Approximation algorithms for the bottleneck stretch factor problem

Giri Narasimhan* Michiel Smid†

August 23, 2000

Abstract

The stretch factor of a Euclidean graph is the maximum ratio of the distance in the graph between any two points and their Euclidean distance. Given a set S of n points in $\mathbb{R}^d$, we show how to construct a data structure of size $O(\log n)$, such that for an arbitrary query value $b > 0$, we can in $O(\log \log n)$ time compute an approximation of the stretch factor of the graph G_b, which is the threshold graph on S containing all edges of length at most b. Even though there could be up to $\binom{n}{2}$ different stretch factors, we show that this data structure can be constructed in subquadratic time. If we think of the points of S as being airports, then the stretch factor of G_b gives a measure of the maximum percentage increase in flight distance using flight segments of length at most b over the direct distance.

Our algorithm uses techniques from computational geometry, such as well-separated pairs, minimum spanning trees, data structures for the nearest-neighbor problem, and algorithms for selecting and ranking distances.

Classification: Algorithms and data structures, computational geometry.

*Department of Mathematical Sciences, The University of Memphis, Memphis TN 38152. E-mail: `giri@msci.memphis.edu`.

†Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany. E-mail: `michiel@isg.cs.uni-magdeburg.de`.

1 Introduction

Assume that we are given the coordinates of n airports. Given an airplane that can fly a distance of b miles without refueling, a typical query is to determine the smallest value of t such that the airplane can travel between any pair of airports using flight segments of length at most b miles, such that the sum of the lengths of the flight segments is not longer than t times the direct "as-the-crow-flies" distance between the airports. This problem falls under the general category of *bottleneck problems*. In our case, the *stretch factor*, i.e., the value of t, is a measure of the maximum increase in fuel costs caused by choosing a path other than the direct path between any source and any destination. (Clearly, this direct path cannot be taken if its length is larger than b miles.)

Let us formalize this problem. For simplicity, we take the Euclidean metric for the distance between two airports. In practice, one needs to take into account the curvature of the earth and the wind conditions.

Let $d \geq 2$ be a small constant. For any two points p and q in $\mathbb{R}^d$, we denote their Euclidean distance by $|pq|$. Let S be a set of n points in $\mathbb{R}^d$, and let G be an undirected graph having S as its vertex set. The length of any edge (p,q) of G is defined as $|pq|$. Furthermore, the length of any path in G between two vertices p and q is defined as the sum of the lengths of the edges on this path. We call such a graph G a *Euclidean graph*. For any two vertices p and q of G, we denote by $|pq|_G$ their distance in G, i.e., the length of a shortest path connecting p and q. If there is no path between p and q, then $|pq|_G = \infty$. The *stretch factor* t^* of G is defined as

$$t^* := \max\left\{\frac{|pq|_G}{|pq|} : p \in S, q \in S, p \neq q\right\}.$$

Note that $t^* = \infty$, if the graph G is not connected.

The *bottleneck stretch factor problem* is to preprocess the points of S into a data structure, such that for any real number $b > 0$, we can efficiently compute the stretch factor of the subgraph of the complete graph on S containing all edges of length at most b.

Let $G = (S,E)$ denote the Euclidean graph on S containing all edges having length at most b. The time complexity of solving the *all-pairs-shortest-path* problem for G is an upper bound on the time complexity of computing the stretch factor of G. Hence, running Dijkstra's algorithm—implemented with Fibonacci heaps—from each vertex of G, gives the stretch factor of G, in $O(n^2 \log n + n|E|)$ time (c.f., [9]). Note that $|E|$ can be as large as $\binom{n}{2}$. Hence, without any preprocessing, we can answer queries in $O(n^3)$ time. It may be possible to improve the query time, but we are not aware of any algorithm that computes the stretch factor in subquadratic time. (For example, we do not even know if the stretch factor of a Euclidean path can be computed in $o(n^2)$ time.)

A second solution for the bottleneck stretch factor problem is obtained from the observation that there are only $\binom{n}{2}$ "different" query values b. Hence, if we store all $\binom{n}{2}$ different stretch factors, then a query can be solved in $O(\log n)$ time by searching with the query value b in the sorted sequence of all $\binom{n}{2}$ Euclidean distances between the pairs of points of S. Clearly, in this case, the preprocessing time and the amount of space used are at least quadratic in n.

This leads to the question whether more efficient solutions exist, if we are satisfied with an approximation to the stretch factor of the graph G.

Let $c_1 \geq 1$ and $c_2 \geq 1$ be real numbers, let G be an arbitrary Euclidean graph on the point set S, and let t^* be the stretch factor of G. We say that the real number t is a (c_1, c_2)-*approximate stretch factor* of G, if $t/c_1 \leq t^* \leq c_2 t$. The current paper considers the following problem:

Problem 1 *The* (c_1, c_2)-approximate bottleneck stretch factor problem *is to preprocess the points of* S *into a data structure, such that for any real number* $b > 0$*, we can efficiently compute a* (c_1, c_2)*-approximate stretch factor of the subgraph of the complete graph on* S *containing all edges of length at most* b*.*

1.1 Our results

In this paper, we will present a data structure that solves Problem 1. The general approach, which is given in Section 3, is as follows. We partition the sequence of $\binom{n}{2}$ exact stretch factors into $O(\log n)$ subsequences, such that any two stretch factors in the same subsequence are approximately equal. Our data structure contains a sequence of $O(\log n)$ stretch factors, one from each subsequence. We also store a corresponding sequence of $O(\log n)$ distances between pairs of points. The latter sequence is used to search in $O(\log \log n)$ time in the sequence of $O(\log n)$ stretch factors. The result is a data structure of size $O(\log n)$ that can be used to solve the queries of Problem 1 in $O(\log \log n)$ time. The time to build this data structure, however, is at least quadratic in n.

In Section 4, we show that it suffices to use a sequence of $O(\log n)$ approximate stretch factors instead of the sequence of $O(\log n)$ exact stretch factors. Since the graphs whose stretch factors we have to approximate may have a quadratic number of edges, however, we need to make one more approximation step. That is, in Section 5, we use Callahan and Kosaraju's well-separated pair decomposition [7] to approximate the graph G containing all edges of length at most b by a graph H having $O(n \log n)$ edges and having approximately the same stretch factor. Then we use the algorithm of Narasimhan and Smid [13] to compute an approximate stretch factor of the graph H. In this way, we obtain the main result of this paper: a data structure of size $O(\log n)$, query time $O(\log \log n)$, and that can be built in subquadratic time.

1.2 Related work

There has been substantial work on the problem of constructing a Euclidean graph on a given set of points whose stretch factor is bounded by a given constant $t > 1$. A good overview of results in this direction can be found in the surveys by Eppstein [11] and Smid [15].

The problem of approximating the stretch factor of any given Euclidean graph has been considered by the authors in [13]. There, we prove the following result, which will be used in the current paper.

Theorem 1 ([13]) *Let* S *be a set of* n *points in* $\mathbb{R}^d$*, let* $G = (S, E)$ *be an arbitrary connected Euclidean graph, let* $\beta \geq 1$ *be an integer constant, and let* ϵ *be a real constant, such that* $0 < \epsilon \leq 1/2$*. In* $O(|E| n^{1/\beta} \log^2 n)$ *expected time, we can compute a* $(2\beta(1+\epsilon), 1+\epsilon)$*-approximate stretch factor of* G*.*

The proof of this theorem uses the well-separated pair decomposition (WSPD) of Callahan and Kosaraju [7]. We use this WSPD in Section 5 to approximate the graph containing all edges of length at most b by a graph having $O(n \log n)$ edges and having approximately the same stretch factor. For other applications of the WSPD, see [2, 5, 6, 7].

To the best of our knowledge, the exact and approximate bottleneck stretch factor problems have not been considered before.

2 Some preliminary results

We start by introducing some notation and terminology. Let S be a set of n points in $\mathbb{R}^d$, and let m be the number of distinct distances defined by any two distinct points of S. Let $\delta_1 < \delta_2 < \ldots < \delta_m$ be the sorted sequence of these distances. Note that $m \leq \binom{n}{2}$.

Let G_0 be the graph on S having no edges. Furthermore, for any i, $1 \leq i \leq m$, let G_i be the i-th *bottleneck graph*, i.e., the subgraph of the complete graph on S containing all edges of length at most δ_i. Clearly, for any i, $0 \leq i < m$, G_i is a subgraph of G_{i+1}, and G_m is the complete graph on S. For any i, $0 \leq i \leq m$, we denote by t_i^* the (exact) stretch factor of the graph G_i. The sequence $\mathcal{T} = \langle t_0^*, t_1^*, t_2^*, \ldots, t_m^* \rangle$ will be referred to as the *stretch factor spectrum* of S.

It is clear that determining the stretch factor spectrum of S solves the exact version of the bottleneck stretch factor problem. However, this involves determining the stretch factor of $\Theta(n^2)$ distinct graphs, which is likely to be prohibitively expensive.

First, we observe that $t_0^* = \infty$, $t_m^* = 1$, and $t_{i+1}^* \leq t_i^*$ for all i, $0 \leq i < m$. Also, the graph G_0 is not connected, whereas the graph G_m is connected. Let k be the smallest index such that the graph G_k is connected. Then $t_0^* = t_1^* = \ldots = t_{k-1}^* = \infty$, t_k^* is finite, and $1 = t_m^* \leq t_{m-1}^* \leq \ldots \leq t_{k+1}^* \leq t_k^*$. We will henceforth refer to the distance δ_k (corresponding to index k) as the *connectivity threshold*.

The following lemma characterizes the connectivity threshold. It is a restatement of the well-known folklore theorem that states that the minimum spanning tree is also a bottleneck minimum spanning tree.

Lemma 1 *Let T be a minimum spanning tree of S. Then the longest edge in T has length δ_k.*

Using Lemma 1, we can prove an upper bound on the stretch factor t_k^* of the bottleneck graph G_k. The bound is useful because it suggests that binary search on the stretch factor spectrum can be performed efficiently.

Lemma 2 *We have $t_k^* \leq n - 1$.*

3 A first solution

We start by describing the general idea of our solution to the approximate bottleneck stretch factor problem. Let $c > 1$ be an arbitrary constant. For the preprocessing phase, we partition the index set $\{k, k+1, \ldots, m\}$ into $O(\log n)$ subsets of consecutive integers, such that for any two indices i and i' of the same subset, the stretch factors t_i^* and $t_{i'}^*$ are within a factor of c of each other. This partition induces partitions of the two sequences δ_i, $k \leq i \leq m$, and t_i^*, $k \leq i \leq m$, into $O(\log n)$ subsequences. For each j, we let a_j denote the smallest index in the j-th subset of the partition of $\{k, k+1, \ldots, m\}$.

Our data structure consists of the $O(\log n)$ values δ_{a_j} and $t_{a_j}^*$. For the query phase, given a value $b > 0$, we search for the largest index j, such that $\delta_{a_j} \leq b$, and report the value of $t_{a_j}^*$. We will prove later that $t_{a_j}^*$ approximates the stretch factor of the subgraph of the complete graph on S containing all edges of length at most b. In the rest of this section, we will formalize this approach.

As mentioned above, we fix a constant $c > 1$. For any integer $j \geq 0$, we define

$$X_j := \{i : k \leq i \leq m \text{ and } c^j \leq t_i^* < c^{j+1}\}.$$

Since all stretch factors t_i^* are greater than or equal to one, these sets X_j partition the set $\{k, k+1, \ldots, m\}$. Also, if $X_j \neq \emptyset$, then there is an index i such that $c^j \leq t_i^*$. Since $t_i^* \leq t_k^*$ and, by Lemma 2, $t_k^* \leq n - 1$, we have $c^j \leq n - 1$, which implies that $j \leq \lfloor \log_c(n-1) \rfloor$.

Let ℓ be the number of non-empty sets X_j. Then $\ell \leq 1 + \lfloor \log_c(n-1) \rfloor$. Each non-empty set X_j is a set of consecutive integers. We denote these non-empty sets by $I_1, I_2, \ldots, I_\ell$, and write them as $I_j = \{a_j, a_j+1, \ldots, a_{j+1}-1\}$, $1 \leq j \leq \ell$, where $k = a_1 < a_2 < \ldots < a_{\ell+1} = m+1$.

The following lemma states that any two stretch factors, whose indices are from the same set I_j, are within a factor of c of each other. The proof follows immediately from the definition of the sets I_j.

Lemma 3 *Let j be any integer such that $1 \leq j \leq \ell$, and let i and i' be any two elements of the set I_j. Then $1/c < t_i^*/t_{i'}^* < c$.*

Now we are ready to give the data structure for solving the approximate bottleneck stretch factor problem. This data structure consists of the following.

1. The connectivity threshold δ_k.
2. An array $\Delta[1 \ldots \ell]$, where $\Delta[j] = \delta_{a_j}$, $1 \leq j \leq \ell$.
3. An array $SF[1 \ldots \ell]$, where $SF[j] = t_{a_j}^*$, $1 \leq j \leq \ell$.

Note that the array Δ is sorted in increasing order, whereas the array SF is sorted in non-increasing order.

Recall that in a query, we get a real number $b > 0$, and have to compute an approximate stretch factor t of the graph containing all edges having length at most b. Such a query is answered as follows.

1. If $b < \delta_k$, then the subgraph of the complete graph on S containing all edges of length at most b is not connected. Hence, we report $t := \infty$.
2. If $b \geq \delta_k$, then we search in Δ for the largest index j such that $\Delta[j] \leq b$, and report the value of t defined as $t := SF[j]$.

The following lemma proves the correctness of this query algorithm.

Lemma 4 *Assume that $b \geq \delta_k$. Let t^* be the stretch factor of the subgraph of the complete graph on S containing all edges of length at most b. The value of t reported by the query algorithm satisfies $t/c < t^* < ct$.*

Proof. Consider the index j that was found by the query algorithm. Hence, $t = SF[j] = t_{a_j}^*$. Note that $a_j \in I_j$. Let i be the largest index such that $\delta_i \leq b$. Then $t^* = t_i^*$, and i is also an element of I_j. The claim now follows from Lemma 3. ∎

Let us analyze the complexity of our solution. We need $O(\ell) = O(\log n)$ space to store the data structure. If we implement the query algorithm using binary search, then the query time is bounded by $O(\log \ell) = O(\log \log n)$.

It remains to describe and analyze the preprocessing algorithm. First, we compute the sorted sequence of $m \leq \binom{n}{2}$ distances. This takes $O(n^2 \log n)$ time. Then we compute a minimum spanning tree of S. The length of a longest edge in this tree gives us the distance δ_k, and its index k. (See Lemma 1.) This step also takes $O(n^2 \log n)$ time. (Note that a minimum spanning tree of a set of n points in $\mathbb{R}^d$ can be computed faster. The $O(n^2 \log n)$–time bound, however, is good enough for the moment.) Now consider the sequence

$$1 = t_m^* \leq t_{m-1}^* \leq \ldots \leq t_{k+1}^* \leq t_k^* \leq n-1 \tag{1}$$

of stretch factors. The index sets $I_1, I_2, \ldots, I_\ell$ are obtained by locating the real numbers c^j, $0 \leq j \leq \lfloor \log_c(n-1) \rfloor$, in the sequence (1). Let $T_{SF}(n)$ denote the worst-case time to compute the exact stretch factor of any Euclidean graph on n points. Then, using binary search, we locate c^j in the sequence (1) in time

$$O\left(T_{SF}(n) \log(m-k+1)\right) = O\left(T_{SF}(n) \log n\right).$$

Hence, we can compute all index sets I_j, $1 \leq j \leq \ell$, in $O(T_{SF}(n) \log^2 n)$ total time. Given these index sets, we can compute the two arrays Δ and SF, in $O(T_{SF}(n) \log n)$ time. If we write the constant c as $1+\epsilon$, then we have proved the following result.

Theorem 2 *Let S be a set of n points in $\mathbb{R}^d$, and let $\epsilon > 0$ be a constant. For the $(1+\epsilon, 1+\epsilon)$-approximate bottleneck stretch factor problem, there is a data structure*

1. *that can be built in $O\left(n^2 \log n + T_{SF}(n) \log^2 n\right)$ time,*
2. *that has size $O(\log n)$, and*
3. *whose query time is bounded by $O(\log \log n)$.*

Note that even though the data structure has size $O(\log n)$, the algorithm given above uses $\Theta(n^2)$ space to compute it.

As mentioned in Section 1, the time complexity for computing the stretch factor of an arbitrary Euclidean graph is bounded by $O(n^3)$. Even though it may be possible to improve this upper bound, it is probably very hard to get a subquadratic time bound. Therefore, in the next section, we show that the preprocessing time can be reduced, at the cost of an increase in the approximation factor. The main idea is to store *approximate* stretch factors in the array SF.

4 An improved solution

Here we exploit the fact that approximate stretch factors can be computed more efficiently than exact stretch factors. In the previous section, we fixed a constant $c > 1$, and partitioned the sequence

$$t_m^* \leq t_{m-1}^* \leq \ldots \leq t_{k+1}^* \leq t_k^* \tag{2}$$

of exact stretch factors into $O(\log n)$ subsets, such that any two stretch factors in the same subset are within a factor of c of each other. We obtained this partition, by locating the values c^j, $0 \leq j \leq \lfloor \log_c(n-1) \rfloor$, in the sorted sequence (2).

In this section, we fix two additional constants c_1 and c_2 that are both greater than or equal to one. For any i, $k \leq i \leq m$, let t_i be a (c_1, c_2)-approximate stretch factor of the bottleneck graph G_i. Hence, we have $t_i/c_1 \leq t_i^* \leq c_2 t_i$. We will show how to use the sequence $t_m, t_{m-1}, \ldots, t_k$ of approximate stretch factors to partition the index set $\{k, k+1, \ldots, m\}$ into $O(\log n)$ subsets, such that for any two indices i and i' within the same subset, the exact stretch factors t_i^* and $t_{i'}^*$ are approximately equal. (The approximation factor depends on c, c_1, and c_2.) This partition is obtained by locating the values c^j in the sequence $t_m, t_{m-1}, \ldots, t_k$. Here, we have to be careful, because the values t_i are not sorted. They are, however, "approximately" sorted, and we will see that this suffices for our purpose.

Let $x > 0$ be a real number. We want to use binary search to "approximately" locate x in the "approximately" sorted sequence $t_m, t_{m-1}, \ldots, t_k$. We specify this algorithm by its decision tree[1].

[1]Note that this decision tree is not constructed (its size is quadratic in n), it is just a convenient way to describe the algorithm. The decision tree represents all possible computations of the algorithm on any input x.

This tree is a balanced binary tree that enables us to search in a sequence of numbers that have indices $k, k+1, \ldots, m$. More precisely,

1. the leaves of the tree store the indices $k, k+1, \ldots, m$, in this order, from left to right, and
2. each internal node u of the tree stores the smallest index that is contained in the right subtree of u.

Given the real number $x > 0$, we search as follows:

Algorithm *search*(x)
  $u :=$ root of the decision tree;
  while $u \neq$ leaf **do**
    $j :=$ index stored in u;
    if $x \leq t_j$ **then** $u :=$ right child of u **else** $u :=$ left child of u **endif**
  endwhile;
  return the index stored in u

Lemma 5 *Let $x > 0$ be a real number, and let z be the index that is returned by algorithm search(x).*

1. *For each i, $k \leq i < z$, we have $t_i^* \geq x/c_1$.*
2. *For each i, $z < i \leq m$, we have $t_i^* < c_2 x$.*

Hence, running algorithm *search*(x) implicitly partitions the sequence $t_k^*, t_{k+1}^*, \ldots, t_m^*$ of exact stretch factors into the following three subsequences:

1. $t_k^*, t_{k+1}^*, \ldots, t_{z-1}^*$; these are all greater than or equal to x/c_1.
2. t_z^*.
3. $t_{z+1}^*, t_{z+2}^*, \ldots, t_m^*$; these are all less than $c_2 x$.

We are now ready to give the algorithm that partitions the sequence $t_k^*, t_{k+1}^*, \ldots, t_m^*$ of exact stretch factors into $O(\log n)$ subsets, such that any two stretch factors in the same subset are approximately equal. First, we run algorithm *search*(c). Let z be the index returned. Then we report the two sets $\{z\}$ and $\{z+1, z+2, \ldots, m\}$ of indices. Next, we run algorithm *search*(c^2) on the index set $\{k, k+1, \ldots, z-1\}$. This results in a partition of the latter set into three subsets. The "last" two subsets are reported, whereas the "first" subset is partitioned further by running algorithm *search*(c^3). After $O(\log n)$ iterations, we obtain the partition we are looking for.

Let ℓ be the number of non-empty index sets that are computed by this algorithm. As in Section 3, we denote these by $I_1, I_2, \ldots, I_\ell$, and write them as $I_j = \{a_j, a_j + 1, \ldots, a_{j+1} - 1\}$, $1 \leq j \leq \ell$, where $k = a_1 < a_2 < \ldots < a_{\ell+1} = m + 1$. It is easy to see that $\ell = O(\log n)$. The following lemma states that we have obtained the desired partition.

Lemma 6 *Let y be any integer such that $1 \leq y \leq \ell$, and let i and i' be any two elements of the set I_y. Then $1/(cc_1c_2) < t_i^*/t_{i'}^* < cc_1c_2$.*

The data structure for solving the approximate bottleneck stretch factor problem consists of the following:

1. The connectivity threshold δ_k.

2. An array $\Delta[1 \dots \ell]$, where $\Delta[j] = \delta_{a_j}$, $1 \leq j \leq \ell$.

3. An array $SF_{approx}[1 \dots \ell]$, where $SF_{approx}[j] = t_{a_j}$, $1 \leq j \leq \ell$.

The query algorithm is basically the same as before. Given any real number $b > 0$, we do the following.

1. If $b < \delta_k$, then the subgraph of the complete graph on S containing all edges of length at most b is not connected. Hence, we report $t := \infty$.

2. If $b \geq \delta_k$, then we search in Δ for the largest index j such that $\Delta[j] \leq b$, and report the value of t defined as $t := SF_{approx}[j]$.

The following lemma gives the approximation ratio of this query algorithm.

Lemma 7 *Assume that $b \geq \delta_k$. Let t^* be the exact stretch factor of the subgraph of the complete graph on S containing all edges of length at most b. The value of t reported by the query algorithm satisfies*

$$\frac{1}{cc_1^2 c_2} t < t^* < cc_1 c_2^2 t.$$

Proof. Let j be the largest index such that $\Delta[j] \leq b$. Then $t = SF_{approx}[j] = t_{a_j}$. Let i be the largest index such that $\delta_i \leq b$. Then $t^* = t_i^*$. Since i and a_j both belong to the index set I_j, Lemma 6 implies that

$$\frac{1}{cc_1 c_2} < \frac{t^*}{t_{a_j}^*} < cc_1 c_2.$$

The lemma now follows from the fact that $1/c_1 \leq t_{a_j}^* / t_{a_j} \leq c_2$. ∎

It is clear that the data structure has size $O(\log n)$, and that the query time is bounded by $O(\log \log n)$. In the rest of this section, we analyze the time that is needed to construct the data structure. We will use the following notation.

- $T_{MST}(n)$: the time needed to compute a minimum spanning tree of a set of n points in $\mathbb{R}^d$.

- $T_{rank}(n)$: the time needed to compute the rank of any positive real number δ in the set of distances in a set of n points in $\mathbb{R}^d$. (The *rank* of δ is the number of distances that are less than or equal to δ.)

- $T_{approx\,SF}(n)$: the time needed to compute a (c_1, c_2)-approximate stretch factor of any bottleneck graph on a set of n points in $\mathbb{R}^d$.

- $T_{sel}(n)$: the time needed to compute the i-th smallest distance in a set of n points in $\mathbb{R}^d$, for any i, $1 \leq i \leq \binom{n}{2}$.

The preprocessing algorithm starts by computing a minimum spanning tree of the point set S. Let δ be the length of a longest edge in this tree. Note that the rank of δ is equal to k. Hence, we can compute the distance $\delta_k = \delta$, and the corresponding index k, in $O(T_{MST}(n) + T_{rank}(n))$ time. Given k and δ_k, we can compute the partition of $\{k, k+1, \dots, m\}$ into non-empty index sets I_j, in $O(T_{approx\,SF}(n) \log^2 n)$ time. Given this partition, we can compute the array $SF_{approx}[1 \dots \ell]$ in $O(T_{approx\,SF}(n) \log n)$ time. To compute the array $\Delta[1 \dots \ell]$, we have to solve $O(\log n)$ selection queries of the form "given an index j, compute the a_j-th smallest distance δ_{a_j} in the point set S".

One such query takes $T_{sel}(n)$ time. Hence, we can compute the entire array Δ in $O(T_{sel}(n)\log n)$ time.

We observe that $T_{rank}(n) = O(T_{sel}(n)\log n)$: We can compute the rank of a positive real number δ, by performing a binary search in the index set $\{1, 2, \ldots, \binom{n}{2}\}$. During this search, comparisons are resolved in $T_{sel}(n)$ time.

If we write the constant c as $1+\epsilon$, then we obtain the following result.

Theorem 3 *Let S be a set of n points in $\mathbb{R}^d$, and let $\epsilon > 0$ be a constant. For the $((1+\epsilon)c_1^2c_2, (1+\epsilon)c_1c_2^2)$-approximate bottleneck stretch factor problem, there is a data structure*

1. *that can be built in $O(T_{MST}(n) + T_{approx\,SF}(n)\log^2 n + T_{sel}(n)\log n)$ time,*
2. *that has size $O(\log n)$, and*
3. *whose query time is bounded by $O(\log\log n)$.*

5 A fast implementation of the improved algorithm

In order to apply Theorem 3, we need good upper bounds on the functions $T_{MST}(n)$, $T_{sel}(n)$, and $T_{approx\,SF}(n)$. For the first two functions, subquadratic bounds are known. Theorem 1 implies an upper bound on $T_{approx\,SF}(n)$: We run the algorithm of [13] on the bottleneck graph. Since such a graph can have a quadratic number of edges, however, this gives a bound that is at least quadratic in n. In Section 5.2, we will show that the bottleneck graph G_i can be approximated by a graph H_i having fewer edges. That is, H_i has $O(n\log n)$ edges, and its stretch factor is approximately equal to that of G_i. This will allow us to approximate the stretch factor of G_i in subquadratic time.

The computation of the graph H_i is based on the *well-separated pair decomposition*, devised by Callahan and Kosaraju [7].

5.1 The well-separated pair decomposition

In this section, we briefly review well-separated pairs and some of their relevant properties.

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 w.r.t. *s, if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that (i) C_A contains all points of A, (ii) C_B contains all points of B, and (iii) the distance between C_A and C_B is at least equal to s times the radius of C_A.*

We will assume that s is a constant, called the *separation constant*. The following lemma follows easily from Definition 1.

Lemma 8 *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) $|xy| \leq (1+4/s)\cdot|pq|$, and (ii) $|px| \leq (2/s)\cdot|pq|$.*

Definition 2 ([7]) *Let S be a set of n points in $\mathbb{R}^d$, and $s > 0$ a real number. A* well-separated pair decomposition *(WSPD) for S (w.r.t. s) is a sequence of pairs of non-empty subsets of S,*

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_\ell, B_\ell\},$$

such that

1. *$A_i \cap B_i = \emptyset$, for all $i = 1, 2, \ldots, \ell$,*

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) $p \in B_i$ and $q \in A_i$,*

3. *A_i and B_i are well-separated w.r.t. s, for all $i = 1, 2, \ldots, \ell$.*

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

In [5], Callahan shows that a WSPD of size $\ell = O(n \log n)$ can be computed, such that each pair $\{A_i, B_i\}$ contains at least one singleton set. This WSPD is computed using a binary tree T, called the *split tree*. We briefly describe the main idea. The split tree is similar to a kd-tree. Callahan starts by computing the bounding box of the points of S, which is successively split by d-dimensional hyperplanes, each of which is orthogonal to one of the axes. If a box is split, he takes care that each of the two resulting boxes contains at least one point of S. As soon as a box contains exactly one point, the process stops (for this box).

The resulting binary tree T stores the points of S at its leaves; one leaf per point. Also, each node u of T is associated with a subset of S. We denote this subset by S_u; it is the set of all points of S that are stored in the subtree of u.

The split tree T can be computed in $O(n \log n)$ time. Callahan shows that, given T, a WSPD of size $\ell = O(n \log n)$ can be computed in $O(n \log n)$ 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 $B_i = S_{v_i}$. Since at least one of A_i and B_i is a singleton set, at least one of u_i and v_i is a leaf of T.

Theorem 4 ([5]) *Let S be a set of n points in $\mathbb{R}^d$, and $s > 0$ a separation constant. In $O(n \log n)$ time, we can compute a WSPD for S of size $O(n \log n)$ such that each pair $\{A_i, B_i\}$ contains at least one singleton set.*

5.2 Approximating the bottleneck graph

Let $b > 0$ be a fixed real number, and let G be the Euclidean graph on the point set S containing all edges of length at most b. In this section, we show that we can use well-separated pairs to define a graph H whose stretch factor approximates that of G. In Section 5.3, we will give an algorithm that computes such a graph H having only $O(n \log n)$ edges.

Let $s > 4$ be a separation constant, and consider an arbitrary well-separated pair decomposition $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_\ell, B_\ell\}$ for the point set S. For any index i, $1 \leq i \leq \ell$, let $x_i \in A_i$ and $y_i \in B_i$ be two points for which $|x_i y_i|$ is minimum.

The graph H has the points of S as its vertices, and contains all edges (x_i, y_i) whose length is less than or equal to b.

We first show that any two points p and q of S that have Euclidean distance at most b are connected in H by a path whose length is at most a constant times $|pq|$.

Lemma 9 *Let p and q be any two points of S such that $|pq| \leq b$. Then*

$$|pq|_H \leq \frac{s+4}{s-4} \cdot |pq|.$$

Proof. The proof is basically the same as Callahan and Kosaraju's proof in [6] that the WSPD yields a spanner for S. ∎

The next lemma states that the stretch factors of G and H are approximately equal.

Lemma 10 *Let t_G^* and t_H^* denote the exact stretch factors of the graphs G and H, respectively. We have*

$$\frac{s-4}{s+4} \cdot t_H^* \leq t_G^* \leq t_H^*.$$

5.3 Computing the approximation graph H

We saw in the previous subsection that the graph H approximates the bottleneck graph G. In this section, we show how this graph H can be computed if we use an appropriate WSPD. Consider a WSPD $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_\ell, B_\ell\}$ in which each pair $\{A_i, B_i\}$ contains at least one singleton set. By Theorem 4, such a WSPD of size $\ell = O(n \log n)$ can be computed in $O(n \log n)$ time.

The main problem is that we have to compute for each pair $\{A_i, B_i\}$ in this WSPD, the points $x_i \in A_i$ and $y_i \in B_i$ for which $|x_i y_i|$ is minimum. Hence, if A_i is a singleton set, i.e., $A_i = \{x_i\}$, then we have to compute a nearest-neighbor y_i of x_i in the set B_i. We will show that by traversing the split tree T that gives rise to this WSPD, all these pairs (x_i, y_i), $1 \leq i \leq \ell$, can be computed efficiently.

Recall that for any node u of the split tree T, we denote by S_u the subset of S that is stored in the subtree of u. Also, each pair $\{A_i, B_i\}$ in the WSPD is defined by two nodes u_i and v_i of T. That is, $A_i = S_{u_i}$ and $B_i = S_{v_i}$.

We store with each node u of T, a list of all leaves v such that the two nodes u and v define a pair in the WSPD. (Hence, v defines a singleton set in this pair.)

Let DS be a data structure that stores a set of points in $\mathbb{R}^d$, that supports nearest-neighbor queries of the form "given a query point $q \in \mathbb{R}^d$, find a point in the set that is nearest to q", and that supports insertions of points.

The algorithm that computes the required closest pair of vertices in each well-separated pair of point sets, traverses the nodes of T in postorder. To be more precise, let u be an internal node of T, and let u' and u'' be the two children of u. At the moment when node u is visited, the nodes u' and u'' store nearest-neighbor data structures $DS(u')$ and $DS(u'')$ storing the point sets $S_{u'}$ and $S_{u''}$, respectively. If $|S_{u'}| \leq |S_{u''}|$, then we insert all points of $S_{u'}$ into $DS(u'')$. Otherwise, all points of $S_{u''}$ are inserted into $DS(u')$. Hence, after these insertions, we have a nearest-neighbor data structure $DS(u)$ storing the point set S_u. For each leaf v of T such that u and v define a pair in the WSPD, we query $DS(u)$ to find a point of S_u that is nearest to the point stored at leaf v.

During this postorder traversal of the split tree T, we get all pairs (x_i, y_i), $1 \leq i \leq \ell$. Clearly, the approximation graph H can be computed from these pairs, in time $O(\ell) = O(n \log n)$.

We analyze the running time of this algorithm. The number of nearest-neighbor queries is equal to the number ℓ of pairs in the WSPD. For any internal node u of T, the data structure $DS(u)$ is obtained by inserting the points from the child's structure whose subtree is smaller, into the structure of the other child of u. It is easy to prove that in this way, each point of S is inserted at most $\log n$ times. The total number of insertions is therefore bounded by $O(n \log n)$.

Let $Q_{NN}(n_0)$ and $I_{NN}(n_0)$ denote the query and insertion times of the data structure DS, respectively, if it stores a set of n_0 points. Since $n_0 \leq n$ at any moment during the algorithm, we have proved the following result.

Lemma 11 *Let S be a set of n points in $\mathbb{R}^d$. After $O(n(Q_{NN}(n) + I_{NN}(n)) \log n)$ preprocessing time, we can compute the approximation graph H of any bottleneck graph G, in $O(n \log n)$ time.*

In order to apply Lemma 11, we need to specify the data structure DS. This data structure stores a set of points in $\mathbb{R}^d$, and supports nearest-neighbor queries and insertions of points. We can obtain such a semi-dynamic data structure by applying Bentley's logarithmic method, see [3, 4]. This technique transforms an arbitrary static data structure for nearest-neighbor queries into one that also supports insertions of points. To be more specific, let DS^s be a static data structure storing a set of n points in $\mathbb{R}^d$, that supports nearest-neighbor queries in $Q^s_{NN}(n)$ time, and that can be built in $P^s_{NN}(n)$ time. The logarithmic method transforms DS^s into a semi-dynamic structure DS, in which nearest-neighbor queries can be answered in $O(Q^s_{NN}(n) \log n)$ time, and in which points can be inserted in $O((P^s_{NN}(n)/n) \log n)$ amortized time.

Corollary 1 *Let S be a set of n points in $\mathbb{R}^d$, let $\beta \geq 1$ be an integer constant, and let ϵ be a real constant, such that $0 < \epsilon \leq 1/2$. After $O(n\, Q^s_{NN}(n) \log^2 n + P^s_{NN}(n) \log^2 n)$ preprocessing time, we can compute a (c_1, c_2)-approximate stretch factor, where $c_1 = 2\beta(1+\epsilon)^2$ and $c_2 = 1+\epsilon$, of any bottleneck graph in $O(n^{1+1/\beta} \log^3 n)$ expected time.*

Proof. Apply Lemma 11 with separation constant $s := 4 + 8/\epsilon$. The claim follows from Theorem 1 and Lemmas 10 and 11. ∎

If we combine Theorem 3 and Corollary 1, then we obtain the main result of this paper.

Theorem 5 *Let S be a set of n points in $\mathbb{R}^d$, let $\beta \geq 1$ be an integer constant, and let ϵ be a real constant, such that $0 < \epsilon \leq 1/2$. In*

$$O\left(n\, Q^s_{NN}(n) \log^2 n + P^s_{NN}(n) \log^2 n + n^{1+1/\beta} \log^5 n + T_{sel}(n) \log n\right)$$

expected time, we can compute a data structure of size $O(\log n)$, such that for any real number $b > 0$, we can compute, in $O(\log\log n)$ time, a real number t, such that

$$\frac{1}{4\beta^2(1+\epsilon)^6}\, t \leq t^* \leq 2\beta(1+\epsilon)^5 t,$$

where t^* is the exact stretch factor of the Euclidean graph containing all edges of length at most b.

We conclude this section by giving concrete bounds on the preprocessing time. We start with the case when the dimension d is equal to two. The static nearest-neighbor problem can be solved using Voronoi diagrams, and a data structure for point location queries, see Preparata and Shamos [14]. For this data structure, we have $Q^s_{NN}(n) = O(\log n)$, and $P^s_{NN}(n) = O(n \log n)$. Chan [8] gives a randomized distance selection algorithm, whose expected running time $T_{sel}(n)$ is bounded by $O(n^{4/3} \log^{5/3} n)$. Hence, if $d = 2$, the expected time needed to build the data structure of Theorem 5 is bounded by $O(n^{1+1/\beta} \log^5 n + n^{4/3} \log^{8/3} n)$. If $\beta = 2$, then the expected preprocessing time is roughly $n^{3/2}$. For $\beta = 3$, it is roughly $n^{4/3}$. For larger values of β, the time bound remains roughly $n^{4/3}$, but then the approximation ratio increases.

Assume that $d \geq 3$. Agarwal, in a personal communication to Dickerson and Eppstein [10], has shown that

$$T_{sel}(n) = O(n^{2(1-1/(d+1))+\eta}), \tag{3}$$

where η is an arbitrarily small positive real constant. (The constant in the Big-Oh bound depends on η.) Agarwal and Matoušek [1], and Matoušek and Schwarzkopf [12] have given a static nearest-neighbor data structure for which $n\, Q^s_{NN}(n) \log^2 n + P^s_{NN}(n) \log^2 n$ is asymptotically smaller than the quantity on the right-hand side of (3). Hence, the expected time needed to build the data structure of Theorem 5 is bounded from above by $O(n^{1+1/\beta} \log^5 n + n^{2(1-1/(d+1))+\eta})$. This becomes $O(n^{2(1-1/(d+1))+\eta})$, i.e., subquadratic, if we take $\beta = 2$. Again, for larger values of β, we get the same time bound, but a larger approximation ratio.

6 Concluding remarks

We have given a subquadratic algorithm for preprocessing a set S of n points in $\mathbb{R}^d$ into a data structure of size $O(\log n)$ such that for an arbitrary query value $b > 0$, we can, in $O(\log\log n)$ time, compute an approximate stretch factor of the bottleneck graph on S containing all edges of length at most b. This result was obtained by (i) approximating the sequence of $\binom{n}{2}$ different stretch factors of all possible bottleneck graphs, and (ii) approximating bottleneck graphs by graphs containing only $O(n \log n)$ edges.

Our algorithms need exact solutions for computing minimum spanning trees, and solving nearest-neighbor queries, distance selection queries, and distance ranking queries. It would be interesting to know if approximation algorithms for these problems can be used to speed up the preprocessing time.

References

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22:794–806, 1993.

[2] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 489–498, 1995.

[3] J. L. Bentley. Decomposable searching problems. *Inform. Process. Lett.*, 8:244–251, 1979.

[4] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[5] P. B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications.* Ph.D. thesis, Dept. Comput. Sci., Johns Hopkins University, Baltimore, Maryland, 1995.

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

[7] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[8] T. M. Chan. On enumerating and selecting distances. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 279–286, 1998.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[10] M. T. Dickerson and D. Eppstein. Algorithms for proximity problems in higher dimensions. *Comput. Geom. Theory Appl.*, 5:277–291, 1996.

[11] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science, Amsterdam, 1999.

[12] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10:215–232, 1993.

[13] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM J. Comput.*, to appear.

[14] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[15] 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, Amsterdam, 1999.

A Appendix

Proof of Lemma 1: Let p and q be two points of S such that (p, q) is a longest edge in T, and let $\delta := |pq|$. We have to show that $\delta = \delta_k$.

If we remove the edge (p, q) from T, then we get two disjoint trees T_1 and T_2. Let A and B be the vertex sets of T_1 and T_2, respectively. Then A and B form a partition of the set S, and p and q are in different subsets. We may assume w.l.o.g. that $p \in A$ and $q \in B$. Since T is a minimum spanning tree of S, it is clear that

$$\delta = |pq| = \min\{|ab| : a \in A, b \in B\}.$$

Hence we have $|ab| \geq \delta$ for all $a \in A$ and $b \in B$. Let i be the index such that $\delta = \delta_i$. Then it follows that none of the graphs $G_0, G_1, \ldots, G_{i-1}$ is connected. On the other hand, each edge of T has length at most $\delta = \delta_i$. Hence, T is contained in G_i, which implies that G_i is connected. This shows that $i = k$ and, hence, $\delta = \delta_k$. ∎

Proof of Lemma 2: First note that the claim holds if $k = m$. Therefore, we may assume that $k < m$. Consider the graph G_k, and let p and q be any two distinct points of S that are not connected by an edge in G_k. Then $|pq| > \delta_k$. We will show that the distance $|pq|_{G_k}$ in G_k between p and q is less than $(n-1)|pq|$.

Let T be a minimum spanning tree of S. Then, by Lemma 1, T is contained in G_k. Let P be the path in T between p and q. This path consists of at most $n-1$ edges, each having length less than or equal to δ_k. Hence, the length $|P|$ of P is bounded from above by

$$|P| \leq (n-1)\delta_k < (n-1)|pq|.$$

It follows that

$$|pq|_{G_k} \leq |P| < (n-1)|pq|. \tag{4}$$

Clearly, (4) also holds if the points p and q are connected by an edge in G_k. Therefore, we have shown that $t^*_k < n-1$. ∎

Proof of Lemma 5: We will only prove the first claim. The second claim can be proved in a symmetric way.

Let i be any index such that $k \leq i < z$. Let u be the lowest common ancestor of the leaves that store the indices i and z. Note that i is stored in the left subtree of u, whereas z is stored in the right subtree of u.

Let j be the index that is stored in node u. Algorithm *search*(x) visits node u and proceeds to the right child of u. Hence, it follows from the algorithm that $x \leq t_j$. This implies that

$$t^*_j \geq \frac{1}{c_1} t_j \geq \frac{1}{c_1} x.$$

Since $i < j$, we have $t^*_i \geq t^*_j$, from which the first claim follows. ∎

Proof of Lemma 6: If the set I_y contains only one element, then $i = i'$, and the claim clearly holds. Assume that $|I_y| \geq 2$.

Let j be the integer such that the index set I_y is reported in the j-th iteration of the partition algorithm. First assume that $j \geq 2$. It follows from Lemma 5 that $t^*_i < c_2 c^j$ and $t^*_{i'} < c_2 c^j$. In the $(j-1)$-st iteration, both indices i and i' were in the "first" subset. Hence, Lemma 5 implies that

$t_i^* \geq c^{j-1}/c_1$ and $t_{i'}^* \geq c^{j-1}/c_1$. The claim follows from these four inequalities. The proof for the case when $j = 1$ is similar. ∎

Proof of Lemma 9: The proof is basically the same as Callahan and Kosaraju's proof in [6] that the WSPD yields a spanner for S. We include the proof here in order to be self-contained. Let $t := (s+4)/(s-4)$.

The proof is by induction on the rank of the distance $|pq|$ in the sorted sequence of distances in S that are at most b. To start the induction, the claim clearly holds if $p = q$. So assume that $p \neq q$. Moreover, assume that for any two points x and y of S with $|xy| < |pq|$, the inequality $|xy|_H \leq t \cdot |xy|$ holds.

Let i be the index such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $p \in B_i$ and $q \in A_i$. We may assume w.l.o.g. that (i) holds. Our choice of the points x_i and y_i implies that $|x_i y_i| \leq |pq| \leq b$. Hence, (x_i, y_i) is an edge in the graph H, i.e., $|x_i y_i|_H = |x_i y_i|$.

By Lemma 8, we have $|px_i| \leq (2/s) \cdot |pq| < |pq|$. Therefore, by the induction hypothesis, we have $|px_i|_H \leq t \cdot |px_i|$. By a symmetric argument, we have $|y_i q|_H \leq t \cdot |y_i q|$. Finally, by Lemma 8, we have $|x_i y_i| \leq (1 + 4/s) \cdot |pq|$. It follows that

$$\begin{aligned} |pq|_H &\leq |px_i|_H + |x_i y_i|_H + |y_i q|_H \\ &\leq t \cdot |px_i| + |x_i y_i| + t \cdot |y_i q| \\ &\leq t \cdot (2/s) \cdot |pq| + (1 + 4/s) \cdot |pq| + t \cdot (2/s) \cdot |pq| \\ &= t \cdot |pq|, \end{aligned}$$

completing the proof. ∎

Proof of Lemma 10: We have to show that for any two points p and q in S, the two inequalities

$$|pq|_G \leq |pq|_H \leq \frac{s+4}{s-4} \cdot |pq|_G$$

hold. Since H is a subgraph of G, it is clear that $|pq|_G \leq |pq|_H$. The second inequality clearly holds if $|pq|_G = \infty$. So assume that $|pq|_G < \infty$. Let $p = p_0, p_1, \ldots, p_j = q$ be a shortest path in G between p and q. For each i, $0 \leq i < j$, we have $|p_i p_{i+1}| \leq b$. Hence, by Lemma 9, we have $|p_i p_{i+1}|_H \leq (s+4)/(s-4) \cdot |p_i p_{i+1}|$. It follows that

$$|pq|_H \leq \sum_{i=0}^{j-1} |p_i p_{i+1}|_H \leq \frac{s+4}{s-4} \cdot \sum_{i=0}^{j-1} |p_i p_{i+1}| = \frac{s+4}{s-4} \cdot |pq|_G.$$

∎

Proof of Corollary 1: We apply Lemma 11 with separation constant $s := 4 + 8/\epsilon$. The bound on the preprocessing time follows from Lemma 11. Consider any bottleneck graph G, and let H be the corresponding approximation graph. After the preprocessing, we can compute H in $O(n \log n)$ time. Let t_G^* and t_H^* be the exact stretch factors of G and H, respectively. Applying Lemma 10, and noting that $(s+4)/(s-4) = 1 + \epsilon$, we have

$$\frac{1}{1+\epsilon} \cdot t_H^* \leq t_G^* \leq t_H^*. \tag{5}$$

Since the number of edges of H is bounded by $O(n \log n)$, Theorem 1 implies that we can compute, in $O(n^{1+1/\beta} \log^3 n)$ expected time, a real number t, such that

$$\frac{1}{2\beta(1+\epsilon)} \cdot t \leq t_H^* \leq (1+\epsilon)t. \tag{6}$$

Combining (5) and (6), it follows that

$$\frac{1}{2\beta(1+\epsilon)^2} \cdot t \leq t^*_G \leq (1+\epsilon)t.$$

■

Proof of Theorem 5: The claims on the size of the data structure, the query time, and the approximation ratio follow from Theorem 3 and Corollary 1. By Theorem 3, the time needed to build the data structure is bounded by

$$O\left(T_{MST}(n) + T_{approx\,SF}(n)\log^2 n + T_{sel}(n)\log n\right).$$

By Corollary 1, we have

$$T_{approx\,SF}(n)\log^2 n = O\left(n\, Q^s_{NN}(n)\log^2 n + P^s_{NN}(n)\log^2 n + n^{1+1/\beta}\log^5 n\right). \tag{7}$$

Consider the closest pairs (x_i, y_i), $1 \leq i \leq \ell = O(n \log n)$, that we computed in Section 5.3. Callahan [5] has shown that the graph containing these pairs as edges contains a minimum spanning tree of the point set. Hence, given these edges, we can use Prim's algorithm to compute a minimum spanning tree of S, in $O(n \log n)$ time. (See [9].) Therefore, $T_{MST}(n)$ is bounded from above by the expression in (7). ■