Approximating the Weight of the Euclidean Minimum Spanning Tree in Sublinear Time*

Artur Czumaj† Funda Ergün‡ Lance Fortnow § Avner Magen ¶ Ilan Newman‖
Ronitt Rubinfeld ** Christian Sohler††

Abstract

We consider the problem of computing the weight of a Euclidean minimum spanning tree for a set of n points in $\mathbb{R}^d$. We focus on the setting where the input point set is supported by certain basic (and commonly used) geometric data structures that can provide efficient access to the input in a structured way. We present an algorithm that estimates with high probability the weight of a Euclidean minimum spanning tree of a set of points to within $1+\varepsilon$ using only $\widetilde{\mathcal{O}}(\sqrt{n}\,\mathrm{poly}(1/\varepsilon))$ queries for constant d. The algorithm assumes that the input is supported by a minimal bounding cube enclosing it, by orthogonal range queries, and by cone approximate nearest neighbors queries.

1 Introduction

As the power and connectivity of computers increase and the cost of memory becomes cheaper, we have become inundated with large amounts of data. Although traditionally linear time algorithms were sought to solve our problems, it is no longer clear that a linear time algorithm is good enough in every setting. The question then is whether we can solve *anything* of interest in sublinear time, when the algorithm is not even given time to read all of the input data. The answer is yes; in recent years, several sublinear time algorithms have been presented which solve a wide range of property testing and approximation problems.

In this paper we consider the problem of estimating the weight of a minimum spanning tree, where the input is a set of points in the Euclidean space $\mathbb{R}^d$. Since the location of a single point may dramatically influence the value of the weight of the Euclidean minimum spanning tree (EMST), we cannot hope to get a reasonable approximation in sublinear time with only access to the locations of the

*A preliminary version of this paper appeared in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 813–822, Baltimore, MD, January 12–14, 2003. SIAM, Philadelphia, PA, 2003.

†Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA. Email: czumaj@cis.njit.edu. Research supported in part by NSF grant CCR-0105701.

‡School of Computer Science, Simon Fraser University, Vancouver, BC V5A 1S6, Canada. Email: funda@cs.sfu.ca. Part of this work was done while the author was at NEC Research, Princeton, NJ 08540, USA.

§Department of Computer Science, University of Chicago, Chicago, IL 60637, USA. Email: fortnow@cs.uchicago.edu. Part of this work was done while the author was at NEC Research, Princeton, NJ 08540, USA.

¶Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4, Canada. Email: avner@cs.toronto.edu. Part of this work was done while the author was at NEC Research, Princeton, NJ 08540, USA.

‖Department of Computer Science, University of Haifa, Haifa, Israel. Email: ilan@cs.haifa.ac.il. Part of this work was done while the author was at NEC Research, Princeton, NJ 08540, USA.

**MIT CSAIL, Cambridge, MA 02139, USA. Email: ronitt@theory.csail.mit.edu. Part of this work was done while the author was at NEC Research, Princeton, NJ 08540, USA.

††Heinz Nixdorf Institute and Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, D-33102 Paderborn, Germany. Email: csohler@uni-paderborn.de. Research partly supported by DFG grant Me 872/8-1 and EU grant IST-1999-14186 (ALCOM-FT).

points. This is true even when we consider probabilistic algorithms. However, it is often the case that massive databases, particularly in a geometric context, contain sophisticated data structures on top of the raw data, that support various forms of queries. Examples of such queries are the nearest neighbor of a point, or the point with the highest value in a coordinate. Consequently, in this paper, we assume that algorithms have access to certain commonly used data structures which aid the algorithm in its computation. This may be considered a motivation for maintaining such data structures, particularly if they aid in other tasks as well.

1.1 Results

In this paper we describe three algorithms for estimating the weight of a Euclidean minimum spanning tree over n given points in a Euclidean space $\mathbb{R}^d$, where the algorithms are given access to basic geometric data structures supporting the input. Throughout the paper we assume that d is a constant, though our analysis can be easily carried over for arbitrary values of d. It should be noted that our algorithms do not supply a low weight spanning tree (which takes linear space to represent), but only estimate its weight.

We first consider the case when the algorithm is given, in addition to access to the input point set, (1) a *minimal bounding cube* that contains all points in the input set and (2) access to an *orthogonal range query* data structure which, given an axis-parallel cube, answers whether there is an input point within the cube. In this model, we give a deterministic $\mathcal{O}(n^{1/2})$-time algorithm for the 2-dimensional case which outputs a value w such that $\frac{1}{\alpha}\,\textsc{emst}(P) - \mathcal{L}n^{-c} \le w \le \alpha\,\textsc{emst}(P) + \mathcal{L}n^{-c}$, where $\alpha = \Theta(n^{1/8}\,\log n)$, $\mathcal{L}$ is the side-length of a minimal axis parallel bounding cube of the point set, and c is an arbitrary constant. We also show that any deterministic algorithm that uses $o(n^{1/2})$ orthogonal range queries cannot significantly improve the quality of approximation.

We next consider the case where, in addition to the above data structures, we are also given (3) access to a *cone nearest neighbor* data structure, which given a point p and a cone C, returns a nearest point to p in the cone $p + C$. Our second algorithm combines the extra power of the cone nearest neighbor data structures with ideas from the recent randomized sublinear-time algorithm for estimating the MST in general graphs [10]. The algorithm outputs a value which with high probability is within a $1 + \varepsilon$ factor of the EMST and it runs in $\mathcal{O}(\Lambda/\varepsilon^3)$ time, where Λ is the *spread* of P (ratio between maximum and minimum distance between points in P); observe that Λ can be arbitrarily large.

Our main contribution is the third algorithm that does not have any dependency on Λ and requires only cone *approximate* nearest neighbor queries which we define in the next section. For a constant d, the algorithm runs in $\widetilde{\mathcal{O}}(\sqrt{n}\,\mathrm{poly}(1/\varepsilon))$ time and outputs an approximation of the EMST weight to within a multiplicative factor of $1+\varepsilon$ with high probability. The algorithm combines the ideas from our first two algorithms. It partitions the input points into components and estimates the EMST separately by considering pairs of points that lie in the same component and pairs of points that belong to different components. To estimate the EMST within components, we use an extension of our second algorithm. To estimate the weight required to connect the components we use a variant of our first algorithm. The combination of these two algorithms leads to a significant improvement in the quality of approximation (compared to the first algorithm) and in the running time (compared to the second algorithm).

We notice also that our algorithms lead to sublinear-time $(2 + \varepsilon)$-approximation algorithms for two other classical geometric problems: Euclidean TSP and the Euclidean Steiner tree problem. These results follow from the well known relationship between the weight of EMST and the weight of Euclidean TSP and of Euclidean Steiner tree (see, e.g., [24]). Indeed, it is known that in metric spaces the weight of Euclidean TSP is between the weight of the EMST and twice the EMST weight. Similarly, it is known that in metric spaces the EMST weight is between the weight of the Steiner tree and twice of its weight.

On the plane, one can improve this result by using the fact that the EMST weight is upper bounded by at most $2/\sqrt{3}$ times the weight of the Euclidean Steiner tree [13].

1.2 Relation to previous works

The Euclidean minimum spanning tree problem is a classical problem in computational geometry and has been extensively studied in the literature for more than two decades. It is easy to see that to find the EMST of n points, $\mathcal{O}(d\,n^2)$ time suffices, by reducing the problem to finding the MST in dense graphs. In the simplest case where $d = 2$ (on the plane), Shamos and Hoey [22] show that the EMST problem can be solved in $\mathcal{O}(n \log n)$ time. For $d \geq 3$, no $\widetilde{\mathcal{O}}(n)$-time algorithm is known and it is a major open question whether an $\mathcal{O}(n \log n)$-time algorithm exists even for $d = 3$ [16]; in fact, it is even conjectured (see, e.g., [16]) that no $o(n^{4/3})$-time algorithm does exist. Yao [26] was the first who broke the $\mathcal{O}(n^2)$-time barrier for $d \geq 3$ and designed an $\widetilde{\mathcal{O}}(n^{1.8})$-time algorithm for $d = 3$. This bound has been later improved and the fastest currently known (randomized) algorithm achieves the running time of $\widetilde{\mathcal{O}}(n^{4/3})$ [2] for $d = 3$ (and the running time tends to $\mathcal{O}(n^2)$ as d grows). Significantly better bounds can be achieved if one allows to approximate the output. Callahan and Kosaraju [7] give a $\mathcal{O}(n \log n + n \log(1/\varepsilon)\, \varepsilon^{-d/2})$-time algorithm that finds an approximate Euclidean minimum spanning tree to within a multiplicative factor of $1 + \varepsilon$.

Our algorithms rely on a recent randomized algorithm of [10] that, given a connected graph in adjacency list representation with average degree d, edge weights in the range $[1 \dots W]$, and a parameter $0 < \varepsilon < \frac{1}{2}$, approximates, with high probability, the weight of a minimum spanning tree in time $\widetilde{\mathcal{O}}(d\,W\,\varepsilon^{-3})$ within a factor of $1 + \varepsilon$. The time bound does not directly depend on the number of vertices or edges in the graph. We emphasize, however, that our representation is quite different, and in general would give a graph with average degree n. Therefore, a direct application of this result to the EMST problem does not lead to a sublinear-time algorithm.

We also notice that very recently, a similar approach of partitioning the input points on a grid as that used in our paper has been used in the problem of approximating the EMST in the streaming model in one pass by Indyk [18]. He showed that the EMST of n points from $\{1, \dots, \Delta\}^d$ is approximated to within an $O(d \log \Delta)$ factor using $O(d \log^2 \Delta)$ bits of storage for dynamic data streams through a hierarchically well-separated trees defined over a set of grids over the input points. In another recent work, Czumaj and Sohler [12] designed a randomized $(1 + \varepsilon)$-approximation algorithm for metric MST with the running time $\mathcal{O}(n\,\mathrm{polylog}(n)/\varepsilon^{\mathcal{O}(1)})$, where the input graph has n vertices and $\Theta(n^2)$ edges.

1.3 Dynamic algorithms

Our model of computation is also interesting in the context of dynamic algorithms. There exist fully dynamic algorithms that maintain the EMST subject to point insertions and deletions; [15] gives an algorithm with amortized time $\widetilde{\mathcal{O}}(\sqrt{n})$ and $\mathcal{O}(n^{1-\varepsilon})$ per update operation for $d \leq 4$ and $d > 4$ respectively. A disadvantage of this algorithm (and of all typical dynamic algorithms) is that it requires as much as $\widetilde{\mathcal{O}}(\sqrt{n})$ time per input update, making the algorithm very costly in situations where the EMST queries are very rare. The data structures we require in our setting are dynamically maintained by standard geometric databases anyway. Thus, if the database supports all required data structures in polylogarithmic time, the amortized time required by our algorithm is $\widetilde{\mathcal{O}}(\sqrt{n}/U)$, where U is the typical number of updates per one EMST calculation. We note again that our algorithm does not supply the minimum spanning tree, but returns only its approximate weight.

1.4 Discussion on the model of computations

Our results assume that the data is organized using certain types of commonly used data structures. This idea has been frequently used in other areas, like databases, where one assumes a selected set of operations on the top of the data set is provided to the user. A similar model of computations has been used recently in a related context of geometric computations in [11] (see also [23]).

The data structures used in our paper are commonly used in algorithms in general, and some (as in supporting approximate nearest neighbor queries) are used more in the context of computational geometry. Our algorithms cannot be expected to construct these data structures given its sublinear nature. However, neither can it be assumed to have collected the input set either. The assumption that some other source has collected the data is inherent in the model of sublinear algorithms, and in our paper we strengthen this assumption to require that during this preprocessing some well known and natural data structures have been built. In most of the cases, the construction time for these data structures is not much in excess of the time needed to read and store the entire data set. One can view this as having separate agents performing the data collection and computation. Alternately, one can think of the data collection and organization step as preprocessing, to be done only once. One would like to know the degree of the dependence on these underlying data structures and explore the tradeoff resulting from them. We believe that this work is an important step in understanding the nature of geometric problems and the time/resource tradeoffs (resources can include the underlying data structures) involved in their solutions. We would like to learn what inherent properties of a problem and the data set lead to better sublinear algorithms. In the future we expect to solve more problems in this area and develop an understanding of the inherent structure of geometric objects and their collections which allow different time/resource tradeoffs.

Organization of the paper. We start by presenting an algorithm that only needs access to a minimal bounding cube of the point set P and to an orthogonal range query oracle in Section 3. In Section 5, we present a simple algorithm that uses additionally the cone nearest neighbor oracle. Finally, in Section 6, we discuss the main contribution of this paper, a sublinear time algorithm that uses a minimal bounding cube oracle, the orthogonal range query oracle and the cone *approximate* nearest neighbor oracle.

2 Preliminaries

For a given set P of points in a Euclidean space $\mathbb{R}^d$, a *(Euclidean) graph* on P can be modeled as a weighted undirected graph $G = (P, E)$, where P is a vertex set, E is a subset of the (unordered) pairs of points in P, and the *length/weight* of edge $\{p, q\}$ is equal to the Euclidean distance between points p and q, denoted $|p - q|$. The *weight of the graph* is the sum of the weights of its edges.

Throughout the paper we denote by $\mathbb{K}_P$ the complete (undirected) graph on P where the edge weights are the Euclidean distances between the endpoints. A graph G on a set of points P is called a *Euclidean minimum spanning tree (*EMST*)* of P if it is a minimum-weight spanning subgraph of $\mathbb{K}_P$. We denote by EMST(P) both the EMST of P and the weight of the EMST of P. Similarly, for a given graph G we will denote by MST(G) the minimum spanning tree of G as well as the weight of the minimum spanning tree of G.

For a given point set P, we denote by Λ the *spread* of P, that is, the ratio between the maximum and the minimum distances between points in P. We let $\mathcal{BC}$ be a minimal bounding cube of P (which is made available via the *minimal bounding cube oracle*) and let $\mathcal{L}$ denote its side length.

2.1 Models of computation

In this paper we use some basic geometric data structures supporting access to the input point set. Given a point set P in $\mathbb{R}^d$, we use data structures supporting the following types of queries:

- **minimal bounding cube of P:** returns the location of a minimum size axis-parallel d-dimensional cube containing P, that is, returns the location of a cube $C = [a_1, a_1 + \mathcal{L}] \times [a_2, a_2 + \mathcal{L}] \times \ldots \times [a_d, a_d + \mathcal{L}]$ that contains P such that no axis-parallel cube of edge length smaller than $\mathcal{L}$ contains P.

- **(orthogonal) range query oracle:** for a given axis-parallel cube C, tests if C contains a point from P.

- **cone $(1+\delta)$-approximate nearest neighbor oracle:** δ is any non-negative real number and it is assumed that a set of cones $\mathcal{C}$ with apexes at the origin is given in advance. The cone $(1+\delta)$-approximate nearest neighbor oracle, for a given point $p \in P$ and a given cone $C \in \mathcal{C}$, returns a $(1+\delta)$-approximate nearest neighbor[1] of p in $(P \setminus \{p\}) \cap (p + C)$. (We denote by $p + C$ the translated cone $\{a + p : a \in C\}$.) If $(P \setminus \{p\}) \cap (p + C)$ is empty, then a special value is returned.

  In the special case where $\delta = 0$, the oracle gives the true nearest neighbor, and is simply called the **cone nearest neighbor oracle**.

2.1.1 Implementing supporting data structures

To make our model of computations viable, we discuss here how our supporting data structures (oracles) can be implemented efficiently using standard geometric data structures.

Minimal bounding cube. The query about the *minimal bounding cube* of a set of points $P \in \mathbb{R}^d$ can be supported by many standard geometric data structures. Indeed, the only information required to find the minimal bounding cube is to know the minimum and maximum d-dimensional coordinates of all input points. Therefore, many standard geometric data structures can support this query in time $\mathcal{O}(d)$ or $\mathcal{O}(d \log n)$.

Orthogonal range query oracle. There are many efficient data structures supporting the *orthogonal range query oracle* and actually, orthogonal range queries are perhaps the most widely supported geometric queries (for survey expositions, see, e.g., [1, 3, 6]). One of the first data structures for orthogonal range searching is the *quadtree*. Despite its bad worst-case behavior, the quadtree is still used in many applications because it provides an easy-to-implement linear-space data structure that often has a very good performance. Another standard data structure for orthogonal range queries is the range tree. It can be used to report the k points in a query range in $O(\log^d n + k)$ time. Using the fractional cascading technique the query time can be improved to $O(\log^{d-1} n + k)$ [20, 25]. The best known data structures for orthogonal range searching based on compressed range trees and some other techniques such as filtering search can be found in [8, 9]. For more details we refer to the survey [3].

[1] For a point $p \in P$ and a set of points $Q \subseteq \mathbb{R}^d$, a $(1+\delta)$-approximate nearest neighbor of p in Q is any point $q \in Q$ such that for every $x \in Q$ it holds that $|p - q| \leq (1 + \delta) \cdot |p - x|$.

Cone nearest neighbor oracle. In the seminal paper on Euclidean minimum spanning trees, Yao [26] examined algorithms for cone nearest neighbor in the cones with the angular diameter $\pi/4$. Cone nearest neighbor queries have been also studied extensively in follow-up papers dealing with the EMST problem (see, e.g., [2]).

Cone approximate nearest neighbor oracle. Cone *approximate* nearest neighbor queries have been widely investigated. They play an important role in the context of construction of Euclidean spanners (see, e.g., [4, 5, 14, 21]). And thus, among others, Ruppert and Seidel [21] show how to answer a query in amortized time $\mathcal{O}(n \log^{d-1} n)$ per cone in C, simultaneously for all points in P; a similar construction is presented in [5]. Arya *et. al.* [4] present a fully dynamic algorithm which in polylogarithmic time supports cone approximate nearest neighbor queries. Notice also that a single cone approximate nearest neighbor query can be answered using a logarithmic number of *simplex (triangular) range queries*, which is another classical geometric data structure (see, e.g., [1, 3, 6]).

3 Estimating the EMST with bounding cube and range queries

In this section we describe a natural approach to the approximation of $\text{EMST}(P)$ using minimum bounding cube oracle and orthogonal range queries. This approach, by itself, does not give a good enough multiplicative approximation, but is used as a building block in the sublinear algorithm we present later. For simplicity, we only describe in detail the two-dimensional case ($d = 2$); the algorithm can be generalized to arbitrary d in an obvious way. The algorithm we supply is deterministic and outputs a value w such that $\frac{1}{\alpha}\,\text{EMST}(P) - \beta \le w \le \alpha\,\text{EMST}(P) + \beta$, where $\alpha = \mathcal{O}(n^{1/8} \log n)$, and $\beta = \mathcal{L}\, n^{-c}$, where $\mathcal{L}$ is the side-length of a minimal bounding cube of P and c is a constant. The algorithm has a running time of $\mathcal{O}(n^{1/2})$. We also show that any algorithm that uses the same running time (in fact, the same amount of queries and arbitrary large running time) cannot significantly improve the quality of the approximation.

3.1 The quad-tree algorithm

We apply a standard quad-tree subdivision to the bounding cube $\mathcal{BC}$ (see, e.g., [6, Chapter 14]). That is, we first partition $\mathcal{BC}$ into four disjoint blocks (squares) of equal size. We can check which blocks contain points from P via orthogonal range queries. We then further subdivide the nonempty blocks, and iterate this process as long as fewer than $\sqrt{n}$ queries are made. This induces a tree structure on the blocks, where a block at level i has side length $\mathcal{L}/2^i$. Let k be the depth of this tree. We may assume that all nonempty blocks at level $k-1$ were subdivided into subblocks (of level k) and each subblock of level k was queried. Let B be the set of nonempty blocks at level k and let $b = |B|$. Clearly $b = \mathcal{O}(\sqrt{n})$. We now run any minimum spanning tree algorithm (as we will see later, a $(1+\varepsilon)$-approximation is good enough) on the centers of the blocks in B. This would result in a value L. We set $U = L + s\sqrt{b\,n}$, where $s = \mathcal{L} \cdot 2^{-k}$ and output the value $w = \sqrt{L\,U}$ as an approximation for $T^* = \text{EMST}(P)$.

Claim 1 *For an arbitrary constant c, $\frac{1}{\alpha}T^* - \beta \le w \le \alpha T^* + \beta$, where $\alpha = \mathcal{O}(n^{1/4} \log n)$ and $\beta = \mathcal{L}\, n^{-c}$.*

Proof. First note that the minimum spanning tree of any n points in a d-dimensional cube with side-length h is $\mathcal{O}(h\, n^{\frac{d-1}{d}})$ and this bound is tight (i.e., it is achievable for some inputs), see, e.g., [19]. Now, we set L^* be the weight of a minimum weight tree that touches every block in B. It is easy to see that $L^* \le T^* \le U$ (the last inequality is by the above upper bound and using convexity).

Assume now that $b \geq \sqrt{n}/(4(c+1)\log n)$; then it can be seen that L upper bounds L^* and approximates it within an additive term of $\mathcal{O}(s\,b)$, and hence within a constant factor, say δ. Namely, $a \cdot b \cdot s \leq L^* \leq L \leq \delta \cdot L^*$ for some constants a and δ.

Hence, as U is an upper bound on T^*, the approximation factor is $\alpha = \max\{U/w, w/L^*\}$. By our choice of w and the fact that L approximates L^* up to a constant we get $\alpha = \mathcal{O}\left(\frac{U}{w}\right) = \mathcal{O}\left(\sqrt{\frac{U}{L^*}}\right) = \mathcal{O}\left(\left(\frac{s\sqrt{b\,n}}{L}\right)^{1/2}\right) = \mathcal{O}\left((n/b)^{1/4}\right)$ (where the last inequality follows by plugging in the expression for U and L and the previous follows from the fact that L approximates L^* within a constant factor). Now, by the above bound on L and on b we obtain that $\alpha \leq \tilde{\mathcal{O}}(n^{1/8})$. Note that, if we used an approximation L' guaranteed to be within a constant factor of L, we would still get the same result.

Assume now that $b < \sqrt{n}/(4(c+1)\log n)$. Then it can be seen that the depth of the quad-tree is at least $(c+1)\log n$ and hence $s \leq \mathcal{L} \cdot n^{-(c+1)}$. Therefore, the additive term is upper bounded by $U - L \leq \mathcal{O}(s \cdot \sqrt{b\,n}) = \mathcal{O}(n^{-(c+1)} \cdot \mathcal{L} \cdot n) = \mathcal{O}(\mathcal{L} \cdot n^{-c})$. □

A note on the running time is due here. We use $\mathcal{O}(\sqrt{n})$ queries in the course of constructing the quad-tree. Next, we have to find the minimum spanning tree (or any $(1+\delta)$ approximation to it for any fixed δ). In the two-dimensional case this can be done in $\widetilde{O}(\sqrt{n})$ time [22], and this term dominates the total complexity.

Higher dimensions: In the case of dimension $d > 2$ the quad-tree has to be replaced with a 2^d-ary tree. The algorithm will be run similarly to the above until $\mathcal{O}(2^d\sqrt{n})$ queries have been made, and all rectangles at the bottom level have been queried. Then, L is set similarly to the two-dimensional case, and $U = L + s \cdot n^{\frac{d-1}{d}} \cdot b^{1/d}$. The approximation w for T^* is taken to be the same. To have an efficient running time, a constant approximation for L can be used, rather then the exact value. This can be done in time $\mathcal{O}(n\log n)$ by Callahan and Kosaraju result [7].

It is easy to see that the following replaces Claim 1 with an analogous proof.

Claim 2 *For an arbitrary constant c, $\frac{1}{\alpha}T^* - \beta \leq w \leq \alpha T^* + \beta$, where $\alpha = \mathcal{O}(2^{d/2} \cdot n^{(d-1)/4d}\log n)$ and $\beta = \mathcal{L}\,n^{-c}$.*

As it turns out, the above quality of approximation is nearly optimal for the given time bound as shown by the following claim (shown only for the two-dimensional case, a similar result is true for the d-dimensional case as well).

Claim 3 *Any deterministic algorithm for approximating* EMST(P) *in the two-dimensional case that uses $\mathcal{O}(\sqrt{n})$ orthogonal range queries has an approximation factor of $\Omega(n^{1/4})$.*

Proof. Consider any deterministic algorithm that uses at most $\sqrt{n}$ range queries. Consider the following adversary for supplying the answer to the queries: The adversary will subdivide the unit square into a mesh of squares, each of side length $s = \frac{n^{-1/4}}{10}$, namely into $100\sqrt{n}$ squares, denoted blocks. The adversary commits itself to locate $\sqrt{n}/100$ input points in each block. In what follows, the adversary will mark some blocks in which he will commit to the internal location of points. The invariant that is kept is that in unmarked blocks, any configuration of input points is still consistent with the answers so far.

At the beginning no block is marked. Now, for each queried rectangle, if the query intersects an unmarked block then the adversary will answer “not-empty.” In addition it will choose one unmarked block that intersects the given query, mark it and commit to have all points in that block, in an arbitrary single point in the intersection. If the query intersects only previously marked blocks, then if it contains any of the previous locations in which the adversary has already committed to have input points then a "non-empty" answer will be given (this is forced). If the query does not include any of the previous locations in which the adversary has committed to have input points then the adversary will answer "empty."

Keeping up this way, it is easy to see that the adversary can supply consistent answers to all $\sqrt{n}$ queries.

At the end, since there are $100\sqrt{n}$ blocks while the adversary has marked at most $\sqrt{n}$ blocks, in $99\sqrt{n}$ blocks there is complete freedom as to where the input points are located within such block. Now notice that if the adversary chooses to locate all points within a block in one (arbitrary) point then the minimum spanning tree is of cost $\mathcal{O}(n^{1/4})$, while, if it chooses to locate the points in each unmarked block spread uniformly within the block, then the cost of the tree is $\Omega(\sqrt{n})$. Hence the lower bound follows. □

Finally, we note that our choice of using $\mathcal{O}(\sqrt{n})$ orthogonal range queries was arbitrary; one can use a different number of queries and obtain a whole range of tradeoffs between the running time and the quality of approximation. As another example in this tradeoff, assume that αn queries are made. The adversary divides the square into βn blocks and proceeds as below. In this situation, the algorithm has no way of telling whether all cells have their points concentrated in one location, resulting in an EMST of at most $\sqrt{\beta n}$ or that they are distributed in the unmarked blocks as above, resulting in an EMST or at least $\frac{(\beta-\alpha)\sqrt{n}}{\beta}$, giving an approximation ratio of at least $\frac{\beta-\alpha}{\beta\sqrt{\beta}}$. To exploit the tradeoff here, we set $\beta = 1/4$. To obtain an approximation ratio of ε, the number of queries must be at least $n(1/4 - \varepsilon/8)$. One can exploit this tradeoff in other ways also.

4 Two related previous results

We now describe two previous results that we utilize in our EMST algorithms: the concept of Yao graphs [26] and an algorithm for approximating the MST in bounded degree graphs due to Chazelle *et al.* [10].

4.1 Yao graphs

Yao graphs are Euclidean graphs that relate the EMST to the cone nearest neighbor oracle presented in Section 2.1. Fix an integer $d \geq 2$. Let $\mathcal{C}$ be a collection of d-dimensional cones with apex at the origin such that (a) each cone has angular diameter[2] at most θ, where θ is some fixed angle, and (b) $\bigcup_{C\in\mathcal{C}} C = \mathbb{R}^d$. There is always such a collection $\mathcal{C}$ of $\mathcal{O}(d^{3/2} \cdot \sin^{-d}(\theta/2) \cdot \log(d \sin^{-1}(\theta/2)))$ cones (not necessarily disjoint); note that for constant d and θ this bound is $\mathcal{O}(1)$. Yao [26] gives one possible construction for such a collection. For a point $p \in \mathbb{R}^d$ and a cone $C \in \mathcal{C}$, let C_p be $p + C = \{a + p \ : \ a \in C\}$, that is, a translation of C so that its apex is at p. Let $N_P\langle p, C\rangle$ be the nearest neighbor of the apex p of C_p in the set $(P \setminus \{p\}) \cap C_p$. Given a point set P and a collection of cones $\mathcal{C}$, the *Yao graph of P (with respect to $\mathcal{C}$)* is the Euclidean graph G with vertex set P and (undirected) edge set $E = \{(p, q) \mid \exists C \in \mathcal{C}$ such that $q = N_P\langle p, C\rangle\}$. That is, each $p \in P$ is connected to its nearest neighbor in each cone which has p at its apex. The following result due to Yao [26] motivates our use of these graphs.

Claim 4 [26] *Let P be a point set in $\mathbb{R}^d$. Let G be the undirected Yao graph for P with $\theta < \pi/3$. Then, the Euclidean minimum spanning tree of P is a subgraph of the Yao graph G.* □

[2]The angular diameter of a cone C in $\mathbb{R}^d$ having its apex at point $p \in \mathbb{R}^d$ is defined as the maximum angle between any two vectors $\overrightarrow{px}$ and $\overrightarrow{py}$, $x, y \in C$.

4.2 Chazelle et al.: approximate MST in low-degree graphs

Our algorithms make use of a recent algorithm for estimating the weight of MST in graphs due to Chazelle *et al.* [10]. This algorithm assumes that the input graph (i) is represented by an adjacency list, (ii) has degree at most ν (the full version of [10] allows ν to be the average degree), and (iii) has known minimum and maximum edge weights, where the ratio of the maximum edge weight to the minimum is Λ. Then, for $0 < \varepsilon < \frac{1}{2}$, the algorithm estimates the weight of the minimum spanning tree with a relative error of at most ε, with probability at least $\frac{3}{4}$, and runs in time $\mathcal{O}(\nu \cdot \Lambda \cdot \log(\nu \Lambda/\varepsilon)/\varepsilon^3)$. (The authors also give a nearly matching lower bound of $\Omega(\nu \cdot \Lambda/\varepsilon^2)$ on the time complexity of any ε-approximation algorithm for the MST.)

Let $H = (V, E)$, be an input graph having n vertices with maximum degree ν and edge weights in the interval $[1, \Lambda]$. For any $w \in \mathbb{R}$, let $H^{(w)}$ denote the maximal subgraph of H containing edges of weight at most w, and c_w denote the number of connected components in $H^{(w)}$. The main ingredient of the algorithm from [10] is a procedure called "approx-number-connected-components", which we call *ACC* from this point on, run on $H^{(w)}$ for estimating c_w for $w = (\frac{1}{2} + i) \cdot \varepsilon$ with $i = 1, 2, \ldots, \Lambda/\varepsilon$. For integer weights, the weight of the MST of H is equal to $n - \Lambda + \sum_{j=1}^{\Lambda-1} c_j$. The algorithm uses the above estimations to produce a value which, with probability at least $\frac{3}{4}$, is a $(1 \pm \varepsilon)$-approximation of the MST of H.

Procedure *ACC* works by sampling $\mathcal{O}(1/\varepsilon^2)$ vertices in H. For each sampled vertex u, a random estimator X_u is computed by traversing $H^{(w)}$ from u (for example, using breadth-first search) with a stochastic stopping rule. X_u is a random variable whose distribution is a function of only the size of the connected component containing u (i.e., the number of vertices reached from u in the traversal) in $H^{(w)}$. The simple relation between these sizes and c_w together with the fact that the distribution of X_u is concentrated around the expected value yields the connection between X_u and c_w. Procedure *ACC* runs in expected time $\mathcal{O}(\nu\, \varepsilon^{-2} \log(\Lambda/\varepsilon))$. Therefore, the expected running time of the algorithm in [10] is $\mathcal{O}(\Lambda\, \nu\, \varepsilon^{-3} \log(\Lambda/\varepsilon))$.

5 A simple estimation for EMST using Yao graphs

The algorithm we present in this section is conceptually an important component of the sublinear algorithm we design later in Section 6. It combines the two results described in Section 4. Our algorithm uses the cone nearest neighbor oracle and achieves a query complexity of $2^{\mathcal{O}(d)} \cdot \widetilde{\mathcal{O}}(\Lambda/\varepsilon^2)$.

Since by Claim 4 the undirected Yao graph G for P contains all edges of the EMST of P, it is natural to try to apply the algorithm of Chazelle *et al.* to G to estimate the weight of the EMST of P. To do that efficiently, instead of generating G at the beginning of the algorithm, we generate the edges of G (using the cone nearest neighbor queries) only when the edges are needed in the algorithm. That is, whenever the algorithm needs edges adjacent in G to a vertex p, we use the cone nearest neighbor query to obtain the nearest neighbor of p in each cone in $\{p + C\}_{C \in \mathcal{C}}$. Motivated by Claim 4, we set the angular diameter of the cones to $\pi/4$. This creates parts of an *implicit directed* Yao graph $\overline{G}$ on P with edges (p, q) such that there is a $C \in \mathcal{C}$ where $q = N_P\langle p, C\rangle$.

The above approach has a number of problems. First, the algorithm of Chazelle *et al.* requires the input graph to be undirected and represented by an adjacency list, whereas in our model, we have fast access only to the *out-going edges* at a vertex in $\overline{G}$. Furthermore, the running time is linear in Λ, which can be arbitrarily large. The following lemma helps in overcoming the first difficulty, while the second one is tackled in the main algorithm in Section 6. The proof of Lemma 1, being a special case of Claim 6, is omitted.

Lemma 1 *Let n_u^ℓ be the number of vertices in $\mathbb{K}_P$ that are reachable from u using only edges of weight at most ℓ. Let m_u^ℓ be the number of vertices in directed Yao graph $\overline{G}$ reachable from u using only edges of weight at most ℓ. Then $m_u^\ell = n_u^\ell$.* □

Equipped with this lemma, we can modify the algorithm due to Chazelle *et al.* to obtain its efficient implementation in our model. The only difference is in procedure *ACC*. We still sample $\mathcal{O}(1/\varepsilon^2)$ vertices and randomly traverse $H^{(w)}$ from the sampled vertices. To implement the traversing algorithm we explore the graph in a breadth-first search fashion by going to the *outgoing* neighbors of the vertices that are closer than the current threshold weight w. Such a procedure can be easily implemented in our model by using the cone nearest neighbor queries; the running time is proportional to the number of the edges traversed. To estimate the value of c_w we use the same estimators as in [10]. Since for each vertex u in the sample, the distribution of X_u depends only on m_u^w, the number of the vertices reachable from u in $H^{(w)}$, by Lemma 1, we can conclude that X_u has the same distribution as in the algorithm of Chazelle *et al.* [10]. Therefore, the quality of this algorithm of the estimation of EMST of P is the same as in the algorithm of Chazelle *et al.* [10]. Since the maximum out-degree of the directed Yao graph is $2^{\mathcal{O}(d)}$, the modified procedure *ACC* has identical complexity to that of running the original algorithm of Chazelle *et al.* in a (undirected) graph with maximum degree $2^{\mathcal{O}(d)}$. Thus, we obtain the following theorem.

Theorem 1 *Let P be a set of points in $\mathbb{R}^d$. Assume the value Λ of the spread of P is known and access to a cone nearest neighbor oracle for P is given. Then, there is an algorithm that outputs a value Υ which, with probability at least $\frac{3}{4}$, approximates the values of EMST(P) to within a factor of $1 \pm \varepsilon$ with query complexity $\widetilde{\mathcal{O}}\left(2^{\mathcal{O}(d)} \cdot \Lambda/\varepsilon^3\right)$.* □

For constant d and ε, this complexity is $\widetilde{\mathcal{O}}(\Lambda)$, which is sublinear for $\Lambda = o(n)$. However, for example, on the plane, Λ is known to be $\Omega(\sqrt{n})$, and in general, Λ may be arbitrarily large. In the next section, we discuss our main contribution, which is a truly sublinear-time approximation algorithm whose complexity is independent of Λ.

6 Sublinear-time approximation algorithm

In this section we show how the two algorithms from Sections 3 and 4 can complement each other. In addition to improving the running time, our algorithm requires a weaker computational model, in which the *cone nearest neighbor query* is replaced by the *cone $(1+\delta)$-approximate nearest neighbor query*.

6.1 Overview of the algorithm

In Section 6.2, we begin by partitioning a minimal bounding cube $\mathcal{BC}$ of P into blocks of equal size; we then consider only blocks containing points from P. Next, we group blocks that are "close" to each other together, calling the resulting clusters *connected block-components*. The algorithm then proceeds in two phases. First, in Section 6.5, we show how to approximate the weight of a minimum spanning forest (MSF) of the connected block-components by using the ideas of Section 5. We then, in Section 6.6, approximate the optimal way to connect different connected block-components. We prove in Lemma 2 that the MSF of the connected block-components combined with the optimal set of edges joining them approximates the EMST of P.

In our analysis, throughout the entire section we assume that $0 < \varepsilon < \frac{1}{15}$.

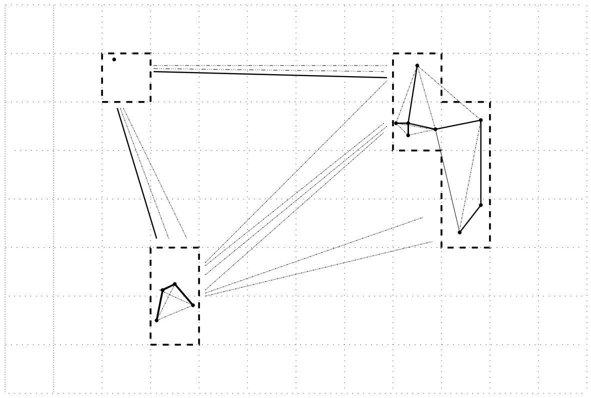

Figure 1: Block-partitioning, connected block-components and a schematics to the sublinear algorithm.

6.2 Partitioning the bounding cube

After the translation and scaling of the points in P we can assume that $\mathcal{BC}$, the bounding cube of P, is $[0, n/\varepsilon]^d$. In particular, the side length is $\mathcal{L} = n/\varepsilon$ and we have a trivial lower bound $\text{EMST}(P) \geq n/\varepsilon$.

We follow the approach from Section 3 with small modifications, by extending it to higher dimensions and applying a different stopping procedure. We first partition $\mathcal{BC}$ into 2^d disjoint blocks of equal size, then partition iteratively the nonempty ones into 2^d disjoint subcubes, and so on. Call a block at level i an *active block* if it contains a point from P. Let b_k be the number of active blocks at level k (number of blocks that contain points from P), and $\Delta_k = \mathcal{L}/2^k$ be the side length of blocks in the kth level of the subdivision. Let $b^* = \max\{\varepsilon^{d/2-3}\sqrt{n}, 2^{d+1}\}$. We stop our subdivision at the first level k_0 such that either $b_{k_0} \geq b^*$ or $\Delta_{k_0} < 2\,\varepsilon$. Let $b = b_{k_0}$ and $\Delta = \Delta_{k_0}$. Notice that $b \leq 2^d\, b^*$ and $\Delta \geq \varepsilon$. By our arguments from Section 3, the active blocks at level k_0 can be found by querying the range query oracle $\mathcal{O}(b\, 2^d \log(n/(\varepsilon\,\Delta))$ times.

6.3 Spanners and connected block-components

For any $t \geq 1$, a *t-spanner* (see, e.g., [7, 14, 17]) for a set S of points in a Euclidean space is any Euclidean graph G with the vertex set S such that for every pair of points $x, y \in S$ there is a path in G between x and y of total length at most $t \cdot |x - y|$.

In our analysis, we will frequently use centers of blocks as the representatives of the blocks. Let B be the set of *centers of active blocks* and let $\mathcal{SPN}$ be a $(1+\varepsilon/4)$-spanner of B with $\mathcal{O}(b\,(4/\varepsilon)^{d-1})$ edges. Such a spanner can be found in time $\mathcal{O}(b \log b + b \log(1/\varepsilon)\,\varepsilon^{-d}) = \widetilde{\mathcal{O}}(\sqrt{n}\;\varepsilon^{3-d/2})$ [7].

Call two blocks *close* if the distance between their centers in the graph $\mathcal{SPN}$ is at most $\Gamma \cdot \Delta$, where $\Gamma = 14\sqrt{d}/\varepsilon$. We use equivalence classes of the transitive closure of the relation *close* to define the *connected block-components*. That is, two blocks are in the same connected block-component if there is a sequence of active blocks between them, where every consecutive pair of blocks in the sequence is close. We shall abuse notation and refer also to the partition of P induced by the connected components as *connected block-components*. Notice that all connected block-components can be found in time proportional to the number of edges in $\mathcal{SPN}$, which is $\mathcal{O}(b\,(4/\varepsilon)^{d-1})$.

6.4 The EMST of P and connected block-components

We refer to the *spanning forest* of a graph G as a union of spanning trees of the connected components of G. A *minimum spanning forest* of G, denoted by $\text{MSF}(G)$, is a spanning forest of G of minimum weight.

Let E_{in} be the set of edges of $\mathbb{K}_P$ whose endpoints lie within the same connected block-component. Let $W = (\Gamma + \sqrt{d})\,\Delta$. We now relate block-components to the distances between points.

Observation 1 *Let p and q be an arbitrary pair of points in P.*

1. *If $|p - q| \leq (\Gamma - 4\sqrt{d})\,\Delta$ then p and q are in the same connected block-component.*
2. *If p and q are in the same connected block-component then there is a path between p and q consisting of edges in E_{in} that are all of length at most $(\Gamma + \sqrt{d})\,\Delta = W$.*
3. *If $|p - q| > (\Gamma + \sqrt{d})\,\Delta = W$ and p and q are in the same connected block-component, then* EMST(P) *does not contain the edge pq.*

Proof. For any point $p \in P$, we let c_p denote the center of the block at level k_0 that contains p.

To see the first assertion, notice that if $|p - q| \leq (\Gamma - 4\sqrt{d})\,\Delta$ then $|c_p - c_q| \leq |p - q| + \sqrt{d}\,\Delta \leq (\Gamma - 3\sqrt{d})\,\Delta$. Therefore, the distance in $\mathcal{SPN}$ between c_p and c_q is at most $(1+\varepsilon/4)\,(\Gamma - 3\sqrt{d})\,\Delta \leq \Gamma\,\Delta$, which implies the first claim. Next, this also implies the existence of a path $c_p = c^{(0)}, c^{(1)}, \dots, c^{(s)} = c_q$ in $\mathcal{SPN}$ such that $|c^{(i)} - c^{(i+1)}| \leq \Gamma \cdot \Delta$ for all i. Clearly, the corresponding path $p = p^{(0)}, p^{(1)}, \dots, p^{(s)} = q$ with $c^{(i)} = c_{p^{(i)}}$ shows the second assertion, since $|p^{(i)} - p^{(i+1)}| \leq |c^{(i)} - c^{(i+1)}| + \sqrt{d}\,\Delta \leq \Gamma\Delta + \sqrt{d}\,\Delta$. The third assertion follows from the second one and the fact that the (strictly) largest edge in a cycle in a graph cannot be part of its MST. □

In our algorithm we use the following graphs:

- G_{block} is the graph containing all edges in E_{in} of weight at most W. By Observation 1, the connected components of MSF(G_{block}) are identical to the connected block-components and the minimum spanning forest of these components is the same as MSF(G_{block}).
- $\overline{G_\delta}$ is the *directed* $(1+\delta)$-*Yao graph* that is obtained from $\mathbb{K}_P$ using the cone $(1+\delta)$-approximate nearest neighbor oracle. We use the same definitions as in the definition of directed Yao graphs and we formally define $N_P^{(1+\delta)}\langle p, C\rangle$ to be the point that is returned by the cone $(1+\delta)$-approximate nearest neighbor oracle for p and C. If $(P \setminus \{p\}) \cap C_p = \emptyset$, then $N_P^{(1+\delta)}\langle p, C\rangle$ is undefined. Then, $\overline{G_\delta}$ is a directed Euclidean graph on P with the edge set containing an edge (p, q) if there is $C \in \mathcal{C}$ such that $q = N_P^{(1+\delta)}\langle p, C\rangle$.
- $\mathcal{M}$ is the minimum weight subgraph of $\mathbb{K}_P$ that, when added to G_{block}, forms a connected graph.
- G_{out} is the same as $\mathbb{K}_P$ except that the weights of edges in E_{in} are considered to be zero. Observe that the weight of MST(G_{out}) is identical to the weight of $\mathcal{M}$.

The following lemma displays the two-level nature of the algorithm that we will present.

Lemma 2 *The sum of the weights of* MSF(G_{block}) *and* MST(G_{out}) *is a $(1 + \varepsilon/2)$-approximation of* EMST(P).

Proof. We show that the union of MSF(G_{block}) and $\mathcal{M}$ is a spanning tree of $\mathbb{K}_P$ whose weight approximates the weight of EMST(P) to within a factor of $1 + \varepsilon/2$. From that the lemma follows immediately.

Clearly, the union of MSF(G_{block}) and $\mathcal{M}$ forms a spanning tree of $\mathbb{K}_P$. To prove the second part of the claim, let us consider an undirected graph G^* obtained from $\mathbb{K}_P$ by decreasing to $(\Gamma - 4\sqrt{d})\,\Delta$ the weight of every edge in E_{in} having weight larger than $(\Gamma - 4\sqrt{d})\,\Delta$ and smaller than or equal to W. (Note that we change only the weights of the edges in G_{block}.) Since the weight of every edge decreases

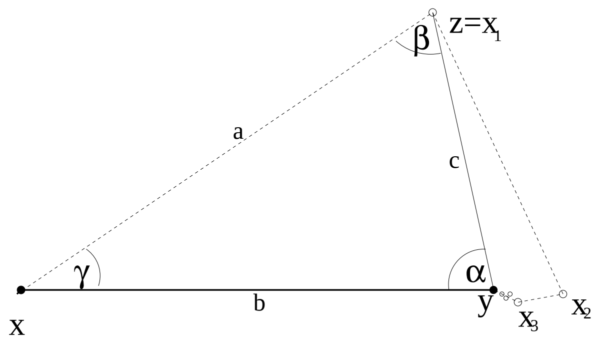

Figure 2: Illustration to the proof of Claim 6. The figure shows the reachability in $\overline{G_\delta}$. The dashed line is the path showing the connectivity of x and y in $\overline{G_\delta}^{((1+\delta)\,\tau)}$.

by a factor of at most $\frac{W}{(\Gamma-4\sqrt{d})\,\Delta} = \frac{(\Gamma+\sqrt{d})\,\Delta}{(\Gamma-4\sqrt{d})\,\Delta} \le 1+\varepsilon/2$, we have $\textsc{mst}(G^*) \ge \textsc{emst}(P)/(1+\varepsilon/2)$. Notice further that by Observation 1, each edge in G^* that is not in G_{block} has weight larger than $(\Gamma-4\sqrt{d})\,\Delta$. This means that $\textsc{mst}(G^*)$ must contain a minimal spanning forest of G_{block}, and hence the weight of the union of $\textsc{msf}(G_{block})$ and $\mathcal{M}$ is a $(1+\varepsilon/2)$ approximation of $\textsc{emst}(P)$. □

6.5 First level – estimating the weight of $\textsc{msf}(G_{block})$

In this section we show how to estimate the weight of the $\textsc{mst}$ within a single block component. This, combined for all block components, yields an estimate on the weight of $\textsc{msf}(G_{block})$. Since our model does not allow constant-time access to the edges of G_{block}, we will use the directed Yao graph $\overline{G_\delta}$ to estimate the weight of $\textsc{msf}(G_{block})$. Our analysis will explore the relationship between $\overline{G_\delta}$ and G_{block}.

For a weighted graph H denote by $\beta \cdot H$ the graph H with edge weights multiplied by β. Recall that $H^{(r)}$ denotes the subgraph of H consisting of the edges of weight at most r, and c_r is the number of connected components in $G^{(r)}_{block}$. Let n^r_u and m^r_u be the number of vertices in $G^{(r)}_{block}$ and in $\overline{G_\delta}^{(r)}$ that are reachable from u respectively. Notice that $c_r = \sum_{u\in P} 1/n^r_u$. Analogously, define $c^*_r = \sum_{u\in P} 1/m^r_u$. Also, let $\hat{c}_r$ be the number of connected components in $(1+\delta)\cdot G^{(r)}_{block}$.

The following is from [10].

Claim 5 *Let $G^{(l)}$ denote the subgraph of a graph G consisting of all the edges of weight at most l. Define $c^{(l)}$ to be the number of connected components in $G^{(l)}$. Then, for integer $w \ge 2$, $M(G) = n - w + \sum_{i=1}^{w-1} c^{(i)}$.*

From the above (see also Section 4) we have that

$$\textsc{msf}(G^{(r)}_{block}) \;\le\; n - rc_r + \sum_{i=1}^{r-1} c_i \;\le\; \textsc{msf}(G^{(r)}_{block}) + n \quad . \tag{1}$$

Since we only have access to $\overline{G_\delta}$, we can only deal with the c^*_r's rather than the c_r's. To bound the error due to this replacement, now we relate reachability in $\overline{G_\delta}$ to reachability in G_{block}.

Claim 6 *Let $\varepsilon \le \frac{1}{5}$ and $\delta \le \frac{1}{10}$. Then for every r and every $u \in P$, $n^{r/(1+\delta)}_u \le m^r_u \le n^r_u$. In particular, $c_{r/(1+\delta)} \ge c^*_r \ge c_r$.*

Proof. Let us first notice that $m_u^r \le n_u^r$ follows directly from the definition. To show that $n_u^{r/(1+\delta)} \le m_u^r$, it suffices to show that for every τ, if a vertex y is reachable in $G_{block}^{(\tau)}$ from a vertex x, then y is reachable from x in $\overline{G_\delta}^{((1+\delta)\,\tau)}$. Assume that y is reachable from x in $G_{block}^{(\tau)}$; this implies that x and y are in the same connected block-component. Assume further, without loss of generality, that $\tau \le W$ (indeed, if $\tau > W$ then $G_{block}^{(\tau)} = G_{block}^{(W)}$).

Let z be the $(1+\delta)$-approximate nearest neighbor of x (returned by the cone approximate nearest neighbor oracle) in the cone C_x containing y. Clearly, if $z = y$, then the claim holds. So let us assume that $z \neq y$. Let $a = |x-z|$, $b = |x-y|$, $c = |y-z|$, and $\alpha = \measuredangle(xyz)$, $\beta = \measuredangle(xzy)$, and $\gamma = \measuredangle(yxz)$, see Figure 2. Notice that since y and z are contained in the cone C_x with the angular diameter $\pi/4$, we have $\gamma \le \pi/4$.

We first show the following three inequalities: (i) $a \le (1+\delta)\,b$, (ii) $c < b$, and (iii) $\min\{a,c\} \le b/(1+\varepsilon)$. Inequality (i) follows directly from the definition of the cone approximate nearest neighbor oracle. To prove inequality (ii), let us suppose that $c \ge b$. Then, $\beta \le \gamma$, and since $\gamma \le \pi/4$, we obtain that $\alpha \ge \pi/2$. This in turn implies that $a \ge \sqrt{b^2+c^2} \ge \sqrt{2}\,b$, which contradicts the first inequality that $a \le (1+\delta)\,b \le 1.1 \cdot b$. For inequality (iii), we first use the law of cosines to get $c^2 = a^2+b^2-2\,a\,b\,\cos\gamma \le a^2+b^2-\sqrt{2}\,a\,b$, since $\gamma \le \pi/4$. To show $\min\{a,c\} \le b/(1+\varepsilon)$ we assume $a > b/(1+\varepsilon)$ and show $c \le b/(1+\varepsilon)$. Since $a > b/(1+\varepsilon) \ge \frac{\sqrt{2}}{2}b$, the expression $a^2+b^2-\sqrt{2}ab$ increases with a. Therefore, by inequality (i) we obtain

$$c^2 \;\le\; a^2+b^2-\sqrt{2}ab \;\le\; ((1+\delta)b)^2+b^2-\sqrt{2}(1+\delta)b^2 \;=\; b^2((2-\sqrt{2})(1+\delta)+\delta^2) \;\le\; (b/(1+\varepsilon))^2 \;,$$

where the last inequality holds for $\varepsilon \le \frac{1}{5}$ and $\delta \le \frac{1}{10}$.

Now, we prove the claim using inequalities (i–iii). Assume, without loss of generality, that $|x-y| \le \tau$; otherwise apply the following arguments to all edges on the path between x and y in $G_{block}^{(\tau)}$ (all the edges on this path are of length at most τ). We define inductively the sequence $x = x_0, x_1, x_2, \dots y$ such that for every i, if $x_i \neq y$, then x_{i+1} is the $(1+\delta)$-approximate nearest neighbor of x_i in the cone C_{x_i} containing y. By inequality (ii), the sequence $|x_i - y|$ is strictly decreasing. This immediately implies that $x_i = y$ for some i, and so the sequence is finite.

Next, we show inductively that each x_i is in the same connected block-component as y. Suppose that x_i is in the same connected block-component as y. Since the sequence $|x_i-y|$ is decreasing and since $|x-y| \le \tau \le W = (\Gamma+\sqrt{d})\cdot\Delta$, we obtain

$$\frac{|x_i-y|}{1+\varepsilon} \;\le\; \frac{|x-y|}{1+\varepsilon} \;<\; \frac{(\Gamma+\sqrt{d})\cdot\Delta}{1+\varepsilon} \;\le\; \frac{(\Gamma+\sqrt{d})\cdot\Delta}{1+\varepsilon} \;.$$

Therefore, using inequality (iii) with $x = x_i$ and $z = x_{i+1}$, we obtain

$$\min\{|x_i-x_{i+1}|, |x_{i+1}-y|)\} \;\le\; \frac{|x_i-y|}{1+\varepsilon} \;\le\; (\Gamma-\sqrt{d})\cdot\Delta \;.$$

Hence, by Observation 1, either x_i and x_{i+1} are in the same connected block-component or x_{i+1} and y are in the same connected block-component. In either case, the transitivity ensures that x_{i+1} and y are in the same connected block-component. We finally observe that inequality (i) implies that $|x_i-x_{i+1}| \le (1+\delta)\,|x_i-y|$, and since $|x_i-y| \le |x-y|$, we obtain $|x_i-x_{i+1}| \le (1+\delta)\,|x-y|$. Hence, the sequence $x = x_0, x_1, x_2, \dots, y$ corresponds to a path contained in a connected block-component having all edges of length at most $(1+\delta)\,\tau$. This implies that y is reachable from x in $\overline{G_\delta}^{((1+\delta)\,\tau)}$. $\square$

Let $W' = \lceil W\,(1+\delta)\rceil$. Motivated by inequality (1), let us introduce an estimator $\mathcal{A}$ for the value of $\text{MSF}(G_{block})$.

$$\mathcal{A} \;=\; n + \sum_{i=1}^{W'-1} c_i^* - W' \cdot c_{W'}^* \;.$$

We analyze now the quality of this estimator.

Lemma 3 $\textsc{msf}(G_{block}) \le \mathcal{A} \le (1+\delta)\cdot \textsc{msf}(G_{block}) + n$.

Proof. Let us first remind that $\hat{c}_r = c_{r/(1+\delta)}$. Next, let us observe that if $r \ge W$ then $c_r = c_W$. As a corollary, $c^*_{W'} = c_W = c_{W'}$. With this, we have the following sequence of inequalities:

$$\begin{aligned}
\textsc{msf}(G_{block}) &\le n + \sum_{i=1}^{W-1} c_i - W\cdot c_W \;\le\; n + \sum_{i=1}^{W'-1} c_i - W'\cdot c_{W'} \;=\; n + \sum_{i=1}^{W'-1} c_i - W'\cdot c^*_{W'} \\
&\le n + \sum_{i=1}^{W'-1} c^*_i - W'\cdot c^*_{W'} \;=\; \mathcal{A} \;\le\; n + \sum_{i=1}^{W'-1} c_{i/(1+\delta)} - W'\cdot c_W \\
&= n + \sum_{i=1}^{W'-1} \hat{c}_i - W'\cdot \hat{c}_{W'} \;\le\; \textsc{msf}((1+\delta)\cdot G_{block}) + n \\
&= (1+\delta)\cdot \textsc{msf}(G_{block}) + n \;\; .
\end{aligned}$$

The first inequality is due to inequality (1). The second one follows from the observation above. Next, we use Corollary 6 and the both observations above. The last inequality is implied by inequality (1). □

We now modify the algorithm of Chazelle *et al.* [10] to obtain a good approximation of $\mathcal{A}$. Let us first notice that similarly as in Section 5, we can easily traverse the graph $\overline{G_\delta}^{(r)}$: each time we want to access all edges incident to a point $p \in P$, we first ask the cone approximate nearest neighbor queries to all cones C_p and then for each nearest neighbor q of p in C_p, we verify if $|p-q| \le r$ and if the blocks to which p and q belong are in the same connected block-component. The first test is a simple $\mathcal{O}(1)$ time calculation, while the second requires the computation of the connected block-components. Establishing that, we can apply the approach from Sections 4.2 and 5 to estimate the value $c^* = \sum_{r=1}^{W'-1} c^*_r$, and hence to estimate the value of $\mathcal{A}$. For this, we run procedure ACC to get an estimator X_r to c^*_r for all $r = 1, 2, \ldots, W'$, and we now show that $X = \sum_{r=1}^{W'-1} X_r$ is a good approximation to $c^* = \sum_{r=1}^{W'-1} c^*_r$.

We have from [10] that after using procedure $ACC(G, \varepsilon, W)$, where G denotes the input graph, ε the approximation parameter and W the stopping condition for the breadth first search, we have $c^* - n/W \le \mathrm{E}X \le c^*$ and $\mathrm{var}X \le 2\,n\,c^*/s$, where s is the number of initial random choices for vertices. Our algorithm yields similar properties, thus, we have the following.

$$c^* - n/2 \le \mathrm{E}X \le c^* \;\; ,$$

and

$$\mathrm{var}X \le 2\,n\,c^*/s \;\; ,$$

where s is the number of random choices of initial vertices in ACC. Next, using the bounds above, the fact that $\textsc{emst}(P) \ge n/\varepsilon$, and Chebyshev's inequality, we have

$$\begin{aligned}
\Pr[|X - c^*| \ge \varepsilon/2\cdot \textsc{emst}(P)] &\le \Pr[|X - \mathrm{E}X| \ge \varepsilon/4\cdot \textsc{emst}(P)] \\
&\le \frac{16\,\mathrm{var}X}{\varepsilon^2\cdot(\textsc{emst}(P))^2} \le \frac{32\cdot n\cdot c^*}{\varepsilon^2\cdot s\cdot(\textsc{emst}(P))^2} \;\; .
\end{aligned}$$

We argue that $32\,n\,c^*/(\varepsilon^2\,s\,(\textsc{emst}(P))^2) = \mathcal{O}\left(\frac{1}{\varepsilon\,s}\right)$, or alternatively, that $(\textsc{emst}(P))^2 = \Omega(n\,c^*/\varepsilon)$. Indeed, if $c^* \le 2\,n/\varepsilon$, then $(\textsc{emst}(P))^2 \ge (n/\varepsilon)^2 \ge 2\,n\,c^*/\varepsilon$, by our assumption in Section 6.2. Otherwise, we have to use a stronger lower bound for $\textsc{emst}(P)$. By Lemma 2, we have

$$\textsc{emst}(P) \;=\; \Omega\left(\textsc{msf}(G_{block}) + \textsc{mst}(G_{out})\right) \;=\; \Omega\left(\textsc{msf}(G_{block}) + W'\cdot(c'_W - 1)\right) \;\; .$$

Next, by Lemma 3, we have

$$(1+\delta)\cdot \textsc{msf}(G_{block}) \;\geq\; \mathcal{A}-n \;=\; c^* - W'\cdot c^*_{W'} \;\; .$$

Hence,

$$\textsc{emst}(P) \;=\; \Omega(c^* - W'\cdot c^*_{W'} + W'\cdot(c'_W - 1)) \;=\; \Omega(c^*) \;\; ,$$

from which it follows that for $c^* > 2\,n/\varepsilon$ we have, $(\textsc{emst}(P))^2 = (\Omega(c^*))^2 = \Omega(n\,c^*/\varepsilon)$, as required.

Summarizing the discussion above, we have always $(\textsc{emst}(P))^2 = \Omega(n\,c^*/\varepsilon)$ and hence

$$\Pr[|X-c^*| \geq \varepsilon/2\cdot \textsc{emst}(P)] \;\leq\; \mathcal{O}(\tfrac{1}{\varepsilon\cdot s}) \;\; .$$

Therefore, if we choose $s=\mathcal{O}(1/\varepsilon)$, then we obtain

$$\Pr[|X-c^*| \geq \varepsilon/2\cdot \textsc{emst}(P)] \;\leq\; 1/4 \;\; .$$

Next, observe that $c^*_{W'}$ is nothing but the number of connected block-components, which is known to the algorithm that computes the connected block-components. This leads to an efficient algorithm that calculates $\mathcal{A}' = n + X - W'\cdot c^*_{W'}$ for which $\Pr[|\mathcal{A}'-\mathcal{A}| > \frac{1}{2}\,\varepsilon\cdot\textsc{emst}(P)] \leq 1/4$. The complexity of this algorithm follows from the analysis from Section 5 and from [10] (result stated in Section 4.2) and is $\widetilde{\mathcal{O}}(W\cdot 2^{\mathcal{O}(d)}/\varepsilon)$ cone approximate nearest neighbor queries. The algorithm approximates c^* to within an additive error of n with probability at least $\frac{3}{4}$ (see [10], presented here in Section 4.2), and hence $\textsc{emst}(P)$ to within an additive error of $\frac{1}{2}\,\varepsilon\cdot\textsc{emst}(P)+\delta\cdot\textsc{emst}(P)+n = (\delta+\frac{1}{2}\,\varepsilon)\cdot\textsc{emst}(P)+n$.

We note that by scaling down all weights by a factor $\lambda > 1$, applying the algorithm above, and then rescaling to the original weight, we decrease the running time by a factor of λ, and increase the additive error by the same factor. In this way we obtain an algorithm that performs $\widetilde{\mathcal{O}}(W\cdot 2^{\mathcal{O}(d)}/(\lambda\,\varepsilon))$ cone approximate nearest neighbor queries and achieve an additive error of $(\delta+\frac{1}{2}\,\varepsilon)\cdot\textsc{emst}(P)+\lambda\cdot n$.

Let us examine the term W/λ in the running time and the additive error term $(\delta+\frac{1}{2}\,\varepsilon)\cdot\textsc{emst}(P)+\lambda\cdot n$. Recall that there are two possible termination states: $b\geq b^*$ or $\Delta < 2\,\varepsilon$.

Consider first the case $b\geq b^*$. Since P has b active blocks of size Δ we have that $\textsc{emst}(P)\geq \frac{1}{2}\,\Delta\,(\lceil b/2^d\rceil - 1)$. This bound is achieved by considering a subdivision of the active block to $2^d\,b$ subcubes of size $\Delta/2$. Now color these subblocks with 2^d different colors, using the same arrangement of colors for each of the original active blocks. This induces a partition of the active blocks into 2^d monochromatic sets. There has to be a set of $\lceil b/2^d\rceil$ points in P from different active blocks that are colored the same. Clearly, the minimal distance between these points must be at least $\Delta/2$, and hence the bound. Once we established that $\textsc{emst}(P)\geq\frac{1}{2}\,\Delta\,(\lceil b/2^d\rceil - 1)$, we use the inequalities $b\geq b^*\geq 2^{d+1}$ to get $\textsc{emst}(P)\geq\frac{1}{4}\cdot\Delta\cdot b/2^d$. Setting $\lambda=\frac{\Delta\,b\,\varepsilon}{8\cdot 2^d\,n}$ we upper bound the relative error by

$$\delta+\varepsilon/2+\frac{\lambda\cdot n}{\frac{1}{4}\cdot\Delta\cdot b/2^d} = 1+\delta+\varepsilon \;\; .$$

The running time, using the fact that $b\geq b^*\geq\varepsilon^{d/2-3}\sqrt{n}$, is bounded by

$$\widetilde{\mathcal{O}}(W'/(\lambda\,\varepsilon)) \;=\; \widetilde{\mathcal{O}}(\sqrt{d}\cdot 2^d\cdot n/(b\varepsilon^3)) \;\leq\; \widetilde{\mathcal{O}}(\sqrt{n}\cdot 2^d\cdot\sqrt{d}/\varepsilon^{d/2}) \;=\; \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2}) \;\; .$$

On the other hand, when $\Delta < 2\,\varepsilon$, we use the trivial lower bound $\textsc{emst}(P)\geq n/\varepsilon$, and by setting $\lambda = 1/2$ obtain a multiplicative error of $1+\delta+\varepsilon$. In this case notice that $W' = \lceil(1+\delta)\cdot\Delta\cdot\sqrt{d}\cdot(1+14/\varepsilon)\rceil = \mathcal{O}(1)$. And so, we bound the running time by

$$W/(\lambda\,\varepsilon^3) \;=\; \widetilde{\mathcal{O}}(\varepsilon^{-3}) \leq \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$$

for $d\geq 2$. Thus we have

Lemma 4 *Given the graph G_{block}, there is an algorithm that estimates with probability at least $\frac{3}{4}$ the weight of $\textsc{msf}(G_{block})$ to within a multiplicative relative error of $\delta+\varepsilon$. The algorithm requires $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ range queries and cone $(1+\delta)$-approximate nearest neighbor queries (for $\delta\leq\varepsilon/6$).* $\square$

6.6 Second level — estimating the weight of $\mathrm{MST}(G_{out})$

Let Q be the complete undirected graph with the vertex set B, the set of active blocks, and with the edge weights equal to the Euclidean distances between the corresponding block-centers *if* the blocks are in different connected block-components, and zero otherwise. Arguments similar in the spirit as those used in Observation 1 can be used to show the following lemma.

Lemma 5 $(1 - \varepsilon/2) \cdot \mathrm{EMST}(Q) \leq \mathrm{MST}(G_{out}) \leq (1 + \varepsilon/2) \cdot \mathrm{EMST}(Q)$.

Proof. Given $\mathrm{EMST}(Q)$, consider a subgraph $\widehat{G}$ of G_{out} defined as follows. Include in $\widehat{G}$ the MST of the subgraph in each connected block component of G_{out}; notice that the total weight of the MST is 0. Then, for each nonzero weight edge e in $\mathrm{EMST}(Q)$ connecting two connected block components, include in $\widehat{G}$ the lowest weight edge e' in G_{out} that connects the two block components joined by e. Due to the construction of Q, the weight of e is at least $\frac{\Gamma \cdot \Delta}{1+\varepsilon/4}$ and the weight of e' exceeds that of e by at most $\sqrt{d}\,\Delta$, which is less than an $\varepsilon/2$ fraction of the weight of e. Also note that any two connected block components in Q will be connected by at most one edge in $\mathrm{EMST}(Q)$, thus such an e' will always be found. It is easy now to see that $\widehat{G}$ is a spanning tree of $\mathrm{MST}(G_{out})$ of weight at most $\mathrm{EMST}(Q)(1+\varepsilon/2)$. Thus, $\mathrm{MST}(G_{out}) \leq (1 + \varepsilon/2) \cdot \mathrm{EMST}(Q)$.

To see the lower bound, we follow similar arguments. We start with $\mathrm{MST}(G_{out})$ and construct a spanning tree of Q in a similar fashion — with the exception that for an edge in $\mathrm{MST}(G_{out})$ joining two block components, we include an edge from Q which joins these components by linking two block centers — again with an increase in weight of at most an $\varepsilon/2$ fraction. Using similar argument as above, we can show that $(1 - \varepsilon/2) \cdot \mathrm{EMST}(Q) \leq \mathrm{MST}(G_{out})$, which concludes the proof. $\square$

In view of Lemma 5, to obtain a good estimation of the weight of $\mathrm{MST}(G_{out})$ it is sufficient to estimate the weight of a minimum spanning tree of Q.

We could find a minimum spanning tree of Q by calling any algorithm that finds a minimum spanning tree in graphs. However, any such algorithm requires time significantly more than $\mathcal{O}(b)$ time (at least for $d \geq 3$), because Q contains $\Theta(b^2)$ edges. To improve the running time to $\widetilde{\mathcal{O}}(b\,\varepsilon^{1-d}) = \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ we use $\mathcal{SPN}$, which is the $(1 + \varepsilon/4)$-spanner of B (having $\mathcal{O}(b\,(1/\varepsilon)^{d-1})$ edges) defined in Section 6.2. Let $\mathcal{F}$ be any spanning forest of the subgraph of Q induced by the edges of weight 0. It is easy to see that the weight of any minimum spanning tree of Q is identical to the weight of a minimum spanning tree of Q that uses the edges from $\mathcal{F}$.

We create a new graph SG with the vertex set B and the edge set which is the union of the edges in $\mathcal{F}$ and the spanner edges. Note that the number of edges in $\mathcal{F}$ is less than the number of spanner edges. Then, we apply, for instance, the classical Kruskal's algorithm to find in time $\mathcal{O}(b\,\varepsilon^{1-d}\,\log(b/\varepsilon^{d-1})) = \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ a minimum weight spanning tree of SG. It is easy to see that the obtained spanning tree of B is a spanning tree of B that uses edges from $\mathcal{F}$, since those edges are of weight 0 and do not form a cycle, and thus are bound to be picked by Kruskal's algorithm. Also, note that the resulting spanning tree's weight is at most $(1+\varepsilon/4)$ times the minimum, due to the properties of a spanner. This allows us to summarize the discussion in this section in the following lemma.

Lemma 6 *There is an algorithm which, given as input the graph G_{block}, estimates the weight of $\mathcal{M}$ to within a relative error of $\frac{3}{4}\,\varepsilon$ with running time $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$.*

Our analysis in this section can be slightly improved in the case where $d = 2$. In this case, one can simplify the arguments to achieve the running time of $\mathcal{O}(b\,\log b) = \mathcal{O}(\sqrt{n}\,\varepsilon^{-2}\,\log(\sqrt{n}/\varepsilon^2))$. The main difference is that for $d = 2$ we can find in $\mathcal{O}(b\,\log b)$ time a minimum spanning tree of Q using a modification of the classical algorithm due to Shamos and Hoey [22] for finding an Euclidean minimum spanning tree in $\mathbb{R}^2$. We first construct (in $\mathcal{O}(b \log b)$ time) the Delaunay triangulation of B. Next, we observe that there is a minimum spanning tree of Q that uses only edges of cost zero and edges from the Delaunay triangulation (for example, this follows from the proof of Lemma 1 in [14] or from [22]). Therefore, a minimum spanning tree of Q can be found by calling, for example, Kruskals algorithm on the subgraph of Q containing only the edges from the Delaunay triangulation of B and the edges of the forest $\mathcal{F}$ as defined above. Since the number of edges in the Delaunay triangulation and in $\mathcal{F}$ is $\mathcal{O}(b)$, this algorithm finds $\textsc{mst}(Q)$ in $\mathcal{O}(b \log b) = \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ time.

6.7 Estimating the weight of $\textsc{msf}(G_{block}) \cup \textsc{mst}(G_{out})$

We can now summarize our algorithm for estimating the EMST of any set of points in $\mathbb{R}^d$. We use the fact that $\mathcal{L} = \Theta(n/\varepsilon)$ and apply Lemmas 2, 4, and 6 to estimate the weight of the EMST. Summing up the error terms in our estimation stated in each one of those lemmas, we get that the multiplicative relative error is at most $\delta + 2\frac{1}{4}\varepsilon$ with probability at least $\frac{3}{4}$. Using $\varepsilon' = \varepsilon/3$ as the input parameter for our algorithm we can conclude with the following main theorem of the paper.

Theorem 2 *Let P be a set of n points in $\mathbb{R}^d$ for a constant d. Let ε be any real number, $0 < \varepsilon < \frac{1}{15}$, and let $\delta \le \varepsilon/4$. There is an algorithm that with probability at least $\frac{3}{4}$ estimates the weight of a Euclidean minimum spanning tree of P with a relative error of at most ε. This algorithm runs in $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ time and requires $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ orthogonal range queries, $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{2+d/2})$ cone $(1+\delta)$-approximate nearest neighbor queries, and a single minimal bounding cube of P.* □

Let us also mention that the remark at the end of Section 6.6 can be incorporated here to improve the complexity bounds in the most basic case when $d = 2$, that is, for the EMST problem on the Euclidean plane. Then, we obtain the following theorem.

Theorem 3 *Let P be a set of n points in $\mathbb{R}^2$. Let ε be any real number, $0 < \varepsilon < \frac{1}{15}$, and let $\delta \le \varepsilon/4$. There is an algorithm that, with probability at least $\frac{3}{4}$, estimates the weight of a Euclidean minimum spanning tree of P with a relative error of at most ε. This algorithm runs in $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ time and requires $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ orthogonal range queries, $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^2)$ cone $(1+\delta)$-approximate nearest neighbor queries, and a single minimal bounding cube of P.* □

Acknowledgements

We thank Bernard Chazelle and Sariel Har-Peled for helpful discussions on geometric data structures. We also thank the anonymous referees for their helpful comments.

References

[1] P. K. Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry*, pp. 575–598. CRC Press, Boca Raton, FL, 1997.

[2] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete & Computational Geometry*, 6:407–422, 1991.

[3] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pp. 1–56. AMS Press, 1999.

[4] S. Arya, D. M. Mount, and M. Smid. Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. *Discrete & Computational Geometry*, 13(2):91–107, 1999.

[5] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33–54, January 1997.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[7] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 291–300, 1993.

[8] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3): 703–724, August 1986.

[9] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3): 427–462, June 1988.

[10] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. In *Proceedings of the 28th Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 190–200, 2001.

[11] A. Czumaj and C. Sohler. Property testing with geometric queries. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, pp. 266–277, 2001.

[12] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 175–183, 2004.

[13] D. Z. Du and F. K. Hwang. Gilbert-Pollack conjecture on Steiner ratio is true. *Algorithmica*, 7:121–135, 1992.

[14] D. Eppstein. Spanning trees and spanners. In *Handbook of Computational Geometry*, pp. 425–461. Elsevier Science B.V., 1997.

[15] D. Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete & Computational Geometry*, 13(1):111–122, Jan 1995

[16] J. Erickson. On the relative complexity of some geometric problems. In *Proceedings of the 7th Canadian Conference on Computational Geometry (CCCG)*, pp. 85–90, 1995.

[17] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM Journal on Computing*, 31(5): 1479–1500, 2002.

[18] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 373–380, 2004.

[19] R. M. Karp and J. M. Steele. Probabilistic analysis of heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salemsan Problem*, chapter 6, pages 181–205. John Wiley & Sons, 1985.

[20] G. S. Lueker. A data structure for orthogonal range queries. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 28–34, 1978.

[21] J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry (CCCG)*, pp. 207–210, 1991.

[22] M. I. Shamos and D. Hoey. Closest-point problems. In *Proceedings of the 16th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 151–162, 1975.

[23] C. Sohler. *Property Testing and Geometry.* PhD. Dissertation, HNI-Verlagsschriftenreihe, Vol. 119, Heinz-Nixdorf Institut and University of Paderborn, 2003. Available from the author's web page at `http://www.upb.de/fachbereich/AG/agmadh/WWW/english/csohler/Diss.ps.gz`

[24] V. V. Vazirani. *Approximation Algorithms.* Springer-Verlag, Berlin, 2001.

[25] D. E. Willard. *Predicate-Oriented Database Search Algorithms.* PhD thesis, Harvard University, Aiken Computation Lab, Cambridge, MA, 1978. Report TR-20-78.

[26] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, November 1982.