Upon rearranging, this inequality can be rewritten as

$$|a_i b_i| \leqslant \frac{s}{s-4}|pq|.$$

Since $s > 4(t+1)$, we have

$$\delta_G(p, x) \leqslant \left(\frac{4t}{s}\right)\frac{s}{s-4}|pq| = \frac{4t}{s-4}|pq| < |pq|,$$

Hence, if $p \neq x$, then each edge on the shortest path in G between p and x has length less than $|pq|$. Therefore, since (p, q) is a shortest edge in E, it follows that $p = x$. Similarly, we have $q = y$. Hence, (p, q) is an edge of E', which implies that $\delta_{G'}(p, q) = |pq| \leqslant (1+\varepsilon)|pq|$.

Now assume that (p, q) is not a shortest edge in E. Furthermore, assume that $\delta_{G'}(u, v) \leqslant (1 + \varepsilon)|uv|$ for all edges (u, v) in E with $|uv| < |pq|$. As before, let i be the index such that $(p, q) \in C_i$, and let (x, y) be the edge of C_i that is contained in E'. Hence, we have $|pa_i| \leqslant (2/s)|a_ib_i|$, $|qb_i| \leqslant (2/s)|a_ib_i|$, $|xa_i| \leqslant (2/s)|a_ib_i|$, and $|yb_i| \leqslant (2/s)|a_ib_i|$.

Exactly as before, we have

$$\delta_G(p, x) \leqslant \frac{4t}{s-4}|pq| < |pq|;$$

therefore, each edge on the shortest path in G between p and x has length less than $|pq|$. By induction, it follows that

$$\delta_{G'}(p, x) \leqslant (1 + \varepsilon) \cdot \delta_G(p, x) \leqslant (1 + \varepsilon)\frac{4t}{s-4}|pq|.$$

In a completely symmetric way, we get

$$\delta_{G'}(y, q) \leqslant (1 + \varepsilon)\frac{4t}{s-4}|pq|.$$

Hence,

$$\delta_{G'}(p, q) \leqslant \delta_{G'}(p, x) + |xy| + \delta_{G'}(y, q) \leqslant (1 + \varepsilon)\frac{8t}{s-4}|pq| + |xy|.$$

Since

$$|xy| \leqslant |xa_i| + |a_ib_i| + |b_iy| \leqslant (1 + 4/s)\,|a_ib_i| \leqslant (1 + 4/s)\,\frac{s}{s-4}|pq| = \frac{s+4}{s-4}|pq|,$$

and using our choice of s, it follows that

$$\delta_{G'}(p, q) \leqslant (1 + \varepsilon)\frac{8t}{s-4}|pq| + \frac{s+4}{s-4}|pq| = (1 + \varepsilon)|pq|.$$

□

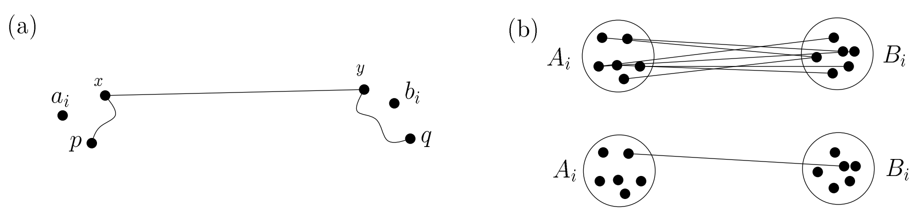

Fig. 1. (a) A $(1 + \varepsilon)$-spanner path between p and q. (b) Pruning the spanner graph using the WSPD.

The above process essentially prunes G using set P as a "guide". Each edge of G is "mapped" to a pair in P, and in the pruned subgraph, for each pair in P, we retain one edge that is mapped to it (if any). In order to apply the above general result, we need algorithms that do the following:

1. Compute $P = \{a_i, b_i\}_{1 \leqslant i \leqslant m}$, with $m = \mathcal{O}(n)$.
2. For each edge (p, q) in E, compute the index i such that the condition for the set P holds.

A straight-forward approach for step 1, which appears already in [13], is as follows. Compute the WSPD with separation constant $s = ((1+\varepsilon)(8t+4)+4)/\varepsilon$, using the algorithm of Callahan and Kosaraju, see Section 2. Given this WSPD, define the set P as follows. For each well-separated pair $\{A_i, B_i\}$, choose a pair consisting of an arbitrary point in A_i and an arbitrary point in B_i; see Fig 1b. Using Lemma 1, the properties that are needed for P are satisfied, thus we can apply Lemma 5.

As for step 2, Arya *et al.* [3] showed that, after an $\mathcal{O}(n \log n)$-time preprocessing of the split tree, this index can be computed in $\mathcal{O}(\log n)$ time. Hence, the entire graph G' can be computed in $\mathcal{O}((n + |E|) \log n) = \mathcal{O}(|E| \log n)$ time. We have proved the following result.

Theorem 1. *Given a geometric graph $G = (S, E)$ with n vertices, which is a t-spanner for S, for some real constant $t > 1$, we can compute, in $\mathcal{O}(|E| \log n)$ time, a $(1+\varepsilon)$-spanner of G having $\mathcal{O}(n)$ edges, for any given real constant $\varepsilon > 0$.*

4 An improved algorithm

Above we showed that the time-complexity of the algorithm can be written as $\mathcal{O}(|E| + n \log n + |E| \cdot \tau(n))$, where $\tau(n)$ is the time needed to find the pair $\{a_i, b_i\}$ in P, given a query (x, y) such that the condition mentioned at the beginning of Section 3 holds. Below, we show a stepwise refinement of the basic scheme presented in Section 3.

4.1 Improvements for a restricted case – bounded aspect ratio

Let T be the split tree for S, and let $\{A_i, B_i\}$, $1 \leqslant i \leqslant m$, be the well-separated pair decomposition of S obtained from T, with separation constant $s > 0$. Let $L > 0$ be a real number, let $c \geqslant 1$ be an integer constant, and let F be a set of k pairs of points in S such that $L/n^c \leqslant |xy| \leqslant L$ holds for each pair $\{x, y\} \in F$. We say that F has *polynomially bounded aspect ratio.*

In this section, we show how to compute, for every $\{x, y\} \in F$, the corresponding well-separated pair, i.e., the index i for which $x \in A_i$ and $y \in B_i$ or $x \in B_i$ and $y \in A_i$. Recall that every node of T stores the bounding box of the set of all points stored in its subtree.

Let

$$\alpha = \frac{2}{\sqrt{d}(s+4)}.$$

For each point $x \in S$, we define the following nodes in T:

u_x: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $(2/s)L$.

u'_x: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $\alpha L/n^c$.

Moreover, for each pair $e = \{x, y\} \in F$, we define the following nodes in T.

u_{ex}: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $(2/s)|xy|$.

u'_{ex}: the highest node on the path from the leaf storing x to the root, such that its bounding box has sides of length at most $\alpha|xy|$.

Observation 6 *By traversing T, all nodes u_x and u'_x, $x \in S$, can be computed in $\mathcal{O}(n)$ time.*

Because of the polynomially bounded aspect ratio assumption, the path from u'_x to u_x contains all nodes whose subsets contain x and are involved in well-separated pairs corresponding to pairs in F. In particular, the path from u'_{ex} to u_{ex} contains the node whose subset is A_i. These observations are formalized in the lemma below.

Lemma 7. *Let $e = \{x, y\}$ be a pair in F, and let i be the index such that $x \in A_i$ and $y \in B_i$. Let v_i and w_i be the nodes of T that represent A_i and B_i, respectively. Then,*

1. *if we walk in T from the leaf storing x to the root, then we will encounter the nodes, u'_x, u'_{ex}, v_i, u_{ex}, and u_x, in that order;*
2. *the path in T between u'_x and u_x contains $\mathcal{O}(\log n)$ nodes; and,*
3. *the path in T between u'_{ex} and u_{ex} contains $\mathcal{O}(1)$ nodes.*

Proof. Since $\alpha L/n^c \leqslant \alpha|xy| \leqslant (2/s)|xy| \leqslant (2/s)L$, it follows that we encounter u'_x, u'_{ex}, u_{ex} and u_x, in that order.

It follows from Lemma 3 that v_i is in the subtree of u_{ex} and w_i is in the subtree of u_{ey}. By Lemma 4, v_i is an ancestor of u'_{ex} *or* w_i is an ancestor of u'_{ey}. In fact, it follows from the construction of the well-separated pairs in [5] that v_i is an ancestor of u'_{ex} *and* w_i is an ancestor of u'_{ey}.

To prove the second claim we first note that $(\frac{2L}{s}/\frac{\alpha L}{n^c}) = \mathcal{O}(n^c)$, which together with Lemma 2 proves the claim. Similarly, the third claim also follows from Lemma 2 and the fact that $(\frac{2}{s}/\frac{2}{\sqrt{d}(s+4)}) = \mathcal{O}(1)$, since d is a constant. □

Lemma 8. *Let $e = \{x, y\}$ be a pair in F, and let i be the index such that $x \in A_i$ and $y \in B_i$. Let v_i and w_i be the nodes of T that represent A_i and B_i, respectively. Given pointers to the nodes u_{ex} and u_{ey}, the nodes v_i and w_i can be computed in $\mathcal{O}(1)$ time.*

Proof. According to Lemma 7, v_i is on the path between u_{ex} and u'_{ex}, and w_i is on the path between u_{ey} and u'_{ey}. Moreover, both these paths consist of $\mathcal{O}(1)$ nodes. The nodes v_i and w_i can be found by using the algorithm in [5] that computes the well-separated pairs from the split tree T. This algorithm starts at the nodes u_{ex} and u_{ey} and traverses the two paths down in T until two nodes are reached whose bounding boxes are well-separated. These two nodes are v_i and w_i. □

The original problem has now been reduced to finding, for each query pair $\{x, y\}$ in F, the nodes u_{ex} and u_{ey} in T, where u_{ex} and u_{ey} correspond to nodes whose bounding boxes are of size close to $(2/s)|xy|$. A simple solution would be as follows. For each point x in S, let $\mathcal{T}_x$ be a balanced binary search tree storing the nodes on the path in T between u'_x and u_x, which by Lemma 7.2 has only $\mathcal{O}(\log n)$ nodes. The key value for these nodes is the length of a longest side of the bounding box.

Lemma 9. *Let $e = \{x, y\}$ be a vertex pair in F. Using the trees $\mathcal{T}_x$ and $\mathcal{T}_y$, the nodes u_{ex} and u_{ey} can be computed in $\mathcal{O}(\log \log n)$ time.*

As a result, we have showed that our restricted problem can be solved in $\mathcal{O}(n \log n + k \log \log n)$ time. Each tree uses $\mathcal{O}(\log n)$ space. Thus, the amount of space used is $\mathcal{O}(n \log n)$. Next we will show that the size can be reduced to $\mathcal{O}(n)$ by observing that we have an off-line problem (i.e., all queries are known in advance).

4.2 Achieving linear space

Let $x_1, \ldots, x_n$ be the vertices stored in the leafs of T, ordered from left to right. Note that a query pair $e = \{x, y\}$ in F asks for u_{ex} and u_{ey}. This can be viewed as two different queries, i.e., (x, y) and (y, x).

Next we process the queries in batches. Initially we set $i = 1$. Build $\mathcal{T}_{x_1}$ in linear time. For each query in F of the form $e = (x_1, x_j)$, return u_{ex_1}. A pointer to u_{ex_1} is stored together with x_1 in the query pair (x_1, x_j) in F. When all queries involving x_1 have been answered, i is incremented.

In a generic step we build the binary tree $\mathcal{T}_{x_i}$ from $\mathcal{T}_{x_{i-1}}$ by first deleting the nodes in T that lie on the path between $u_{x_{i-1}}$ and $u'_{x_{i-1}}$, but not on the path between u_{x_i} and u'_{x_i}. Then, inserting all nodes in T that lie on the path between u_{x_i} and u'_{x_i}, but not on the path between $u_{x_{i-1}}$ and $u'_{x_{i-1}}$. Since each node in T is inserted and removed at most once, the total time complexity of building the trees $\mathcal{T}_1, \ldots, \mathcal{T}_n$ is $\mathcal{O}(n \log n)$.

When $\mathcal{T}_{x_i}$ is computed, the queries are answered and stored together with the pairs in F. The process continues until all queries are answered. At all times exactly one tree $\mathcal{T}_{x_i}$ is active, thus the total space complexity is dominated by the fair-split tree and the number of edges in F, which is bounded by $\mathcal{O}(k + n)$. As a result we obtain the following lemma.

Lemma 10. *Given the k query pairs $\{e_i = \{p_i, q_i\}\}_{1 \leqslant i \leqslant k}$ in F, one can compute $u_{e_i p_i}$ and $u_{e_i q_i}$ for each $1 \leqslant i \leqslant k$ in total $\mathcal{O}(n \log n + k \log \log n)$ time using $\mathcal{O}(k + n)$ space.*

4.3 Improving the running time

In this section we will improve the running time to $\mathcal{O}(k + n \log n)$; instead of using the tree $\mathcal{T}_x$ for answering the queries we will use a different data structure, namely an array $A_x\left[0..\left\lfloor \log\left(\frac{2n^c}{\alpha s L}\right)\right\rfloor\right]$. Recall that $\alpha = \frac{2}{\sqrt{d(s+4)}}$ and $s = \frac{1}{\varepsilon}((1+\varepsilon)(8t+4)+4)$. Each entry $A_x[j]$ stores a pointer to the highest node on the path in T between u'_x and u_x whose bounding box has sides of length at most $2^j \alpha L / n^c$.

Lemma 11. *Let $e = \{x, y\}$ be a pair in F, let*

$$j = \left\lfloor \log\left(\frac{2n^c}{\alpha s L}|xy|\right)\right\rfloor,$$

and let $A_x[j]$ point to node u^A_{ex}. Then the path between u_{ex} and u^A_{ex} contains $\mathcal{O}(1)$ nodes.

Proof. Since

$$\log\left(\frac{2n^c}{\alpha s L}|xy|\right) - 1 < j \leqslant \log\left(\frac{2n^c}{\alpha s L}|xy|\right),$$

it follows that $|xy|/s < 2^j \alpha L / n^c \leqslant 2|xy|/s$. The proof follows from Lemma 2. $\square$

Since node u^A_{ex} is close to u_{ex}, we can show the following lemma, which is similar to Lemma 8.

Lemma 12. *Let $e = \{x, y\}$ be a pair in F, and let i be the index such that $x \in A_i$ and $y \in B_i$. Let v_i and w_i be the nodes of T that represent A_i and B_i, respectively, and let j be as in Lemma 11. Given $A_x[j]$ and $A_y[j]$, the nodes v_i and w_i can be computed in $\mathcal{O}(1)$ time.*

Now, the above lemma assumes that the index j, defined by $j = \left\lfloor \log\left(\frac{2n^c}{\alpha s L}|xy|\right)\right\rfloor$ can be easily determined. We will refer to this index as the *index* of two points x and y in $\mathbb{R}^d$. It remains to prove how k index queries can be answered in total time $\mathcal{O}(k + n \log n)$.

4.4 Answering index queries efficiently

Next we consider how to "bucket" distances in constant time, without using the floor function. This problem was considered in [13], but there it was only shown for a special case, see Fact 15. We extend the result to hold for any point set for which the queries have polynomially bounded aspect ratio.

The aim of this section is to show the following theorem.

Theorem 2. *Let S be a set of n points in $\mathbb{R}^d$ and let $L > 0$ be a real number. We can preprocess S in $\mathcal{O}(n \log n)$ time, such that for any two points x and y in S with $L/n^c \leqslant |xy| \leqslant L$, we can compute the quantity*

$$\left\lfloor \log \left(\frac{2n^c}{\alpha s L} |xy| \right) \right\rfloor$$

in constant time.

For each $x \in S$, define $x' = \frac{2n^c}{\alpha s L} x$. This gives a set $V = \{x' : x \in S\}$ of scaled points. Let F' be the set of scaled query pairs $\{x', y'\}$, where $\{x, y\}$ ranges over all pairs in F. If $\{x, y\} \in F$, then $L/n^c \leqslant |xy| \leqslant L$ and, hence,

$$\frac{2}{\alpha s} \leqslant |x'y'| \leqslant \left(\frac{2}{\alpha s} \right) n^c \leqslant n^{c+1}.$$

Furthermore,

$$\left\lfloor \log \left(\frac{2n^c}{\alpha s L} |xy| \right) \right\rfloor = \lfloor \log |x'y'| \rfloor.$$

The one-dimensional case. We will assume that V is a set of n vertices on the line. First the algorithm partitions V into groups $V_1, \ldots, V_\ell$, in $\mathcal{O}(n \log n)$ time as follows.

Sort the points of V in increasing order $x_1, x_2, \ldots, x_n$. Let $j_1 < j_2 < \ldots < j_{\ell-1}$ be *all* the indices such that $x_{j_1+1} > x_{j_1} + n^c$, $x_{j_2+1} > x_{j_2} + n^c$, $\ldots$, $x_{j_{\ell-1}+1} > x_{j_{\ell-1}} + n^c$. In other words, the gaps following $x_{j_1}, x_{j_2}, \ldots, x_{j_{\ell-1}}$ are greater than n^c. Then we define $V_1 = \{x_1, \ldots, x_{j_1}\}$, $V_\ell = \{x_{j_{\ell-1}+1}, \ldots x_n\}$, and $V_i = \{x_{j_{i-1}+1}, \ldots, x_{j_i}\}$ for $1 \leqslant i \leqslant \ell - 1$; this is illustrated by an example in Fig. 2. The following observation about the sequence $V_1, \ldots, V_\ell$ follows immediately from the above partitioning algorithm.

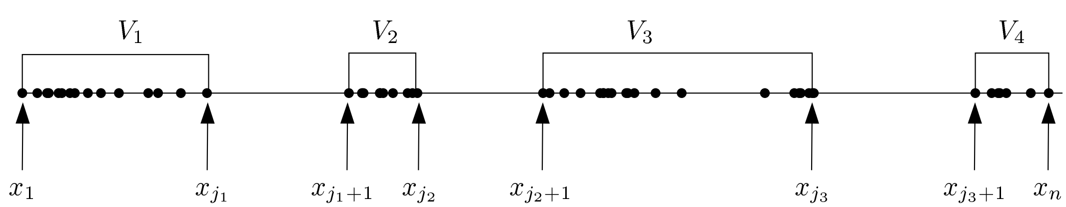

Fig. 2. Illustrating how V is divided into the sets $V_1, \ldots, V_4$. The gap between two sets is larger than n^c.

Consider the sequence $V_1, \ldots, V_\ell$ of vertex sets that are computed by this algorithm. The following observation follows immediately from the algorithm.

Observation 13 *Let i and j be two positive integers, such that $i < j \leqslant \ell$. Then, the following statements hold:*

1. *If $x \in V_i$ and $y \in V_j$, then $|xy| > n^c$.*
2. *If $x, y \in V_i$, then $|xy| \leqslant n^{c+1}$.*
3. *$V_i \cap V_j = \emptyset$, and $V = V_1 \cup \ldots \cup V_\ell$*

The following lemma gives properties of the scaled queries in F'. We know that no pair in F' lies in different subsets of the partition.

Lemma 14. *Consider the sets $V_1, \ldots, V_\ell$ and F' as defined above. Then,*

1. *For each i with $1 \leqslant i \leqslant \ell$ there exists a real number D_i such that the set V_i is contained in the interval $[D_i, D_i + n^{c+1}]$.*
2. *For every pair $\{x, y\}$ in F, there exists an i, such that both x' and y' are contained in V_i. Moreover, the pair $\{x', y'\}$ in F' satisfies*

$$2/(\alpha s) \leqslant |x'y'| \leqslant n^{c+1}, \quad \text{and} \quad \left\lfloor \log \left(\frac{2n^c}{\alpha s L} |xy| \right) \right\rfloor = \lfloor \log |x'y'| \rfloor.$$

Fact 15 *(Theorem 2.1 in [13]) Let X be a set of n real numbers that are contained in the interval $[D, D + n^k]$, for some real number D and some positive integer constant k. We can preprocess X in $\mathcal{O}(n \log n)$ time, using $\mathcal{O}(n)$ space, such that for any two points p and q of X, with $|pq| \geqslant \beta$, where $\beta > 0$ is a constant, we can compute $\lfloor \log |pq| \rfloor$ in constant time.*

In [13], Fact 15 was only proved for the case when $D = 0$. By translating the points, and observing that this does not change distances, it is clear that Fact 15 holds for any real number D. As a result of Lemma 14, it follows that Fact 15 can be applied to every subset V_i. Furthermore, according to Lemma 14, F' has polynomially bounded aspect ratio, thus every query pair in F' can be answered in constant time according to Fact 15. Note that, for each point x, we need to store a pointer to the subset V_i of the partition that it belongs to. As a result, we have proved Theorem 2 for the one-dimensional case.

The d-dimensional case. Again, this is inspired by the algorithm in [13]. Now assume that V is a d-dimensional set of points, where d is a constant. Let $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ be any two points of V with $|pq| \geqslant \beta$, for some constant $\beta > 0$, let j be such that $|p_j - q_j|$ is maximum, and let $i = \lfloor \log |p_j - q_j| \rfloor$. Since

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

we have

$$i \leqslant \lfloor \log |pq| \rfloor \leqslant \frac{1}{2} \log d + i.$$

This suggests the following solution. For each ℓ, $1 \leqslant \ell \leqslant d$, we build the data structure above for the one-dimensional case for the set of ℓ-th coordinates of all points of V.

Given two distinct points p and q of V, we compute the index j such that $|p_j - q_j|$ is maximum. Then we use the one-dimensional algorithm to compute the integer $i = \lfloor \log |p_j - q_j| \rfloor$. Note that this algorithm also gives us the value 2^i. Given i and 2^i, we then compute $\lfloor \log |pq| \rfloor$ in $\mathcal{O}(\log \log d)$ time. Observe that we can indeed apply the one-dimensional case, since $|p_j - q_j| \geqslant 1/\sqrt{d}$. This concludes the proof of Theorem 2.

We say that an edge $\{x, y\}$ *belongs* to a well-separated pair $\{A_i, B_i\}$ if and only if $x \in A_i$ and $y \in B_i$, or vice versa. Our results can be stated as follows:

Theorem 3. *Let S be a set of n points in $\mathbb{R}^d$, let F be a set of pairs of points in S having polynomially bounded aspect ratio, let T be the split tree for S, and let $\{A_i, B_i\}$, $1 \leqslant i \leqslant m$, be the corresponding well-separated pair decomposition of S. In $\mathcal{O}(n \log n + |F|)$ time, we can compute, for every $\{x, y\} \in F$, the index i such that $\{x, y\}$ belongs to the pair $\{A_i, B_i\}$.*

Proof. We have shown that the total time for computing the pairs in the WSPD belonging to the pairs in F can be written as $\mathcal{O}(n \log n + |F| \cdot \tau(n))$; here $\tau(n)$ is the time needed to answer an index query for a given pair $\{x, y\}$ in F. By Theorem 2, we know that $\tau(n) = O(1)$. Hence the result. $\square$

Since we have an off-line problem, we can use the approach of Section 4.2 to reduce the space requirement to $\mathcal{O}(n + |F|)$.

4.5 The general case – unbounded aspect ratio

As a result of the previous section it holds that a t-spanner can be pruned efficiently in the case when the "aspect ratio" of the edge set is polynomially bounded. To generalize this result we will use the following technical theorem that is implicit in [12].

Theorem 4. *Let S be a set of n points in $\mathbb{R}^d$, and let $c \geqslant 7$ be an integer constant. In $\mathcal{O}(n \log n)$ time, we can compute a data structure $D(S)$ consisting of:*

1. *a sequence $L_1, L_2, \ldots, L_\ell$ of real numbers, where $\ell = \mathcal{O}(n)$, and*
2. *a sequence $S_1, S_2, \ldots, S_\ell$ of subsets of S such that $\sum_{i=1}^{\ell} |S_i| = \mathcal{O}(n)$,*

such that the following holds. For any two distinct points p and q of S, we can compute in $\mathcal{O}(1)$ time an index i with $1 \leqslant i \leqslant \ell$ and two points x and y in S_i such that

a. $L_i / n^{c+1} \leqslant |xy| < L_i$, and

b. both $|px|$ and $|qy|$ are less than $|xy|/n^{c-2}$.

Proof. We start by computing, in $\mathcal{O}(n \log n)$ time, an arbitrary t'-spanner H for S having $\mathcal{O}(n)$ edges, where t' is a constant, say $t' = 2$. By applying the algorithms in Section 3.1 in [12] to the graph H, we obtain, in $\mathcal{O}(n \log n)$ time, sequences $L'_1, L'_2, \ldots, L'_\ell$ and $S_1, S_2, \ldots, S_\ell$. By Lemma 1 in [12], we have $\ell = O(n)$. By the analysis in Section 3.6 in [12], we have $\sum_{i=1}^{\ell} |S_i| = \mathcal{O}(n)$. We define $L_i := L'_i / t'$ for $1 \leqslant i \leqslant \ell$.

Let p and q be two distinct points of S. It follows from Lemma 10 and the proof of Lemma 8 in [12] that two lowest common ancestor computations in a tree of size $O(n)$ yield an index i and two points x and y such that $L'_i / n^{c+1} \leqslant |xy| < L'_i / t'$ and both $|px|$ and $|qy|$ are less than L'_i / n^{2c-1}. It follows that both $|px|$ and $|qy|$ are less than $L'_i / n^{2c-1} \leqslant |xy| / n^{c-2}$. Moreover, by our definition of L_i, it follows that

$$L_i / n^{c+1} = L'_i / (t' n^{c+1}) \leqslant L'_i / n^{c+1} \leqslant |xy| < L'_i / t' = L_i.$$

$\square$

Figure 3 shows the complete algorithm, referred to as algorithm PRUNEGRAPH. Recall that the input to algorithm PRUNEGRAPH is a t-spanner $G = (S, E)$ and a positive real value ε.

Algorithm PruneGraph($G = (S, E), t, \varepsilon$)

Step 1: Compute the data structure $D(S)$ with $c = 7$, according to Theorem 4.
Step 2: For each $1 \leqslant i \leqslant \ell$, set $F_i := \emptyset$.
Step 3: For each edge $(p, q) \in E$, compute (i, x_p, y_q), according to Theorem 4, and add $\{x_p, y_q\}$ to F_i.
Step 4: For $i := 1$ to ℓ do
Step 4a. Compute the split tree T_i for S_i.
Step 4b. Compute the well-separated pair decomposition of S_i, $W_i(S_i) := \{\{A_1^i, B_1^i\}, \ldots, \{A_{m_i}^i, B_{m_i}^i\}\}$, using separation constant $2s$, where $s = ((1+\varepsilon)(8t+4)+4)/\varepsilon$.
Step 4c. For each $\{x, y\} \in F_i$, compute the pair $\{A_j^i, B_j^i\}$ that it belongs to.
Step 5: For each $1 \leqslant i \leqslant \ell$ and each $1 \leqslant j \leqslant m_i$, set $C_j^i := \emptyset$
Step 6: For each edge $(p, q) \in E$, compute (i, x_p, y_q), and add (p, q) to C_j^i, where j is the index such that $\{x_p, y_q\}$ belongs to $\{A_j^i, B_j^i\}$.
Step 7: Output $G' = (S, E')$, where E' contains exactly one edge from each non-empty set C_j^i.

Fig. 3. Algorithm PruneGraph.

Theorem 5. *Algorithm* PruneGraph *requires $\mathcal{O}(|E|)$ space and runs in $\mathcal{O}(|E| + n \log n)$ time.*

Proof. Steps 1—3 take $\mathcal{O}(|E| + n \log n)$ time and linear space according to Theorem 4. Steps 4a—c can be done in $\mathcal{O}(|S_i| \log |S_i| + |F_i|)$ time and $\mathcal{O}(|S_i| + |F_i|)$ space per iteration, provided that the well-separated pairs are stored implicitly. Since $\sum_{1 \leqslant i \leqslant \ell} |S_i| = \mathcal{O}(n)$ and $\sum_{1 \leqslant i \leqslant \ell} |F_i| = |E|$, the total time for Step 4 is $\mathcal{O}(|E| + n \log n)$. Steps 5—7 take time $\mathcal{O}(\sum_{i=1}^{\ell} m_j + |E|) = \mathcal{O}(n + |E|)$. We conclude that the algorithm has a total running time of $\mathcal{O}(|E| + n \log n)$ and utilizes $\mathcal{O}(|E|)$ space. □

Theorem 6. *The graph $G' = (S, E')$ computed by algorithm* PruneGraph($G = (S, E), t, \varepsilon$) *is a $(1+\varepsilon)$-spanner of the t-spanner $G = (S, E)$ such that $E' \subseteq E$ and $|E'| = \mathcal{O}(n)$.*

Proof. For each $1 \leqslant i \leqslant \ell$ and each $1 \leqslant j \leqslant m_i$, consider the j-th well-separated pair $\{A_j^i, B_j^i\}$ in the i-th length partition. Let a_j^i be an arbitrary point in A_j^i and let b_j^i be an arbitrary point in B_j^i. Define $P := \{\{a_j^i, b_j^i\} : 1 \leqslant i \leqslant \ell, 1 \leqslant j \leqslant m_i\}$. First, observe that

$$|P| = \sum_{i=1}^{\ell} m_i = \mathcal{O}\left(\sum_{i=1}^{\ell} |S_i|\right) = \mathcal{O}(n).$$

We will show that the set P satisfies the premises of the general framework of Section 3. This will imply that the graph G' is obtained by pruning G with respect to P, as described in the beginning of Section 3, and, therefore, by Lemma 5, G' is a $(1+\varepsilon)$-spanner of G.

Let (p, q) be an arbitrary edge of E. By Theorem 4, there exists an index i, and two points x and y in S_i, such that $|px| < |xy|/n^5$ and $|qy| < |xy|/n^5$. By the definition of the WSPD, there exists an index j such that (i) $x \in A_j^i$ and $y \in B_j^i$ or (ii) $x \in B_j^i$ and $y \in A_j^i$. We may assume without loss of generality that (i) holds.

Consider the point a_j^i in the set A_j^i and the point b_j^i in the set B_j^i. Since we chose the separation ratio for the WSPD to be $2s$, we know from Lemma 1 that $|xa_j^i| \leqslant |a_j^i b_j^i|/s$ and

$|xy| \leqslant (1+2/s)|a^i_j b^i_j|$. It follows that

$$\begin{aligned} |pa^i_j| &\leqslant |px| + |xa^i_j| \\ &\leqslant |xy|/n^5 + |a^i_j b^i_j|/s \\ &\leqslant \left((1+2/s)/n^5 + 1/s\right) |a^i_j b^i_j| \\ &\leqslant (2/s)|a^i_j b^i_j|, \end{aligned}$$

where the last inequality assumes that n is sufficiently large.

In exactly the same way, it can be shown that $|qb^i_j| \leqslant (2/s)|a^i_j b^i_j|$.

This completes the proof of the theorem. □

5 Sparse spanners with low weight

Corollary 1. *Given a geometric t-spanner $G = (S, E)$ of the set S of n points in $\mathbb{R}^d$, for some constant $t > 1$, and given a positive real constant $\varepsilon' > 0$, we can compute in $\mathcal{O}(n \log n + |E|)$ time, a $(1+\varepsilon')$-spanner of G having $\mathcal{O}(n)$ edges and whose weight is $\mathcal{O}(wt(MST(S)))$.*

Proof. The input graph G is pruned in two steps; first apply the pruning algorithm described above with $\varepsilon = \sqrt{1+\varepsilon'} - 1$ to obtain G', followed by applying the greedy algorithm of [11] to G', again with $\varepsilon = \sqrt{1+\varepsilon'} - 1$. The first step ensures that G' is a $(1+\varepsilon)$-spanner of G and that the total number of edges in the G' is $\mathcal{O}(n)$. The second step prunes G' such that the resulting graph G'' is a $(1+\varepsilon)$-spanner of G' with weight $\mathcal{O}(wt(MST(S))$. Since $(1+\varepsilon)^2 = 1+\varepsilon'$ it holds that G'' is a $(1+\varepsilon')$-spanner of G.

The weight bound follows from the results in [7–9, 11]. That is, the greedy algorithm computes a subgraph G'' of G' which is a $(1+\varepsilon)$-spanner of G' and that satisfies the so-called leapfrog property. The graph G'' can be computed in $\mathcal{O}(|E| + n \log n)$ time. Das and Narasimhan [8] have shown that any set of edges that satisfies the leapfrog property has weight $\mathcal{O}(wt(MST(S)))$, where $MST(S)$ is a minimum spanning tree of S. □

6 Applications

The tool presented in this paper for pruning a spanner graph is important as a preprocessing step in many situations. We briefly mention a few. In [13], the algorithm to approximate the length of the shortest path between two query points in a given geometric graphs requires preprocessing time of $\mathcal{O}(|E| \log n)$. The results here would reduce the preprocessing time to $\mathcal{O}(|E| + n \log n)$. As a second application, a similar improvement can be achieved for the algorithm to compute an approximation to the stretch factor of a spanner graph [13, 17]. Using this result, the approximate stretch factor can be computed in $\mathcal{O}(|E| + n \log n)$ time. Finally, similar improvements are achieved for several variants of the closest pair problems. In the monochromatic version, a given geometric graph $G = (V, E)$ (with n vertices corresponding to n points in $\mathbb{R}^d$) is to be preprocessed, in order to answer closest pair queries for a query subset $S \subseteq V$ where distances between points are defined as the length of the shortest path in G. In the bichromatic version, the graph G is to be preprocessed, in order to answer closest pair queries between two given query subsets $X, Y \subseteq V$. Using this result, the preprocessing can be done in $\mathcal{O}(|E| + n \log n)$ time instead of $\mathcal{O}(|E| \log n)$.

In all the above cases, the idea would be to first prune the graph using the algorithm in this paper to obtain a spanner graph with approximately the same stretch factor, but with only a linear number of edges, consequently speeding up the previously designed algorithms.

7 Conclusions

Given a t-spanner $G = (S, E)$, where S is a set of n points in $\mathbb{R}^d$, we have shown how to compute in $\mathcal{O}(|E| + n \log n)$ time a $(1+\varepsilon)$-spanner of G having $\mathcal{O}(n)$ edges and whose weight is proportional to the weight of a minimum spanning tree of S. The interesting fact about this result is that it shows that the pruning problem can be solved *without* sorting the edges of E. We leave open the problem of pruning a spanner in $\mathcal{O}(|E|)$ time.

References

1. I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
2. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symposium on Theory of Computing*, pages 489–498, 1995.
3. S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th IEEE Symposium on Foundations of Computer Science*, pages 703–712, 1994.
4. P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry: Theory and Applications*, 28:11–18, 2004.
5. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.
6. B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *International Journal of Computational Geometry and Applications*, 5:124–144, 1995.
7. G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. 9th Annual ACM Symposium on Computational Geometry*, pages 53–62, 1993.
8. G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7:297–315, 1997.
9. G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 1995.
10. D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers, Amsterdam, 2000.
11. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.
12. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric spanners. Submitted, 2004. `http://www.win.tue.nl/~hgudmund/glns-adogs-04.pdf`.
13. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graph. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.
14. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles revisited. In *Proc. 13th International Symposium on Algorithms and Computation*, volume 2518 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
15. J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithmic Theory*, pages 208–213, 1988.
16. C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.
17. G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30:978–989, 2000.
18. J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1:99–107, 1991.
19. M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science Publishers, Amsterdam, 2000.
20. P. M. Vaidya. A sparse graph almost as good as the complete graph on points in K dimensions. *Discrete Computational Geometry*, 6:369–381, 1991.