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 stretch factor t, and a positive constant ε, we show how to construct a $(1+\varepsilon)$-spanner of G with $\mathcal{O}(|S|)$ edges in time $\mathcal{O}(|E| + |S| \log |S|)$. Previous algorithms require a preliminary step in which the edges are sorted in non-decreasing order of their lengths and, thus, have running time $\Omega(|E| \log |S|)$. We obtain our result by designing a new algorithm that finds the pair in a well-separated pair decomposition separating two given query points. Previously, it was known how to answer such a query in $\mathcal{O}(\log |S|)$ time. We show how a sequence of such queries can be answered in $\mathcal{O}(1)$ amortized time per query, provided all query pairs are from a polynomially bounded range.

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, 7, 15].

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. Then, 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 *stretch factor* 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 points p and q of S.

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

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, 6–8, 10, 14, 15, 17, 19], and the surveys [9, 18]. All these algorithms compute t-spanners for any given constant $t > 1$. However, all these algorithms either start with a point set, or with a spanner that has a linear number of edges.

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 stretch factor 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. The greedy algorithm of [7, 10] can be used to compute a $(1 + \varepsilon)$-spanner G' of G. However, the greedy algorithm starts by sorting the edges and, thus, has running time $\Omega(|E| \log n)$. In [11], 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 [10], 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. [11–13], 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 [11–13] to $\mathcal{O}(|E| + n \log n)$. We also improve the time complexity in [16] for computing a $(1 + \varepsilon)$-approximation to the stretch factor of a geometric graph to $\mathcal{O}(|E| + n \log n)$, provided we know in advance that the stretch factor is bounded from above by a constant. In Section 5 we consider several other applications.

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.*

Lemma 1 ([5]). *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, $\{A_1, B_1\}, \ldots, \{A_m, B_m\}$, 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 $i = 1, \ldots, m$.*

Callahan and Kosaraju showed that the fair 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 two lemmas that will be used later on.

Lemma 2. *Let u and u' be two nodes in the fair 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 the longest side of the bounding box of u' is at most half the length of the longest side of the bounding box of u.*

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 at most $(2/s)|pq|$.*

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 constant $t > 1$ and a real constant $\varepsilon > 0$.

Our goal is to compute a sparse $(1+\varepsilon)$-spanner G' of G. Suppose that there exists a set pairs of points, $P = \{\{a_1, b_1\}, \ldots, \{a_m, b_m\}\}$, with the property that for each edge (p, q) in E, there is an index i such that for some real number 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) in E, the set P contains a "close approximation". Then, we show below that, if $s = \frac{1}{\varepsilon}((1+\varepsilon)(8t+4)+4)$, 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* the set P of pairs 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 1. or 2. 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 4. *The graph $G' = (S, E')$ is a $(1+\varepsilon)$-spanner of G.*

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 an index i such that the condition for the set P holds.

A straight-forward approach for step 1, which appears already in [11], is as follows. Compute the WSPD with separation constant $s = ((1+\varepsilon)(8t+4)+4)/\varepsilon$, 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 4.

As for step 2, Arya *et al.* [3] showed that, after an $\mathcal{O}(n \log n)$-time preprocessing of the fair split tree, the index i can be computed in $\mathcal{O}(\log n)$ time, for any edge (p, q) in E. 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. *[11] Given a geometric graph $G = (S, E)$ with n vertices, which is a t-spanner for S, for some real constant $t > 1$, and a real constant $\varepsilon > 0$ we can compute a $(1+\varepsilon)$-spanner of G having $\mathcal{O}(n)$ edges in $\mathcal{O}(|E| \log n)$ time.*

4 An improved algorithm

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

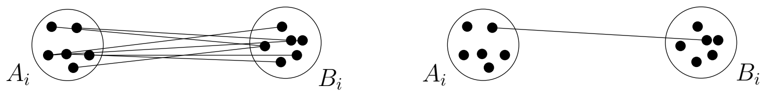

Fig. 1. Pruning the spanner graph using the WSPD.

4.1 Improvements for a restricted case – bounded aspect ratio

Let T be the fair 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 $\alpha = 2/(\sqrt{d}(s+4))$. 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 5 *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. This is formalized in the lemma below.

Lemma 6. *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 this 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.*

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. 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.*

The original problem has now been reduced to finding, for each query pair $e = \{x, y\}$ 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 6.2 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 8. *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 shown 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 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).

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}; and 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 $\mathcal{O}(n \log\log n)$.

After $\mathcal{T}_{x_i}$ has been constructed, all queries involving x_i are solved, and the answers are stored together with the pairs in F. The process continues until all queries have been 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 $\mathcal{O}(k + n)$. We obtain the following lemma.

Lemma 9. *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 in Lemma 9 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 $A_x\,[0..\lfloor\log(2n^c)\rfloor]$. Recall that $s = \frac{1}{\varepsilon}((1+\varepsilon)(8t+4)+4)$. 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 $\frac{2^j L}{s n^c}$.

Lemma 10. *Let $e = \{x, y\}$ be a pair in F, let $j = \left\lfloor \log\left(\frac{2n^c}{L}|xy|\right)\right\rfloor$, 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.*

Since node u^A_{ex} is close to u_{ex}, we can show the following lemma, which is similar to Lemma 7.

Lemma 11. *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 10. 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 $j = \left\lfloor \log\left(\frac{2n^c}{L}|xy|\right)\right\rfloor$ 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 since the floor function is a non-algebraic function. This problem was considered in [11], but there it was only shown for the special case when the points in the set lie in a polynomially bounded interval, see Fact 14. We extend the result to hold for any point set for which the queries have polynomially bounded aspect ratio. The idea is to scale the point set and then partition it into subsets such that each subset consists of points in a polynomially bounded interval. Furthermore, for every pair in F it will be shown that the two corresponding points in the scaled set will belong to the same subset. Consequently, one may apply the results from [11] to each subset.

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

Theorem 2. *Let S be a set of n points in $\mathbb{R}^d$, let $L > 0$ be a real number, and let c be a positive constant. 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 $\left\lfloor \log\left(\frac{2n^c}{L}|xy|\right)\right\rfloor$ in constant time, using only algebraic operations and indirect addressing.*

For each $x \in S$, define $x' = \frac{2n^c}{L}x$. 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, $2 \leqslant |x'y'| \leqslant 2n^c \leqslant n^{c+1}$. Furthermore, $\left\lfloor \log\left(\frac{2n^c}{L}|xy|\right)\right\rfloor = \lfloor \log |x'y'| \rfloor$.

The one-dimensional case. We will assume that V is a set of n points on the real line. First the algorithm partitions V into groups $V_1, \dots, V_\ell$, in $\mathcal{O}(n\log n)$ time as follows. Sort the points of V in increasing order $x'_1, x'_2, \dots, x'_n$. Let $j_1 < j_2 < \dots < j_{\ell-1}$ be *all* the indices such that $x'_{j_1+1} > x'_{j_1} + n^{c+1}$, $x'_{j_2+1} > x'_{j_2} + n^{c+1}$, $\dots$, $x'_{j_{\ell-1}+1} > x'_{j_{\ell-1}} + n^{c+1}$. In other words, the gaps following $x'_{j_1}, x'_{j_2}, \dots, x'_{j_{\ell-1}}$ are greater than n^{c+1}. Then we define $V_1 = \{x'_1, \dots, x'_{j_1}\}$,

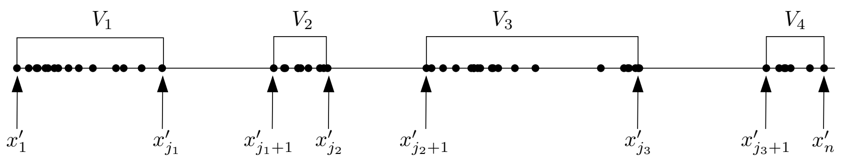

Fig. 2. Illustrating how V is divided into the sets $V_1, \ldots, V_4$. The gap between two sets is larger than n^{c+1}.

$V_\ell = \{x'_{j_{\ell-1}+1}, \ldots x'_n\}$, and $V_i = \{x'_{j_{i-1}+1}, \ldots, x'_{j_i}\}$ for $2 \leqslant i \leqslant \ell - 1$; 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.

Observation 12 *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 $|x'y'| > n^{c+1}$.*
2. *If $x', y' \in V_i$, then $|x'y'| \leqslant n^{c+2}$.*
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'.

Lemma 13. *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 $[D_i, D_i + n^{c+2}]$.*
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*

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

Fact 14 *(Theorem 2.1 in [11]) 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 [11], Fact 14 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 14 holds for any real number D. As a result of Lemma 13, it follows that Fact 14 can be applied to every subset V_i. Furthermore, according to Lemma 13, F' has polynomially bounded aspect ratio, thus every query pair in F' can be answered in constant time according to Fact 14. 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. This is inspired by the algorithm in [11]. 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 $|p_j - q_j| \geqslant \beta/\sqrt{d}$. 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 fair 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 $\{A_i, B_i\}$.*

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 [13].

Theorem 4. *Given a set S of n points in $\mathbb{R}^d$ and an integer constant $c \geqslant 7$ we can compute a data structure $D(S)$ in $\mathcal{O}(n \log n)$ time 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 $|xy|/n^{c-2}$.

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 Thm. 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
          Thm. 4, and add $\{x_p, y_q\}$ to $F_i$.
Step 4:   For $i := 1$ to $\ell$ do
Step 4a.      Compute the fair split tree $T_i$ for $S_i$.
Step 4b.      Compute the well-separated pair decomposition of $S_i$,
              $W_i(S_i) := \{\{A^i_1, B^i_1\}, \ldots, \{A^i_{m_i}, B^i_{m_i}\}\}$, using separation
              constant $2s$, where $s = ((1 + \varepsilon)(8t + 4) + 4)/\varepsilon$.
Step 4c.      For each $\{x, y\} \in F_i$, compute the pair $\{A^i_j, B^i_j\}$ that it belongs to.
Step 5:   For each $1 \leqslant i \leqslant \ell$ and each $1 \leqslant j \leqslant m_i$, set $C^i_j := \emptyset$
Step 6:   For each edge $(p, q) \in E$, add $(p, q)$ to $C^i_j$, where $j$
          is the index such that $\{x_p, y_q\}$ belongs to $\{A^i_j, B^i_j\}$.
Step 7:   Output $G' = (S, E')$, where $E'$ contains exactly one edge from each
          non-empty set $C^i_j$.
```

Fig. 3. Algorithm PruneGraph.

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

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 $\{A^i_j, B^i_j\}$ in the i-th length partition. Let a^i_j be an arbitrary point in A^i_j and let b^i_j be an arbitrary point in B^i_j. Define $P := \{\{a^i_j, b^i_j\} : 1 \leqslant i \leqslant \ell, 1 \leqslant j \leqslant m_i\}$. 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 4, 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) $x \in A^i_j$ and $y \in B^i_j$ or (ii) $x \in B^i_j$ and $y \in A^i_j$. We may assume that (i) holds.

Consider the point a^i_j in the set A^i_j and the point b^i_j in the set B^i_j. 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 $|xy| \leqslant (1+2/s)|a_j^i b_j^i|$. It follows that $|pa_j^i| \leqslant |px| + |xa_j^i| \leqslant |xy|/n^5 + |a_j^i b_j^i|/s \leqslant \big((1+2/s)/n^5 + 1/s\big)\,|a_j^i b_j^i| \leqslant (2/s)|a_j^i b_j^i|$, where the last inequality assumes that n is sufficiently large. In exactly the same way, it can be shown that $|qb_j^i| \leqslant (2/s)|a_j^i b_j^i|$.

This completes the proof of the theorem. □

The above theorem can be combined with results in [10] to prove the following corollary.

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 $\mathcal{O}(wt(MST(S)))$.*

5 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 [11], the algorithm to approximate the length of the shortest path between two query points in a given geometric spanner has a preprocessing time of $\mathcal{O}(|E| \log n)$. The results here 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 [11, 16]. Using this result, an approximate stretch factor can be computed in $\mathcal{O}(|E| + n \log n)$ time, provided the stretch factor is bounded by a constant. Finally, similar improvements are achieved for several variants of the closest pair problem. In the monochromatic version, a given geometric spanner $G = (V, E)$ (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 is to first prune the spanner 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.

6 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. 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 and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7:297–315, 1997.
8. G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.
9. 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.
10. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM Journal of Computing*, 31(5):1479–1500, 2002.
11. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graph. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 828–837, 2002.
12. 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 *LNCS*, pages 357–368. Springer-Verlag, 2002.
13. J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric spanners. Technical report TR-04-08, Carleton University, School of Computer Science, 2004.
14. J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st Scandinavian Workshop on Algorithmic Theory*, pages 208–213, 1988.
15. C. Levcopoulos, G. Narasimhan, and M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32:144–156, 2002.
16. G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30:978–989, 2000.
17. J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry & Applications*, 1:99–107, 1991.
18. 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.
19. 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.