Sublinear-Time Approximation of Euclidean Minimum Spanning Tree

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 situation when 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 increases, 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 we are not even given time to read all of the input data. 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 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.

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. 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 an $\mathcal{O}(n^{1/2})$-time algorithm for the 2-dimensional case which outputs a value w such that $\frac{1}{\alpha}\,\mathrm{EMST}(P) - \mathcal{L}n^{-c} \le w \le \alpha\,\mathrm{EMST}(P) + \mathcal{L}n^{-c}$, 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 algorithm that uses no more than $\mathcal{O}(n^{1/2})$ orthogonal range queries cannot significantly improve the quality of approximation.

We next consider the case when 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

[*]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.

[†]Electrical Engineering and Computer Science Department, Case Western Reserve University, Cleveland, OH 44107, USA. Email: afe@eecs.cwru.edu. This work was done while the author was at NEC Research, Princeton, NJ 08540, USA.

[‡]NEC Research, Princeton, NJ 08540, USA. Email:fortnow, ronitt@research.nj.nec.com

[§]Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4, Canada. Email: avner@cs.toronto.edu. 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.

[||]Heinz Nixdorf Institute and Department of Mathematics & Computer Science, University of Paderborn, D-33095 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).

neighbor data structures with some ideas from the recent sublinear-time algorithm for estimating the MST in general graphs [13]. The algorithm outputs a value which is within a $1+\varepsilon$ factor of the EMST and it runs in $\mathcal{O}(2^{\mathcal{O}(d)} \cdot \Lambda/\varepsilon^3)$ 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.

The third algorithm we present 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 needs $\widetilde{\mathcal{O}}(\sqrt{n}\,\text{poly}(1/\varepsilon))$ time and outputs an approximation of the EMST weight to within a multiplicative factor of $1+\varepsilon$. 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).

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, $\mathcal{O}(dn^2)$ time suffices, by reducing it to the MST problem in dense graphs. In the simplest case where $d = 2$ (on the plane), Shamos and Hoey [29] 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 $d = 3$ [17]. Yao [32] 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 [8] 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 algorithm of [13] that, given a connected graph in adjacency list representation with average degree d, edge weights in the range $[1 \ldots W]$, 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.

Dynamic algorithms. Our model of computation is also interesting in the context of dynamic algorithms. There exist fully dynamic algorithms that maintain EMST subject to point insertions and deletions; [16] 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.

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 $(1+\delta)$-*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 $|pq|$. 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 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 $C = [a_1, a_1 + R] \times [a_2, a_2 + R] \times \ldots \times [a_d, a_d + R]$ that contains P such that no axis-parallel cube of edge length smaller than R 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**.

Appendix A presents a discussion about the complexity of these data structures, which can be implemented efficiently by standard data structures. The cone nearest neighbor oracle, however, is more costly and less common than the cone approximate nearest neighbor oracle..

3 Estimating the EMST with bounding cube and range queries

Here we describe a natural approach to the approximation of 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 here the two dimensional case ($d = 2$). The algorithm we supply is deterministic, has running time $\mathcal{O}(n^{1/2})$ 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. 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.

[1]For a point $p \in P$ and a set of points $Q \subset \mathbb{R}^d$, 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 $|pq| \le (1+\delta) \cdot |px|$.

The quad-tree algorithm. We apply a standard quad-tree subdivision to the bounding cube $\mathcal{BC}$ (see, e.g., [7, 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 those nonempty blocks, and iterate this process as long as fewer than $\Theta(\sqrt{n})$ 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 (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 3.1. *$\frac{1}{\alpha} T^* - \beta \le w \le \alpha T^* + \beta$, where $\alpha = \mathcal{O}(n^{1/8} \log n)$, $\beta = \mathcal{L} n^{-c}$ and c is an arbitrary constant.*

Proof. (sketch) First note that the minimum spanning tree of any n points in a square with side-length h is $\mathcal{O}(h\sqrt{n})$ and this bound is tight. (e.g., when the points are uniformly spread (say, on the grid with edge length $h/\sqrt{n}$.)

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 $b \ge \sqrt{n}/(10 \log n)$; 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, $a \cdot b \cdot s \le L^* \le L \le \delta \cdot L^*$ for some constants a and δ.

Hence, as U is an upper bound on T^*, the approximation factor is $\alpha \le \frac{U}{w} = \sqrt{\frac{U}{L}} = \mathcal{O}((\frac{s\sqrt{b\,n}}{L})^{1/2})$ (where the last inequality follows by plugging in the expression for U and L). Now, by the above bound on L and on b we obtain that $\alpha \le \tilde{\mathcal{O}}(n^{1/8})$.

Assume now that $b < \sqrt{n}/(10 \log n)$, then it can be seen that the depth of the quad-tree is at least $10 \log n$ and hence $s \le \mathcal{L} \cdot n^{-10}$. Therefore, the additive term is bounded by $U - L \le \mathcal{O}(s \cdot \sqrt{b\,n}) = \mathcal{O}(n^{-10} \cdot \mathcal{L} \cdot n) = \mathcal{O}(n^{-9})$. $\square$

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 $\tilde{\mathcal{O}}(\sqrt{n})$ time ([29]), and this term dominates the total complexity.

As it turns out, the above quality of approximation is optimal for the given time bound as shown by the following claim whose proof is deferred to the full version of the paper.

CLAIM 3.2. *Any algorithm with $\mathcal{O}(\sqrt{n})$ orthogonal range queries has an approximation factor for* EMST(P) *of* $\Omega(n^{1/8})$.

Higher dimension. For fixed $d > 2$, the quad-tree can be replaced by a 2^d-ary tree. A total of $\mathcal{O}(2^d\sqrt{n})$ orthogonal range queries will be made in a similar way. Then $\mathcal{L}$ will be set as in the 2-dimensional case while $U = \mathcal{L} + s\, b^{1/d}\, n^{(d-1)/d}$. In this case we get an approximation with a multiplicative factor of $\alpha = \tilde{O}(n^{(d-1)/(4\,d)})$ and an additive term which is $\mathcal{L} \cdot n^{-c}$ for arbitrary constant c.

Finally, we note that our choice of using $\mathcal{O}(\sqrt{n})$ orthogonal range queries was arbitrary, in the sense that one can use a different number of queries and obtain a whole range of tradeoffs between the running time and the quality of approximation.

4 Two related previous results

We now describe two previous results that we utilize in our EMST algorithms: the concept of Yao graphs [32] and an algorithm for approximating the MST in bounded degree graphs due to Chazelle *et al.* [13].

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 $\theta = \mathcal{O}(1)$ this bound is $2^{\mathcal{O}(d)}$. Yao [32] 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 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 [32] motivates our use of these graphs.

CLAIM 4.1. [32] *For any point set $P \in \mathbb{R}^d$, 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 $\overrightarrow{py}$, $x, y \in C$.

Chazelle et al.: approximate MST in low-degree graphs. Our algorithms make use of a recent algorithm for estimating the MST in graphs due to Chazelle *et al.* [13]. This algorithm assumes that the input graph (i) is represented by an adjacency list, (ii) has degree at most ν (the full version of [13] allows ν to be the average degree), and (iii) has known minimum and maximum edge weights. 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 $\mathcal{O}(\nu \cdot \Lambda \cdot \log(\nu\,\Lambda/\varepsilon)/\varepsilon^3)$.

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 [13] is a procedure approx-number-connected-components 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 approx-number-connected-components 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 approx-number-connected-components runs in expected time $\mathcal{O}(\nu\,\varepsilon^{-2}\,\log(\Lambda/\varepsilon))$. Therefore, the expected running time of the algorithm in [13] 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 4.1 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 4.1, 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 5.1, being a special case of Claim 6.1, is omitted.

LEMMA 5.1. *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 approx-number-connected-components. 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 [13]. 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 5.1, we can conclude that X_u has identical distribution as in the algorithm of Chazelle *et al.* [13]. Therefore, the quality of this algorithm of the estimation of EMST of P is the same as in the algorithm of Chazelle *et al.* [13]. Since the maximum out-degree of the directed Yao graph is $2^{\mathcal{O}(d)}$, the modified procedure approx-number-connected-components has identical complexity as 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 lemma.

LEMMA 5.2. *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 $\Omega(\sqrt{n})$. Next we discuss a truly sublinear approximation algorithm whose complexity is independent of Λ.

6 Sublinear-time approximation algorithm

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*.

Overview of the algorithm. In Section 6.1, 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 together blocks that are "close" to each other. We call the resulting clusters *connected block-components*. The algorithm then proceeds in two phases. First, in Section 6.3, 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 6.4, approximate the optimal way to connect different connected block-components. We prove in Lemma 6.1 that the MSF of the connected block-components combined with the optimal set of edges joining them approximates the EMST of P.

6.1 Partitioning bounding cube into active blocks, and connected block-components.

After 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 the trivial lower bound EMST$(P) \geq n/\varepsilon$.

Partitioning the bounding cube into active blocks. 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. As before, b_k is the number of blocks at level k that contain points from P (*active blocks*), and $\Delta_k = \mathcal{L}/2^k$ is 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.

Spanners and connected block-components. For any $t \geq 1$, a *t-spanner* (see, e.g., [8, 15, 21]) 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 $t \cdot |xy|$.

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 $\mathcal{O}(b\log b + b\log(1/\varepsilon)\,\varepsilon^{-d}) = \widetilde{\mathcal{O}}(2^d\sqrt{n}\,\varepsilon^{1-d/2})$ [8].

Call two blocks *close* if the distance between their centers in $\mathcal{SPN}$ is at most $\Gamma\cdot\Delta$, where $\Gamma = 14\sqrt{d}/\varepsilon$.

We use equivalence classes 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 $\mathcal{O}(b\,(4/\varepsilon)^{d-1})$.

6.2 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 $\mathrm{MSF}(G)$, is a spanning forest of G having the 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 6.1. *Let p and q be an arbitrary pair of points in P.*

1. *If $|pq| \le (\Gamma - 4\sqrt{d})\,\Delta$ 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 of length at most $(\Gamma + \sqrt{d})\,\Delta = W$.*
3. *If $|pq| > (\Gamma + \sqrt{d})\,\Delta = W$, and p and q are in the same connected block-component, then $\mathrm{EMST}(P)$ does not contain the edge pq.* □

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 6.1, the connected components of $\mathrm{MSF}(G_{block})$ are identical to the connected block-components and the minimum spanning forest of these components is the same as $\mathrm{MSF}(G_{block})$.
- $\overline{G_\delta}$ is the *directed* $(1+\delta)$*-Yao graph* that is obtained from $\mathbb{K}_P$ using the cone $(1+\delta)$-approximate nearest neighbor oracle. Formally, we use the same definitions as in the definition of directed Yao graphs and 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 $\mathrm{MST}(G_{out})$ is identical to the weight of $\mathcal{M}$.

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

LEMMA 6.1. *The union of $\mathrm{MSF}(G_{block})$ and $\mathcal{M}$ is a spanning tree of $\mathbb{K}_P$ whose weight approximates the weight of $\mathrm{EMST}(P)$ to within a factor of $1+\varepsilon/2$.*

Proof. Clearly, the union of $\mathrm{MSF}(G_{block})$ and $\mathcal{M}$ forms a spanning tree of $\mathbb{K}_P$. To prove the second part of the lemma, 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 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 $\mathrm{MST}(G^*) \le (1+\varepsilon/2)\cdot\mathrm{EMST}(P)$. Notice further that by Observation 6.1 (1), each edge in G^* that is not in G_{block} has weight larger than $(\Gamma - 4\sqrt{d})\,\Delta$. Therefore, $\mathrm{MST}(G^*)$ can be obtained by first taking any minimum spanning forest induced by the edges in G_{block} and then taking some further edges in $\mathcal{M}$. Hence, the weight of the union of $\mathrm{MSF}(G_{block})$ and $\mathcal{M}$ is a $(1+\varepsilon/2)$ approximation to $\mathrm{EMST}(P)$. □

Consequently, to estimate $\mathrm{EMST}(P)$ it suffices to estimate the weights of $\mathrm{MSF}(G_{block})$ and $\mathrm{MST}(G_{out})$.

6.3 First level - estimating the weight of $\mathrm{MSF}(G_{block})$.

In this section we show how to estimate the weight of the MST within a single block component. This, combined for all block components, yields an estimate on the weight of $\mathrm{MSF}(G_{block})$. Since our model does not allow direct access to the edges of $\mathbb{K}_P$, we will use the directed Yao graph $\overline{G_\delta}$ to estimate the weight of $\mathrm{MSF}(G_{block})$. 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 $G_{block}^{(r)}$. Let n_u^r and m_u^r be the number of of vertices in $G_{block}^{(r)}$ and in $\overline{G_\delta}^{(r)}$ reachable from u, respectively. Notice that $c_r = \sum_{u\in P} 1/n_u^r$. Analogously, define $c_r^* = \sum_{u\in P} 1/m_u^r$. Also, let $\hat{c}_r$ be the number of connected components in $(1+\delta)\cdot G_{block}^{(r)}$.

It follows from [13] (see also Section 4) that

$$\text{(6.1)} \quad \mathrm{MSF}(G^{(r)}_{block}) \le n - r c_r + \sum_{i=1}^{r-1} c_i \le \mathrm{MSF}(G^{(r)}_{block}) + n.$$

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 6.1. *Let $\varepsilon \le \frac{1}{5}$ and $\delta \le \frac{1}{10}$. Then for every r, $n_u^{r/(1+\delta)} \le m_u^r \le n_u^r$. In particular, $c_{r/(1+\delta)} \ge c^*_r \ge c_r$.*

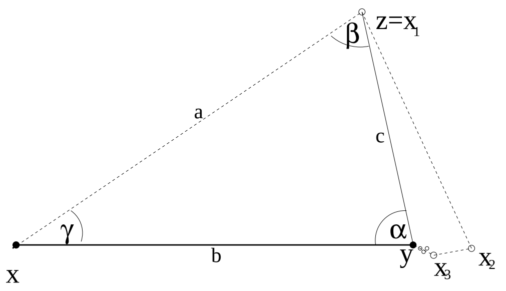

Illustration to the proof of Claim 6.1. The figure shows the reachability in $\overline{G_\delta}$. The dashed line is the path showing the connectivity of x and y in $\overline{G_\delta}^{((1+\delta)\,\tau)}$.

Proof. The right inequality clearly holds. To show the left inequality, it suffices to show that for every τ, if a vertex y is reachable in $G^{(\tau)}_{block}$ 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 $G^{(\tau)}_{block}$; 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 $G^{(\tau)}_{block} = G^{(W)}_{block}$).

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 lemma holds. So let us assume that $z \ne y$. Let $a = |xz|$, $b = |xy|$, $c = |yz|$, and $\alpha = \measuredangle(xyz)$, $\beta = \measuredangle(xzy)$, and $\gamma = \measuredangle(yxz)$. 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 $a > b/(1+\varepsilon) \ge \frac{\sqrt{2}}{2} \cdot b$, the expression $a^2 + b^2 - \sqrt{2}\,a\,b$ increases with a. Therefore, by inequality (i) we obtain

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

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

Now we prove the lemma using inequalities (i–iii). Assume, without loss of generality, that $|xy| \le \tau$; otherwise apply the following arguments to all edges on the path between x and y in $G^{(\tau)}_{block}$ (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 $x_i \ne y$, 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 $|xy| \le \tau \le W = (\Gamma + \sqrt{d}) \cdot \Delta$, we obtain

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

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 6.1, 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 $|x_i y| \le |xy|$, we obtain $|x_i x_{i+1}| \le (1+\delta)\,|xy|$. 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 $\overline{G_\delta}^{((1+\delta)\,\tau)}$. □

We introduce an estimator $\mathcal{A}$ for the value of $\mathrm{MSF}(G_{block})$.

$$\mathcal{A} = n + \sum_{i=1}^{\lceil W\,(1+\delta)\rceil - 1} c^*_i - \lceil W\,(1+\delta)\rceil \cdot c^*_{\lceil W\,(1+\delta)\rceil} \;.$$

We analyze now the quality of this estimator.

LEMMA 6.2. $\mathrm{MSF}(G_{block}) \le \mathcal{A} \le (1+\delta)\mathrm{MSF}(G_{block}) + n.$

Proof. We start with the following two observations.

- If $r \geq W$ then $c_r = c_W$. As a corollary, $c^*_{\lceil W\,(1+\delta)\rceil} = c_W = c_{\lceil W\,(1+\delta)\rceil}$.
- $\hat{c}_r = c_{r/(1+\delta)}$.

Now, we have the following sequence of inequalities:

$$
\begin{aligned}
\text{MSF}(G_{block}) \leq\ & n + \sum_{i=1}^{W-1} c_i - W\,c_W \\
=\ & n + \sum_{i=1}^{\lceil W\,(1+\delta)\rceil-1} c_i - \lceil W\,(1+\delta)\rceil \cdot c_{\lceil W\,(1+\delta)\rceil} \\
=\ & n + \sum_{i=1}^{\lceil W\,(1+\delta)\rceil-1} c_i - \lceil W\,(1+\delta)\rceil \cdot c^*_{\lceil W\,(1+\delta)\rceil} \\
\leq\ & n + \sum_{i=1}^{\lceil W\,(1+\delta)\rceil-1} c^*_i - \lceil W\,(1+\delta)\rceil \cdot c^*_{\lceil W\,(1+\delta)\rceil} \\
=\ & \mathcal{A} \\
\leq\ & n + \sum_{i=1}^{\lceil W\,(1+\delta)\rceil-1} c_{i/(1+\delta)} - \lceil W\,(1+\delta)\rceil \cdot c_W \\
=\ & n + \sum_{i=1}^{\lceil W\,(1+\delta)\rceil-1} \hat{c}_i - \lceil W\,(1+\delta)\rceil \cdot \hat{c}_{\lceil W\,(1+\delta)\rceil} \\
\leq\ & \text{MSF}((1+\delta)\cdot G_{block}) + n \\
=\ & (1+\delta)\cdot \text{MSF}(G_{block}) + n\ .
\end{aligned}
$$

The first inequality is due to inequality (6.1). The first equality follows from the first observation above. Next, we use Corollary 6.1 and the both observations above. The last inequality is implied by inequality (6.1). □

We now modify the algorithm of Chazelle *et al.* [13] to obtain a good approximation of $\mathcal{A}$. Following the approach from Section 5, we run procedure approx-number-connected-components to estimate the number of connected components of the graph with edges of weight at most $r = (\frac{1}{2} + i)\,\varepsilon$ for $i = 1, 2, \ldots, \Lambda/\varepsilon$ (where $\Lambda = \lceil W\,(1+\delta)\rceil$ is an upper bound of the weights in the graph). This estimation is achieved using the estimation of $1/n^r_u$. As was discussed in Section 5, if there is an efficient way to traverse $\overline{G_\delta}$, we can modify the algorithm to apply this traversal and we get an estimation of the c^*_r with the same probabilistic guarantees. As was pointed in Section 5, traversing in $\overline{G_\delta}^{(r)}$ is easy: 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 $|pq| \leq r$ and if the blocks to which p and q belong are in the same connected block-component. The first test is a simple $2^{\mathcal{O}(d)}$ time calculation, while the second requires the computation of the connected block-components. This is done in a preprocessing step as described in Section 6.1. Therefore, following the analysis from [13] (see also Section 5) we can estimate the value of $\sum_{r=1}^{\lceil W\,(1+\delta)\rceil-1} c^*_r$ using $\widetilde{\mathcal{O}}(W \cdot 2^{\mathcal{O}(d)}\varepsilon^{-3})$ cone approximate nearest neighbor queries and $\widetilde{\mathcal{O}}(b \cdot \varepsilon^{-d+1})$ orthogonal range queries. The algorithm approximates $\sum_{r=1}^{\lceil W\,(1+\delta)\rceil-1} c^*_r$ to within an additive error of $\frac{1}{9}\,n$ with probability at least $\frac{3}{4}$ (see [13]). Next, observe that $c^*_{\lceil W\,(1+\delta)\rceil}$ is nothing but the number of connected block-components, which is known to the algorithm that computes the connected block-components. Therefore, together with Lemma 6.2 we get an algorithm that approximates $\text{MSF}(G_{block})$ to within an additive factor of $\frac{10}{9}n$.

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 $\widetilde{\mathcal{O}}(W/(\lambda\,\varepsilon^3))$ cone approximate nearest neighbor queries and achieve an additive error of $\frac{10}{9}\,\lambda\,n$.

Let us examine the term W/λ in the running time and the additive error term $\frac{10}{9}\,\lambda\,n$. 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 $\text{EMST}(P) \geq \frac{1}{2}\Delta(\lceil b/2^d\rceil - 1)$ [3]. Since $b \geq b^* \geq 2^{d+1}$ we get $EMST(P) \geq \frac{1}{4}\Delta \cdot b/2^d$. Setting $\lambda = \frac{9}{40} \cdot \frac{\Delta b \varepsilon}{2^d n}$ and applying the above scaling procedure, we upper bound the multiplicative error by

$$1 + \delta + \frac{\frac{10}{9}\,\lambda\,n}{\frac{1}{4}\Delta \cdot b/2^d} = 1 + \delta + \varepsilon\ .$$

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

$$W/(\lambda\,\varepsilon^3)\ =\ \widetilde{\mathcal{O}}(\sqrt{d}2^d \cdot n/(b\varepsilon^5)) \leq \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})\ .$$

On the other hand, when $\Delta < 2\varepsilon$, we use the trivial lower bound $\text{EMST}(P) \geq n/\varepsilon$, and by setting $\lambda = 9/10$ obtain a multiplicative error of $1 + \delta + \varepsilon$. In this case notice that $W = \Delta\sqrt{d}(14/\varepsilon + 1) = O(1)$. And so we bound the running time by

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

for $d \geq 2$. Thus we have the following lemma.

LEMMA 6.3. *Given the graph G_{block}, there is an algorithm that estimates with probability at least $\frac{3}{4}$ the weight of*

[3]This bound is achieved by considering a subdivision of the active block to $2^d b$ subcubes of size $\Delta/2$. Now colour these subdivisions using 2^d colours, using the same arrangement of colours 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 coloured the same. Clearly, the minimal distance between these points must be at least $\Delta/2$, and hence the bound.

$\textsc{msf}(G_{block})$ *to within a multiplicative relative error of* $\delta+\varepsilon$. *The algorithm requires* $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})$ *range queries and cone* $(1+\delta)$*-approximate nearest neighbor queries (for* $\delta = \varepsilon/6$*).* □

6.4 Second level — estimating the weight of $\textsc{mst}(G_{out})$. 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 spirit to Observation 6.1 can be used to show that $1-\varepsilon/2 \le \textsc{mst}(G_{out})/\textsc{emst}(Q) \le 1+\varepsilon/2$. To obtain a good estimation of the weight of $\textsc{mst}(G_{out})$ we therefore 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 $\Omega(b^2)$, because Q contains $\Theta(b^2)$ edges. To improve the running time to $\widetilde{\mathcal{O}}(b\,\varepsilon^{1-d}) = \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})$ 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 6.1. 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. Then, apply, for example, the classical Kruskal's algorithm to find in time $\mathcal{O}(b\,\varepsilon^{1-d}\log(b/\varepsilon^d)) = \widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})$ a minimum weight spanning tree of SG. It is easy to see now that the obtained spanning tree of B is a spanning tree of B that uses edges from $\mathcal{F}$ and whose weight is at most $\frac{1}{4}\varepsilon$ times greater than the minimum. We summarize the discussion in this section in the following lemma.

LEMMA 6.4. *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* $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})$*.*

Our entire analysis can be improved in the case $d=2$. In this case, one simplify our arguments to achieve the the running time of $\mathcal{O}(b\log b) = \mathcal{O}(\sqrt{n}\log(\sqrt{n}/\varepsilon)/\varepsilon)$.

6.5 Estimating the weight of $\textsc{msf}(G_{block}) \cup \textsc{mst}(G_{out})$. Now, we can summarize our algorithm for estimating the $\textsc{emst}$ of any set of points in $\mathbb{R}^d$. Since $\mathcal{L} = \Theta(n/\varepsilon)$, by Lemmas 6.1, 6.3, and 6.4, we obtain the following theorem. Summing up the errors made in our estimation 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 input parameter for our algorithm we get:

THEOREM 6.1. *Let* P *be a set of* n *points in* $\mathbb{R}^d$ *for a constant* d*. Let* ε *be any real number,* $0 < \varepsilon < \frac{1}{2}$*, 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*. It runs in* $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})$ *time and requires* $\widetilde{\mathcal{O}}(\sqrt{n}\cdot\varepsilon^{d/2+2})$ *orthogonal range queries,* $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{d/2+2})$ *cone* $(1+\delta)$*-approximate nearest neighbor queries, and a single minimal bounding cube of* P*.* □

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, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: Short, thin, and lanky. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pp. 489–498, 1995.

[5] 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.

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

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

[8] 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*, pp. 291–300, 1993.

[9] B. Chazelle. Filtering Search: A new approach to query-answering *SIAM Journal on Computing*, 15:703–724, 1986.

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

[11] B. Chazelle. Lower bounds for orthogonal range searching: I. The reporting case. *Journal of the ACM*, 37(2):200–212, April 1990.

[12] B. Chazelle. Lower bounds for orthogonal range searching: II. The arithmetic model. *Journal of the ACM*, 37(3):439–463, June 1990.

[13] 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*, pp. 190–200, 2001.

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

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

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

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

[18] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[19] M. Fischer, F. Meyer auf der Heide, and W.-B. Strothmann. Dynamic data structures for realtime management of large geormetric scences. In *Proceedings of the 5th Annual European Symposium on Algorithms*, pp. 157–170, 1997.

[20] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, July 1998.

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

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

[23] J. Matoušek. Range searching with efficient hierarchical cutting. In *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, pp. 276–285, 1992.

[24] J. S. B. Mitchell and J. O'Rourke. Computational geometry column 42. *SIGACT News* 32(3):63–72, 2001.

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

[26] J. S. Salowe. Constructing multidimensional spanner graphs. *International Journal of Computational Geometry and Applications*, 1(2):99-107, 1991.

[27] H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.

[28] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

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

[30] P. M. Vaidya. Minimum spanning trees in k-dimensional space. *SIAM Journal on Computing*, 17(3):572–582, 1988.

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

[32] 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.

Appendix

A 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 $\mathcal{O}(d)$.

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 a survey, see, e.g., [1, 3, 7]). 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. The best known data structure for orthogonal range searching based on compressed range trees and some other techniques such as filtering search ([11, 12]). The time for a query is $O(\log^{d-1} n)$. If one uses standard range trees with the fractional cascading technique then the same bound on the query time can be achieved ([22, 31]).

Cone nearest neighbor oracle. In his seminal paper on Euclidean minimum spanning trees, Yao [32] 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., [5, 6, 15, 25]). And so, among others, Ruppert and Seidel [25] show how to answer a query in amortized time $\mathcal{O}(n \log^{d-1} n)$ per each cone in C; a similar construction is presented in [6]. Arya *et. al.* [5] 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, 7]).